Number Theory in Cryptography

Lecture notes for MATH-489

Last update: May 14, 2023

Contents

1 Arithmetic with Large Integers **9**

1.1 Complexity of algorithms . . . 9

1.1.1 Big-$\mathcal{O}$-notation . . . 9

1.1.2 Remarks on big-$\mathcal{O}$ notation to keep in mind . . . 10

1.2 Schoolbook, Karatsuba and Toom–Cook multiplication . . . 10

1.2.1 Schoolbook multiplication . . . 10

1.2.2 Karatsuba multiplication . . . 11

1.2.3 Toom–Cook multiplication . . . 11

1.3 Fast Fourier Transform (FFT) methods . . . 12

1.3.1 Version for polynomials . . . 13

1.3.2 Cooley–Tukey algorithm . . . 15

1.3.3 Multiplying large integers - the method of Schönhage–Strassen 16

1.4 Recent developments and summary . . . 19

1.5 Exercises . . . 20

2 Exponentiation Algorithms **21**

2.1 Basic Binary Ladders . . . 21

2.2 Window exponentiation . . . 22

2.3 Fixed-base ladders . . . 23

2.4 Addition/Lucas chains/Montgomery ladder . . . 24

2.5 Exercises . . . 25

3 Modular and Finite Field Arithmetic **27**

3.1 Congruences and Modular Arithmetic . . . 27

3.1.1 Theorems of Fermat and Euler . . . 27

3.2 Newton Methods - Barrett's Algorithm . . . 28

3.3 Montgomery Arithmetic . . . 29

3.3.1 Montgomery multiplication . . . 30

3.3.2 Montgomery exponentiation . . . 30

3.3.3 Montgomery reduction and Montgomery arithmetic . . . 30

3.3.4 Concurrent Montgomery multiplication . . . 31

3.4 Finite Field Arithmetic . . . 33

3.4.1 Basics on finite fields and finite field extensions . . . 34

3.4.2 Irreducible polynomials over $\mathbf{F}_p$ . . . 35

3.4.3 Normal bases 35
3.4.4 Inversion using normal bases 36
3.4.5 Examples of normal bases 36
3.5 Exercises 37

4 Factoring Polynomials over Finite Fields **41**
4.1 Polynomial rings and quotient rings of polynomial rings 41
4.1.1 Commutative rings with unity 41
4.1.2 Polynomial rings and quotient rings of polynomial rings 42
4.1.3 The Chinese remainder theorem for quotient rings of polynomial rings 42
4.2 Factorization of polynomials 42
4.2.1 Testing polynomials for repeated factors 43
4.2.2 A brief summary of Berlekamp's algorithm 43
4.2.3 A brief summary of the algorithm of Cantor–Zassenhaus 46

5 Integer Factorization **49**
5.1 Exponential-time versus subexponential-time algorithms 49
5.1.1 Trial division 49
5.1.2 Pollard's $p-1$-method 49
5.1.3 The L-notation for complexity 50
5.2 Methods based on $x^2 \equiv y^2 \bmod n$ 50
5.2.1 Fermat's factorization method 50
5.2.2 Dixon's Method and Morrison–Brillhart Approach 50
5.2.3 Smoothness bounds and smoothness probabilities 51
5.2.4 Analysis of Dixon's method 52
5.3 Basics sieving methods - linear and quadratic sieves 53
5.3.1 Linear sieve 53
5.3.2 Quadratic sieve 54
5.4 Number Field Sieve 54
5.4.1 The idea of the general number field sieve 54
5.4.2 The choice of m and f 55
5.4.3 The norm map $N\colon \mathbf{Q}(\alpha) \to \mathbf{Q}$ 56
5.4.4 Several remarks on sieving 56
5.4.5 Several remarks on the “sufficiency of the squareness of the norm" 57
5.4.6 On taking square roots 59
5.4.7 On the complexity analysis 59
5.4.8 More details on the run-time analysis 60

6 The Discrete Logarithm Problem over Finite Fields **61**
6.1 Introduction to the Discrete Logarithm Problem 61
6.2 The Pollard's rho method 62
6.2.1 Description of the algorithm 62
6.2.2 Complexity analysis of Pollard rho 63
6.3 Index calculus for prime fields and for extension fields 64

6.3.1 Discrete logarithm over prime fields 64
6.3.2 Discrete logarithm over extension fields 65
6.4 Applications of discrete log 66
6.4.1 ElGamal encryption 66
6.4.2 Diffie–Hellman key exchange 66

7 Introduction to Elliptic Curves 69
7.1 Why elliptic curve groups or groups of points on Jacobians of curves? 69
7.1.1 Practical implications 69
7.2 Introduction to elliptic curves 70
7.2.1 Weierstrass equations 70
7.2.2 The group law 71
7.2.3 Addition and doubling formulae 72
7.3 Elliptic curves over finite fields and Hasse–Weil bounds 73

8 Elliptic Curve Cryptography 77
8.1 Elliptic Curve Cryptosystems 77
8.1.1 Scalar multiplication of points 77
8.1.2 The elliptic curve discrete logarithm problem 77
8.1.3 Analogue of Diffie–Hellman key exchange 77
8.1.4 Analogue of ElGamal encryption 78
8.1.5 Elliptic curve digital signature algorithm (ECDSA) 78
8.1.6 Encoding plaintexts into points on elliptic curves 78
8.2 Elliptic Curve Factorisation Algorithm 79
8.2.1 Point addition and point doubling formulas 79
8.2.2 Arithmetic with points over $\mathbf{Z}/n\mathbf{Z}$ 79
8.2.3 Lenstra's ECM method 80
8.3 Elliptic Curve Primality Testing 81
8.3.1 Pocklington's test 81
8.3.2 The algorithm of Goldwasser–Killian–Atkin 82

9 Lattice-basis Reduction Algorithms 85
9.1 History of Lattices 85
9.2 Lattices and Lattice-based Cryptography 85
9.2.1 Lattices in $\mathbf{R}^n$ 85
9.2.2 The discriminant of a lattice 86
9.2.3 Theory of short vectors in lattices and Minkowski's theorems . 86
9.2.4 Lattice-based cryptography 88
9.3 The LLL Algorithm 88
9.3.1 The case of $\dim(\mathcal{L}) = 2$ 88
9.3.2 The notion of a reduced basis 88
9.3.3 The algorithm 90
9.3.4 Correctness 90
9.3.5 Termination 90
9.3.6 Complexity analysis 91

9.4 Applications . . . 91
9.4.1 Basic number theoretic applications . . . 91
9.4.2 Finding exact shortest vectors . . . 92
9.4.3 Attack on RSA with low public exponent . . . 93
9.4.4 Approximate GCD problem. . . . 96
9.5 The NTRU Cryptosystem . . . 97
9.5.1 Key generation . . . 97
9.5.2 Encryption . . . 97
9.5.3 Decryption . . . 97
9.5.4 Correctness . . . 97
9.5.5 Analysis . . . 98
9.6 Block Korkine–Zolotarev (BKZ) Algorithm . . . 98

10 Fully Homomorphic Encryption 101
10.1 Homomorphic encryption schemes - basic notions . . . 101
10.2 Additively homomorphic encryption schemes . . . 101
10.2.1 Paillier's scheme . . . 101
10.2.2 The scheme of Damgaard–Jurik . . . 102
10.3 Learning with errors (LWE) problem . . . 102
10.3.1 Learning parity with noise (LPN) problem . . . 102
10.3.2 LWE problem . . . 103
10.3.3 Decisional LWE problem . . . 105
10.3.4 Reductions . . . 105
10.3.5 Quantum reduction to approximate SVP . . . 106
10.3.6 Regev's public-key cryptosystem . . . 106
10.4 Ring Learning with Errors (`Ring-LWE`) Problem . . . 106
10.4.1 Motivation . . . 106
10.4.2 Formulation and basic properties . . . 107
10.4.3 Encryption schemes based on `RingLWE` . . . 107
10.5 Fully homomorphic encryption schemes . . . 108
10.5.1 Historical overview . . . 108
10.5.2 Gentry's original idea . . . 108
10.5.3 Bootstrapping . . . 109
10.5.4 Modulus and key switching . . . 109
10.5.5 The Brakerski–Fan–Vercauteren (BFV) scheme . . . 110
10.5.6 BGV scheme . . . 112
10.5.7 Cheon–Kim–Kim–Song (CKKS) scheme . . . 112
10.5.8 The GSW scheme . . . 112
10.5.9 TFHE scheme . . . 114

Bibliography 115

Introduction to the Course

Cryptography is a science that studies techniques for secure communication between parties in the presence of adversaries. Unlike cryptography from the ancient times (Roman age; Caeser cipher) and the machine age (WWII; Enigma), modern cryptography is built upon mathematical tools and paradoxes. It is thus based on various areas of mathematics such as abstract algebra, elementary and advanced number theory, probability theory and statistics as well as complexity theory and algorithms.

Cryptographic objectives

Cryptography aims to achieve the following objectives:

- **Data security:** data sent from A (Alice) to B (Bob) is invisible to an adversary E (Eve).
- **Data integrity:** data is accurate and consistent over its lifecycle.
- **Authenticity:** the truth of an attribute of data claimed by an entity (certificate).

Symmetric versus Public Key Cryptography

The two major branches of cryptography are symmetric key cryptography and public key cryptography. Symmetric key cryptography assumes that the sending and receiving parties share a secret key used to both encrypt and decrypt data. Encryption of large data is typically done via this method. The challenge and need for public key cryptography is that, e.g., we need mechanisms to securely share secret keys between parties (for the purposes of symmetric key cryptography), we need to be able to do authentication (digitally sign documents and verify the signatures using only publicly known data) and much more!

Public key cryptography is built upon various constructions from elementary and more advanced number theory. As such, the algorithms are significantly slower than those from symmetric key cryptography (you will never encrypt a large document with RSA 2048, but rather use AES-256). Moreover, the cryptanalysis of the main algorithms from PKC is a very challenging task and requires an in-depth mathematical understanding of the structures involved.

The goal of the course

This course is entirely on public key cryptography and more specifically, on the way modern number theory is used to achieve some of the objectives of that area. The goal is to review various mathematical structures such as finite fields, number fields, elliptic curves and lattices from a computational perspective. It is thus mainly algorithmic and aims at giving a basic background on public key cryptanalysis.

Public key cryptanalysis

This course will focus less on the design of cryptographic protocols and algorithms. It will rather focus on the analysis of the existing and widely used ones via more advanced cryptanalytic techniques coming from number theory. The two major problems (hardness assumptions) in public key cryptography are the following:

1. **Integer factorization:** given a large integer N, efficiently factor N.
2. **Computing discrete logarithms:** given a large finite field $\mathbf{F}_q$, a generator g of the multiplicative group $\mathbf{F}_q^\times$ and an element $h = g^x$, find the secret exponent x.

Nowadays, more and more work has been done on the efficient solution of *ii*) for finite fields. Good alternatives to finite fields are provided by the (abelian) group of points on an elliptic curve over a finite field (where the size of the group can be chosen to be much smaller without compromising security).

Apart from understanding various methods and algorithms for solving i) and ii), we will introduce the basic theory of elliptic curves over finite fields and show how the latter are used in cryptographic applications.

The final topic of the course will be lattice-based cryptography - an active area of research with increasingly important applications in both cryptanalysis and protocols. Besides fundamental applications of lattice-basis reduction algorithms to public key cryptanalysis, lattices and hardness problems for lattices are used for the design of various modern public key encryption schemes. If time permits, we will dive deep into one of these types of schemes - Fully Homomorphic Encryption (FHE) schemes allowing for arbitrary computations on encrypted data without decrypting it, the latter becoming increasingly important for secure computations in the cloud.

CHAPTER 1 Arithmetic with Large Integers

The goal of this and the next lecture is to explain how computers and other hardware platforms perform basic operations with large integers in the context of public key cryptographic protocols. On a hardware platform, non-negative integers will be represented in base B where B is typically a power of 2 (e.g., B could be 2^{32} for a 32-bit architecture). Each integer n is then represented by

$$n = \sum_{i=0}^{s-1} n_i B^i, \qquad 0 \leq n_i \leq B-1.$$

Here, the n_i's are blocks, the least significant one being n_0 and the most significant one being n_{s-1}.

1.1 Complexity of algorithms

Often, we will need to give mathematical estimates for the complexity of various algorithms. We do that via a special notation called big-$\mathcal{O}$ notation that we now introduce.

1.1.1 Big-$\mathcal{O}$-notation

Suppose that $f, g \colon \mathbf{Z}_{>0} \to \mathbf{R}$ are two functions of the positive integers n that take real values. We say that $g(n) = \mathcal{O}(f(n))$ if there exist constants $C, N > 0$ such that $|g(n)| \leq C|f(n)|$ for all $n \geq N$. Moreover, we say that $g(n) = \Theta(f(n))$ if $g(n) = \mathcal{O}(f(n))$ and $f(n) = \mathcal{O}(g(n))$. Very often, we will give the run-time of an algorithm via this notation. We will write $g(n) = o(f(n))$ if $f(n) \neq 0$ for all large enough n and $g(n)/f(n)$ tends to 0 as $n \to +\infty$. Finally, we write $f(n) \sim g(n)$ if $f(n) \neq 0$ for all large enough n and $g(n)/f(n)$ tends to 1 as $n \to +\infty$.

As an example, $2n^2 + n = \mathcal{O}(n^2)$ since $2n^2 + n \leq 3n^2$ for all $n \geq 1$. Moreover, $2n^2 + n = \Theta(n^2)$, but $n \neq \Theta(n^2)$ since $n^2 \neq \mathcal{O}(n)$ even if $n = \mathcal{O}(n^2)$. As we see examples of algorithms, we will get more and more used to this notation.

1.1.2 Remarks on big-$\mathcal{O}$ notation to keep in mind

Measuring run-time of various algorithms is a subtle question. The big-$\mathcal{O}$ notation is one kind of asymptotic notation, an idea from mathematics that describes the behavior of functions *in the limit*.

In practice, defining how long an algorithm takes to run is difficult: one cannot usually give an answer in milliseconds because of the dependency on the machine architecture. One can neither give an answer in CPU clock cycles (or as an operation count) because that would be too specific to the particular data to be useful.

The simple way of looking at asymptotic notation is that it discards all the constant factors in a function. In other words, an^2 will always be bigger that bn if n is sufficiently large (assuming everything is positive). Changing the constant factors a and b does not change that - it changes the specific value of n where n^2 is bigger, but does not change that it happens. So we say that $\mathcal{O}(n^2)$ is asymptotically larger than $\mathcal{O}(n)$, and forget about those constants that we probably do not know anyway. That is useful because the problems with large n are usually the ones where things slow down enough that we really care. If n is small enough, the time taken is small and the gains available from choosing different algorithms are small. When n gets large, choosing a different algorithm can thus make a huge difference.

The big-$\mathcal{O}$ notation is a useful mathematical model that abstracts away enough awkward-to-handle details that useful results can be found, but it is certainly not a perfect measure for the complexity of algorithms. We do not deal with infinite problems in practice and there are plenty of times when problems are small enough that those constants are relevant for real-world performance and sometimes you just have to time things with a clock rather than asymptotically.

1.2 Schoolbook, Karatsuba and Toom–Cook multiplication

1.2.1 Schoolbook multiplication

Let us recall the basic multiplication of integers that we learned in school: to compute $123 \cdot 323$, we do

$$\begin{array}{ccccc}
 & & 3 & 6 & 9 \\
 & 2 & 4 & 6 & \\
3 & 6 & 9 & & \\
\hline
3 & 9 & 7 & 2 & 9
\end{array}$$

Simple as that, we will try to analyze the run-time and express it in big-$\mathcal{O}$ notation. Suppose first that the two numbers have n digits each. To compute each of the rows, we need n single digit multiplications, a total of n^2 single digit multiplications. To compute the final result out of these, we need to add n integers of at most $2n$ digits each. This gives us a total of $3n^2$ elementary operations, i.e., the run-time of the algorithm is $\mathcal{O}(n^2)$. Note that in this estimate, we are, for example, not accounting for the time it takes to shift an integer to the left. Yet, in computers, bit shifts are typically considered very fast operations taking almost negligible amount of time.

1.2.2 Karatsuba multiplication

Let x and y be two base-b numbers of n digits each (in their b-base representation). For simplicity, assume that n is even, i.e., $n = 2m$. Write $x = x_0 + b^m x_1$ and $y = y_0 + b^m y_1$. Then

$$xy = x_0 y_0 + (x_0 y_1 + x_1 y_0) b^m + x_1 y_1 b^{2m}.$$

To compute xy, it thus suffices to perform the four multiplications x_0y_0, x_1y_0, x_0y_1 and x_1y_1. Karatsuba [KO63] observed that one only needs to do three multiplications: $(x_0 + x_1)(y_0 + y_1)$, x_0y_0 and x_1y_1, and perform four extra additions in order to get the above product, since

$$x_0y_1 + x_1y_0 = (x_0 + x_1)(y_0 + y_1) - x_0y_0 - x_1y_1.$$

Applying this algorithm recursively, one computes the product xy in $\mathcal{O}(n^{\log_2 3})$ elementary steps (you will see in the homework exercise why this is the run time). This is asymptotically faster than the schoolbook algorithm (in fact, $\log_2 3 \sim 1,585 < 2$).

1.2.3 Toom–Cook multiplication

Toom–Cook's method is based on the observation that, given two polynomials of degree d (for some d),

$$p(x) = p_0 + p_1 x + \cdots + p_{d-1} x^{d-1}$$

and

$$q(x) = q_0 + q_1 x + \cdots + q_{d-1} x^{d-1},$$

the product polynomial $h(x) = p(x)q(x) = h_0 + h_1 x + \cdots + h_{2d-2} x^{2d-2}$ is completely determined by the values at $2d - 1$ distinct points, e.g., at $t = -d + 1, -n + 2, \ldots, 0, 1, \ldots, d - 1$. If we now think of each polynomial as the base-b expansion of an n-bit number, we can turn this idea into an algorithm for multiplying integers.

We illustrate this with an example: consider $d = 3$ and write the symbolic expressions

$$\begin{aligned}
r_{-2} &= (p_0 - 2p_1 + 4p_2)(q_0 - 2q_1 + 4q_2)\\
r_{-1} &= (p_0 - p_1 + p_2)(q_0 - q_1 + q_2)\\
r_0 &= p_0 q_0\\
r_1 &= (p_0 + p_1 + p_2)(q_0 + q_1 + q_2)\\
r_2 &= (p_0 + 2p_1 + 4p_2)(q_0 + 2q_1 + 4q_2).
\end{aligned}$$

But now, one can use symbolic linear algebra to express the coefficients of $h(x)$ in terms of the r_t's (think of this as a precomputation, depending only on the parameter

d):

$$\begin{aligned}
h_0 &= r_0 \\
h_1 &= r_{-2}/12 - 2r_{-1}/3 + 2r_1/3 - r_2/12 \\
h_2 &= -r_{-2}/24 + 2r_{-1}/3 - 5r_0/4 + 2r_1/3 - r_2/24 \\
h_3 &= -r_{-2}/12 + r_{-1}/6 - r_1/6 + r_2/12 \\
h_4 &= r_{-2}/24 - r_{-1}/6 + r_0/4 - r_1/6 + r_2/24.
\end{aligned}$$

Now, to multiply two base-b numbers $x = x_0 + x_1 b + x_2 b^2$ and $y = y_0 + y_1 b + y_2 b^2$, we compute the corresponding r_t's by performing the corresponding additions and 5 multiplications. We then compute the h_i's above for $i = 0, \dots, 4$. Note that these h_i's are not yet the base-b digits (they need not be in $[0, b-1)$). We adjust the carries by a classical procedure known as *carrying*.

To compute the gain in the above procedure, we compare to Karatsuba's algorithm where we multiplied two size b^2 numbers using 3 instead of 4 multiplications of size b numbers. Here, we multiply two size b^3 numbers using 5 instead of 9 multiplication, so the gain factor is $9/5$. If we recursively use the same Toom–Cook procedure to multiply the smaller-size numbers in the computation of the r_t's, we obtain a total of $\mathcal{O}(n^{\log_3 5})$ small size multiplications (here, by small, we mean multiplications of constant size independent of n). Since $\log_3 5 < \log_2 3$ (in Karatsuba's run-time), the Toom–Cook algorithm runs asymptotically faster in our case.

We can generalize the above procedure for any degree d: there are $2d - 1$ multiplications in the computations of $r_{-d+1}, \dots, r_{d-1}$ and again, by a recursive application, we obtain a total of $\mathcal{O}(n^{\log_d(2d-1)})$ small multiplications. We can thus bring the complexity to $\mathcal{O}(n^{1+\varepsilon})$ for any positive number $\varepsilon > 0$.

Note, however, that this is just an asymptotic analysis and is far from what one expects in practice - in particular, the simple analysis above ignores completely the bitwise operations used in the additions and multiplications by small constants (which contribute significantly as the size of the numbers grows and as $n \to \infty$). Even if often multiplications by small constants are cheap, they grow significantly with the size of the coefficients in the computations of the h_i's. Yet, the algorithm is certainly of theoretical interest and in addition, it can be useful for special hardware where multiplications are particularly expensive compared to additions.

1.3 Fast Fourier Transform (FFT) methods

To simplify the exposition, we will present a very similar algorithm that multiplies quickly two polynomials.

1.3.1 Version for polynomials

Let $p(X), q(X) \in \mathbf{R}[X]$, both of degree $n-1$. Without too much loss of generality, we will assume that $n = 2^k$ is a power of 2. If

$$p(X) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} \qquad \text{and} \qquad q(X) = b_0 + b_1 X + \cdots + b_{n-1} X^{n-1},$$

then the product polynomial $r(X) = p(X)q(X) = c_0 + c_1 X + \cdots + c_{2n-2} X^{2n-2}$ has degree $2n-2$ and the coefficients are given by $c_k = \sum_{\substack{i+j=k \\ 0 \le i,j \le n-1}} a_i b_j$. Computing the above product requires multiplying each a_i with each b_j and hence, n^2 multiplications (or less if some coefficients are 0). For each coefficient c_k, the number of additions is equal to the number of pairs (i,j) satisfying the conditions minus one. This yields a total of $\sum_{k=0}^{n-1} k + \sum_{k=n}^{2n-2} (2n-2-k) = n^2 - 2n + 1$ additions We thus have a run-time of $\mathcal{O}(n^2)$ elementary operations in $\mathbf{R}$ which makes the algorithm rather slow in practice (here, multiplications in $\mathbf{R}$ are considered much more expensive than additions in $\mathbf{R}$).

One can use another approach instead based on Fast Fourier Transform (FFT). The idea is that any polynomial $f(X) = u_0 + u_1 X + \cdots + u_{n-1} X^{n-1}$ of degree exactly $n-1$ is uniquely determined by its values at n distinct points (finding the polynomial from the evaluations is known as interpolation). A good choice of evaluation points are the numbers ω_n^k for $0 \le k \le n-1$, where ω_n is a primitive nth root of unity (for example $\omega_n = e^{2\pi i/n}$). In what follows, we explain why this choice is good.

We now represent any polynomial $f(X)$ of degree $n-1$ by the vector in $\mathbf{R}^n$ of its coefficients. The Discrete Fourier Transform (DFT) of $f(X)$ for ω_n is

$$\mathtt{DFT}_{\omega_n} : \mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \qquad \mapsto \qquad \widehat{\mathbf{u}} = \left(\widehat{u_0}, \widehat{u_1}, \ldots, \widehat{u_{n-1}}\right), \tag{1.1}$$

where $\widehat{u_j} = \sum_{t=0}^{n-1} u_t \omega_n^{jt} = f(\omega_n^j)$. The main point here is that there are algorithms such as Algorithm 4 computing the DFT with a complexity better than the naïve approach requiring $\mathcal{O}(n^2)$ elementary operations in $\mathbf{R}$. If this is the case, we speak of Fast Fourier Transform (FFT).

The DFT-based algorithm to compute the product $r(X)$ of two degree $n-1$ polynomials $p(X)$ and $q(X)$ will essentially evaluate $p(X)$ and $q(X)$ at $X = \omega_{2n}^m$ (here, ω_{2n} is a primitive $2n$-roots of unity) for every $0 \le m < 2n$ using the more efficient algorithm and will then compute the products $p(\omega_{2n}^m) q(\omega_{2n}^m) = r(\omega_{2n}^m)$. This is because of the basic convolution theorem according to which

$$\mathtt{DFT}_{\omega_{2n}}(r(X)) = \mathtt{DFT}_{\omega_{2n}}(p(X) \cdot q(X)) = \mathtt{DFT}_{\omega_{2n}}(p(X)) \cdot \mathtt{DFT}_{\omega_{2n}}(q(X)).$$

We will thus get more efficiently the discrete Fourier transform of the coefficient vector of $r(X)$. To get $r(X)$ itself, we will use an inverse-FFT algorithm to compute the inverse $\mathtt{IDFT}_{\omega_{2n}}$ of the transformation (1.1). But the inverse can be performed essentially via the same algorithm as `RECURSIVE-DFT` with the same complexity. The inversion is based on the relation

$$\frac{1}{2n}\,\mathtt{DFT}_{\omega_{2n}^{-1}} \circ \mathtt{DFT}_{\omega_{2n}} = \mathrm{id}\,.$$

Note that we consider a $2n$th roots instead of a nth root because of r.

Algorithm 1 `DFT-Product`

```
Require: Two degree $n-1$ polynomials $p$ and $q$ seen in $\mathbf{R}^n$, $\omega_{2n}$ a primitive $2n$-root
    of unity
Ensure: The vector of the coefficients of $r = p \cdot q$ in $\mathbf{R}^{2n-1}$
 1: $\widehat{p} := \mathtt{DFT}_{\omega_{2n}}(p)$
 2: $\widehat{q} := \mathtt{DFT}_{\omega_{2n}}(q)$
 3: $\widehat{r} := \widehat{p} \cdot \widehat{q}$
 4: $r := \frac{1}{2n}\mathtt{DFT}_{\omega_{2n}^{-1}}(\hat{r})$
 5: return $r$
```

Example 1. For example, let $p(X) = a_0 + a_1X$ and $q(X) = b_0 + b_1X$, so that $n = 2$. Consider $\omega_{2n} = e^{2\pi i/(2n)} = i$. Then,

$$\mathtt{DFT}_i(a_0, a_1) = [a_0 + a_1, a_0 + a_1 i, a_0 - a_1, a_0 - a_1 i]$$

and similarly for q. The product of these two vectors is

$$\begin{aligned}&[\\ &\quad (a_0 + a_1)(b_0 + b_1),\\ &\quad (a_0 + a_1 i)(b_0 + b_1 i),\\ &\quad (a_0 - a_1)(b_0 - b_1),\\ &\quad (a_0 - a_1 i)(b_0 - b_1 i)\\ &]\end{aligned}$$

and the DFT of this last vector at $\omega_{2n}^{-1} = -i$ is $[4a_0b_0, 4(a_0b_1 + a_1b_0), 4a_1b_1, 0]$.

We now present a fast recursive algorithm (Algorithm 4) to compute the DFT. It is based on the remark that, for $n = 2^k$, $f(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1} = (a_0 + a_2X^2 + \cdots + a_{n-2}X^{n-2}) + X(a_1 + a_3X^2 + \cdots + a_{n-1}X^{n-2})$ so that evaluating $f(X)$ at a primitive nth root of unity can be done by evaluating two degree $(n-2)/2$ polynomials at a primitive $(n/2)$th root of unity.

Let $T(n)$ be the run-time (number of elementary steps, i.e., "addition" and "multiplication" of complex numbers). Steps 7. and 8. take a total of $2T(n/2)$ elementary steps whereas steps $12 - 16$ take a linear (in n) number of steps, i.e., $\Theta(n)$. We thus get a recurrence relation

$$T(n) = 2T(n/2) + \Theta(n),$$

from which we compute $T(n) = \Theta(n \log_2 n)$. This is certainly asymptotically faster than the naïve algorithm described in the beginning.

Algorithm 2 `RECURSIVE-DFT`

```
Require: An integer $n = 2^k$ and a vector $\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$.
Ensure: $\mathtt{DFT}_{\omega_n}(\mathbf{a}) = \widehat{\mathbf{a}} = (\widehat{a_0}, \widehat{a_1}, \dots, \widehat{a_{n-1}})$.
 1: $\mathbf{a}^{\text{even}} := (a_0, a_2, \dots, a_{n-2})$
 2: $\mathbf{a}^{\text{odd}} := (a_1, a_3, \dots, a_{n-1})$
 3: if $n = 2$ then
 4:     $\widehat{\mathbf{a}^{\text{even}}} := \mathbf{a}^{\text{even}}$
 5:     $\widehat{\mathbf{a}^{\text{odd}}} := \mathbf{a}^{\text{odd}}$
 6: else
 7:     $\widehat{\mathbf{a}^{\text{even}}} :=$ RECURSIVE-DFT$(n/2, \mathbf{a}^{\text{even}})$
 8:     $\widehat{\mathbf{a}^{\text{odd}}} :=$ RECURSIVE-DFT$(n/2, \mathbf{a}^{\text{odd}})$
 9: end if
10: $\omega_n = e^{\frac{2\pi i}{n}}$
11: $w = 1$
12: for $i = 0, \dots, 2^{k-1} - 1$ do
13:     $\widehat{a_i} = \widehat{a_i^{\text{even}}} + w\widehat{a_i^{\text{odd}}}$
14:     $\widehat{a_{i+2^{k-1}}} = \widehat{a_i^{\text{even}}} - w\widehat{a_i^{\text{odd}}}$.
15:     $w := w \cdot \omega_n$
16: end for
17: return $\widehat{\mathbf{a}} = (\widehat{a_0}, \dots, \widehat{a_{n-1}})$.
```

1.3.2 Cooley–Tukey algorithm

The above recursive FFT algorithm is a particular case of a more general FFT algorithm due to Cooley and Tukey [CT65], one of the most common and general variants of FFT. Here, we assume that $n = n_1 n_2$ and we arrange the coefficients into a 2D array. One then expresses

$$\begin{aligned}
\widehat{a_{k_1 n_2 + k_2}} &= \sum_{m_1=0}^{n_1-1} \sum_{m_2=0}^{n_2-1} a_{n_1 m_2 + m_1} e^{-\frac{2\pi i}{n_1 n_2}(n_1 m_2 + m_1)(n_2 k_1 + k_2)} = \\
&= \sum_{m_1=0}^{n_1-1} e^{-\frac{2\pi i}{n_1 n_2} m_1 k_2} \left(\sum_{m_2=0}^{n_2-1} a_{n_1 m_2 + m_1} e^{-\frac{2\pi i}{n_2} m_2 k_2} \right) = \\
&= \sum_{m_1=0}^{n_1-1} \left(\sum_{m_2=0}^{n_2-1} a_{n_1 m_2 + m_1} e^{-\frac{2\pi i}{n_2} m_2 k_2} \right) e^{-\frac{2\pi i}{n_1 n_2} m_1 (n_2 k_1 + k_2)}.
\end{aligned}$$

The above formula shows that to perform a DFT of size $n_1 n_2$, it suffices to perform:

1. n_1 DFTs of size n_2,
2. Multiplications by the appropriate roots of unity,
3. n_2 DFTs of size n_1.

1.3.2.1 `RecursiveDFT` as Radix-2 DIT

The algorithm `RECURSIVE-DFT` described above is a 2-radix decimation-in-time (radix-2 DIT) form of the algorithm of Cooley–Tukey. There are other forms of Cooley–Tukey that are useful in practice.

1.3.2.2 Data ordering, data access and variations

There are different variations of the above FFT algorithms according to the application or the architecture. See [JF09] for a good overview of some of these variations of FFT.

One particular aspect is designing an *in-place* algorithm: i.e., an algorithm that overwrites its input with its output data that uses constant auxiliary storage. To get an in-place Radix-2 DIT, one uses a technique known as bit-reversal.

Algorithm 3 `IterativeDFT`

```
Require: An integer $n = 2^k$ and a vector $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$.
Ensure: A vector $r$ containing $\mathtt{DFT}_{\omega_n}(\mathbf{a}) = \widehat{\mathbf{a}} = (\widehat{a_0}, \widehat{a_1}, \ldots, \widehat{a_{n-1}})$.
 1: bit-reverse-copy$(a, r)$
 2: for $s = 1, \ldots, k$ do
 3:   $m = 2^s$ and $\omega_m := \exp(-2\pi i/m)$
 4:   for $\ell = 0, m, 2m, \ldots, 2^k - m$ do
 5:     $\omega := 1$
 6:     for $j = 0, \ldots, m/2 - 1$ do
 7:       $t := \omega r[k + j + m/2]$
 8:       $u := r[k + j]$
 9:       $r[k + j] := u + t$
10:       $r[k + j + m/2] := u - t$
11:       $\omega := \omega\omega_m$
12:     end for
13:   end for
14: end for
15: return $r$.
```

Dimitar : Complete!

1.3.3 Multiplying large integers - the method of Schönhage–Strassen

An efficient algorithm based on FFT that multiplies two n-digit numbers was discovered in 1971 by Schönhage and Strassen [SS71]. The run-time of this algorithm is $\mathcal{O}(n \log n \log \log n)$ which is less than the run-time of Karatsuba multiplication. Note that it outperforms the method of Karatsuba and other older methods for numbers of more than 10,000 decimal digits. It is currently a part of the GNU Multiprecision Library and is used for at least 1728 to 7808 64-bit words (33,000 to 150,000 decimal digits), depending on architecture. In addition, a Java implementation of the method is used for implementing multiplication in Java for big integers of more than 74,000 decimal digits[1].

The basic idea is the following: to multiply two numbers, e.g., 156×723, we

[1] The Java `BigInteger` class (`https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html`) uses the algorithm of Schönhage and Strassen.

consider first the linear convolution sequence:

$$\begin{array}{rrrrr} & & 1 & 5 & 6 \\ & & 7 & 2 & 3 \\ \hline & & 3 & 15 & 18 \\ & 2 & 10 & 12 & \\ 7 & 35 & 42 & & \\ \hline 7 & 37 & 55 & 27 & 18 \end{array}$$

The sequence $(7, 37, 55, 27, 18)$ is known as the *linear (or acyclic) convolution* of the two sequences $(1, 5, 6)$ and $(7, 2, 3)$. In general, for two n-digit numbers, the length of that sequence is always $2n - 1$. There are two useful convolutions computed out of that sequence: the *cyclic convolution* and the *negacyclic convolution.* The cyclic convolution in the example above yields the sequence

$$\begin{array}{crrr} & 55 & 27 & 18 \\ + & & 7 & 37 \\ \hline & 55 & 34 & 55 \end{array}$$

The negacyclic convolution is

$$\begin{array}{crrr} & 55 & 27 & 18 \\ - & & 7 & 37 \\ \hline & 55 & 20 & -19 \end{array}$$

The key observation is that the product of two n-digit base-b numbers is equivalent modulo $b^n - 1$ to the cyclic convolution obtained from the two sequences corresponding to the base-b digits of the numbers. Similarly, the product of two n-digit base-b numbers is equivalent modulo $b^n + 1$ to the negacyclic convolution. The Schönhage–Strassen's algorithm relies on the negacyclic convolution rather than the cyclic one for various efficiency (and other) reasons that we explain below.

The idea uses a weighted version of the DFT algorithm in order to compute what is called the negacyclic convolution. More precisely, letting v_x and v_y be the vectors of base-b digits of the numbers x and y, we have

$$\texttt{CyclicConvolution}(v_x, v_y) = \texttt{IDFT}_\omega(\texttt{DFT}_\omega(v_x) \cdot \texttt{DFT}_\omega(v_y)),$$

and

$$\texttt{NegacyclicConvolution}(v_x, v_y) = A^{-1} \cdot \texttt{IDFT}_\omega(\texttt{DFT}_\omega(A \cdot v_x) \cdot \texttt{DFT}_\omega(A \cdot v_y)),$$

where $A = \text{diag}(\omega^i)_{i=0}^n$ where ω is a primitive $2n$th root of unity.

In order to apply the recursive DFT algorithm in practice to integers in b-base representation, we need to work modulo some number N (what is known as the *Number Theoretic Transform, or NTT*). Over the real numbers $\mathbf{R}$ or the complex numbers $\mathbf{C}$, it was natural to use a primitive $2n$th root of unity. Yet, it is not automatic that such a root of unity would exist modulo N. We thus need to determine a special modulus N for which we have a primitive root of unity. If N is sufficiently large, then computing $xy \mod N$ yields the product xy.

A major part of the algorithm is the careful choice of N to ensure that multiplications the primitive root and reductions modulo N are performed very efficiently (essentially, using only bit shifts and additions).

Algorithm 4 `Schönhage-Strassen's algorithm`

```
Require: Integers $x$ and $y$; an integer $n$
Ensure: Computes $xy \mod 2^n + 1$ using the negacyclic convolution
1: Decompose both $x$ and $y$ into $2^k$ equal parts where $2^k \mid n$ and set $n'$ to be the smallest integer that is at least $2n/2^k + k$ and is divisible by $2^k$ ($n'$ is the recursion length)
2: Compute $A \cdot v_x$ and $A \cdot v_y$ via shifts (the $j$th component shifted by $n'j/2^k$)
3: Compute $\texttt{DFT}_\omega(A \cdot v_x)$ and $\texttt{DFT}_\omega(A \cdot v_y)$ via the NTT variant of RECURSIVE-DFT (using $\omega = 2^{2n'}/2^k$ as a primitive $2^k$th root of unity, one performs multiplications as shifts)
4: Apply recursively the algorithm to compute the element-wise product $\texttt{DFT}_\omega(A \cdot v_x) \cdot \texttt{DFT}_\omega(A \cdot v_y)$
5: Compute $\texttt{IDFT}_\omega(\texttt{DFT}_\omega(A \cdot v_x) \cdot \texttt{DFT}_\omega(A \cdot v_y))$ using the NTT variant of RECURSIVE-DFT (multiplications are again shifts)
6: Multiply the result vector by $A^{-1}$ using shifts only
7: return Result after carrying modulo $2^n + 1$.
```

The complexity is thus expressed in terms of the parameter k in Step 1. The optimal k is when $2^k \sim \sqrt{n}$ and in this case, we obtain a complexity $\mathcal{O}(n \log n \log \log n)$.

1.3.3.1 An algorithm for polynomial multiplication in $\mathbf{Q}[x]$.

Note that the rational numbers $\mathbf{Q}$ do not contain a primitive $2n$th root of unity. Instead, assuming that $n = 2^k$, we consider the polynomials in $\mathbf{Q}[x]$ modulo $x^n + 1$. We have the following congruences of polynomials:

$$x^n \equiv -1 \bmod x^n + 1 \qquad \text{and} \qquad x^{2n} \equiv 1 \bmod x^n + 1.$$

This means that the polynomial $\omega = x \mod x^n + 1$ is a $2n$th root of unity. Since 2 is invertible in $\mathbb{Q}$, ω is a primitive $2n$th root of unity. If we are able to replace arithmetic in $\mathbf{Q}$ with arithmetic with rational polynomials modulo $x^n + 1$, we will then be able to compute the product of two polynomials in $\mathbf{Q}[x]$ of degrees at most n in $\mathcal{O}(n \log n)$ operations with polynomials mod $x^n + 1$.

To make this precise, consider $f, g \in \mathbf{Q}[x]$ with $\deg(fg) \le 2n = 2^k$ and let $m = 2^{\lfloor k/2 \rfloor}$, $t = 2n/m$. Write the polynomials f and g as

$$f(x) = f_0(x) + x^m f_1(x) + \cdots + x^{m(t-1)} f_{t-1}(x)$$

and

$$g(x) = g_0(x) + x^m g_1(x) + \cdots + x^{m(t-1)} g_{t-1}(x),$$

where the degrees of the f_i's and g_i's are less than m. We then see that

$$f(x) = F(x, x^m) \text{ for } F(x, y) = f_0(x) + yf_1(x) + \cdots + y^{t-1}f_{t-1}(x),$$

and

$$g(x) = G(x, x^m) \text{ for } G(x, y) = g_0(x) + yg_1(x) + \cdots + y^{t-1}g_{t-1}(x).$$

The key observation is that in order to compute fg, it suffices to compute FG modulo $y^t + 1$. Indeed, if

$$F(x, y)G(x, y) = H(x, y) + q(x, y)(y^t + 1).$$

then

$$f(x)g(x) \equiv H(x, x^m) \bmod x^{mt} + 1 = H(x, x^m) \bmod x^n + 1.$$

1.3.3.2 An algorithm for large integer multiplication.

We are not presenting the version of the algorithm of Schönhage and Strassen, but rather, a version based on the Chinese remainder theorem. Suppose that one has two large integers $a = \sum_{0 \le i < \ell} a_i 2^{Bi}$ and $b = \sum_{0 \le i < \ell} b_i 2^{Bi}$. We will also assume that ℓ does not exceed 2^B (typical examples that occurs in practice is $B = 64$). In this case, we can choose three auxiliary primes p_1, p_2 and p_3 between 2^{B-1} and 2^B. In order to obtain ab, it suffices to compute $ab \bmod p_i$ for each $i = 1, 2, 3$ and then use the Chinese remainder theorem. If the p_i's are chosen appropriately (i.e., such that $p_i - 1$ is divisible by a large power of 2) then $a \bmod p_1$ can be computed using FFT modulo p_i (by the choice of p_i, there will be a primitive $2n$th root of unity modulo p_i).

1.4 Recent developments and summary

In 2007, Fürer [Für07] presented an algorithm for asymptotically faster multiplication of very large integers compared to the algorithm of Schönhage and Strassen.

Surprisingly, a recent algorithm of Harvey and van der Hoeven [HvdH21] multiplies two n-bit integers in $\mathcal{O}(n \log n)$ operations, thus, proving a long-standing conjecture of Schönhage–Strassen.

In summary, we have seen the following multiplication algorithms with indicated complexities:

Algorithm	**Complexity**
Schoolbook Multiplication	$\mathcal{O}(n^2)$
Karatsuba	$\mathcal{O}(n^{\log_2 3})$
Toom–Cook	$\mathcal{O}(n^{1+\varepsilon})$
Fast Fourier Transform	$\mathcal{O}(n \log n)$
Schönhage–Strassen	$\mathcal{O}(n \log n \log \log n)$
Harvey–vdHoeven	$\mathcal{O}(n \log n)$

1.5 Exercises

Exercise 1.1.

1. Prove in detail that if two maps $T, g : \mathbf{R}_{>0} \to \mathbf{R}_{>0}$ are bounded on any bounded interval, satisfy $T(x) = 2T(x/2) + g(x)$ for every real number $x \geq 1$ and $g(x) = \mathcal{O}(x)$, then $T(x) = \mathcal{O}(x \log(x))$.
2. Let $a, b \geq 2$ be integers. Prove that if a map $T : \mathbf{Z}_{>0} \to \mathbf{R}_{>0}$ satisfies $T(n) \leq aT(\lceil n/b \rceil)$ for all $n \geq 1$, then $T(n) = \mathcal{O}(n^{\log_b(a)})$.

Exercise 1.2 (SAGE)**.** Write a SAGE function `Karatsuba(A, B)` that takes two integers A and B and returns their product using the algorithm from section 1.2.2. Run and time the above algorithm on 100 pairs of uniformly random positive integers less than 2^{512}.

Exercise 1.3. Prove the correctness of the `RECURSIVE-DFT` algorithm supposing $n = 2^k$.

Exercise 1.4 (SAGE)**.**

1. Write the function `RecursiveDFT` taking as input a vector $a \in \mathbf{Z}^n$ (e.g., the coefficients of a polynomial $p(x)$ of degree n) as well as a parameter ω (e.g., a complex root of unity) and that outputs the vector $\widehat{a} = \mathrm{DFT}_\omega(a)$. Here we can assume that $n = 2^k$ is a power of 2.
2. Write the function `InverseRecursiveDFT`.
3. Write a function `DFT_product` that computes the product of two integers, using fast Fourier transform. It is not needed to implement Schönhage-Strassen's algorithm, but you can use the idea that 2 is a primitive $2n$-th root of unity in $\mathbf{Z}/(2^n + 1)\mathbf{Z}$. Otherwise working with complex roots of unity is fine.

CHAPTER 2 Exponentiation Algorithms

As hinted earlier in the course, one of the arithmetic operations that needs to be optimized for both RSA encryption, RSA signatures as well as discrete-log based systems is exponentiation. We now review various methods for fast exponentiation.

2.1 Basic Binary Ladders

The basic multiplication algorithm is based on binary ladders (we also commonly call these square-and-multiply algorithms). The point is that they work as if we are doing a recursion, except that they are still iterative.

Algorithm 5 BINEXP_LR

```
Require: A base $x$ and an exponent $y = (y_{d-1} \dots y_0)_2$ in binary.
Ensure: $x^y$.
1: $z = x$
2: for $i = d-2, \dots, 0$ do
3:     $z := z^2$
4:     if $y_i = 1$ then
5:         $z := z \cdot x$
6:     end if
7: end for
8: return $z$
```

For instance, the above algorithm will compute x^{23} as follows:

$$x^{23} = x^{2^4+2^2+2+1} = \left(((x^2)^2 \cdot x)^2 \cdot x\right)^2 \cdot x.$$

Note that if you want to do modular exponentiation (i.e., compute $x^y \bmod N$), you want to do modular multiplications in Steps 3. and 5.

This algorithm is essentially an “unrolled" version of the following simple recursive algorithm POW below (which you will never use in practice for large exponents - why?).

An alternative way to compute x^{23} is as follows:

$$x^{23} = x \cdot x^2 \cdot (x^2)^2 \cdot (x^8)^2$$

Algorithm 6 POW

```
Require: A base $x$ and an exponent $n$.
Ensure: $x^n$.
1: if $n = 1$ return $x$
2: if $n$ is even return POW$(x^2, n/2)$
3: if $n$ is odd return POW$(x^2, (n-1)/2) \cdot x$
```

This displays slightly better the binary expansion of the number 23. The right-to-left binary ladder is then the following algorithm:

Algorithm 7 BINEXP_RL

```
Require: A base $x$ and an exponent $y = (y_{d-1} \dots y_0)_2$ in binary.
Ensure: $x^y$.
1: $z := x$
2: $a := 1$
3: for $i = 0, \dots, d-1$ do
4:     if $y_i = 1$ then
5:         $a := z \cdot a$
6:     end if
7:     $z := z^2$
8: end for
9: return $a$
```

A priori, the above algorithm seems to require the same number of multiplications and squarings as Algorithm 7. There is one practical advantage of Algorithm 5 to Algorithm 7, namely, that the multiplicand x in the former is fixed and in many cases (e.g., primality testing, where x is small and one computes x^{n-1}) the multiplication $z \cdot x$ can be made quite fast.

To write the cost of the above algorithm, we assume that S is the cost of a squaring and M is the cost of multiplication. The total cost is then $(\log y)S + HM$ where H is the Hamming weight of the exponent y (i.e., the number of 1s in the binary expansion of y).

2.2 Window exponentiation

There is an exponential algorithm that is slightly more general than the binary ladder algorithms in the sense that it exploits the base-B representation of the exponent for some $B = 2^b$. It assumes that the exponents $\{x, x^2, \dots, x^{B-1}\}$ have been precomputed. The algorithm is the following:

To illustrate the advantage of the windowing ladder, consider, e.g., computing x^{79}. First, $79 = (1001111)_2 = (1033)_4$. For $B = 2^2$, the windowing ladder computes x^{79} as follows:

$$x^{79} = \left(\left(x^4\right)^4 \cdot x^3\right)^4 \cdot x^3$$

The cost of this algorithm is $6S+2M$. On the other hand, the binary ladder algorithm

Algorithm 8 WINDOWING LADDER

```
Require: A base $x$ and an exponent $y = (y_{d-1} \dots y_0)_B$ in base $B = 2^b$. We assume
    precomputed values of $\{x^m : 0 < m < B,\ m \text{ is odd}\}$.
Ensure: $x^y$.
1: $z := 1$
2: for $i = d-1, \dots 0$ do
3:     Write $y_i =: 2^c \cdot m$ where $m$ is either 0 or odd.
4:     $z := (x^m)^{2^c} \cdot z$
5:     if $i > 0$ then
6:         $z := z^{2^b}$
7:     end if
8: end for
9: return $z$
```

computes it as follows:

$$x^{79} = \left(\left(\left(x^{2^3} \cdot x\right)^2 \cdot x\right)^2 \cdot x\right)^2 \cdot x,$$

and the cost is obviously $6S + 4M$. Of course, one should not forget the cost of the precomputation of $\{x, x^2, x^3\}$. Yet, if the exponent y is large, the windowing ladder saves multiplications.

2.3 Fixed-base ladders

The previous algorithm windowing ladder algorithm provides a hint on how to get a very efficient exponentiation algorithms when the base x is fixed (i.e., we reuse it for performing multiple exponentiations with different y's).

In this case, we can simply do a large precomputation to store all the values

$$\{x^{\ell B^j} : 1 \le \ell \le B-1,\ 1 \le j \le d-1\}.$$

Once this computation has been done, the algorithm then becomes very simple as all we need is a multiplication for each B-ary digit, making a total of $\dfrac{\log y}{\log B}$ multiplications.

Algorithm 9 FIXED-BASE LADDER

```
Require: A base $x$ and an exponent $y = (y_{d-1} \dots y_0)_B$ in base $B = 2^b$. We assume
    precomputed values of $\{x^m : 0 < m < B\}$.
Ensure: $x^y$.
1: $z := 1$
2: for $j = d-1, \dots 0$ do
3:     $z := (y_j \cdot x^{B^j}) \cdot z$
4: end for
5: return $z$
```

2.4 Addition/Lucas chains/Montgomery ladder

We will see how this method works in the context of primality testing a little later in the course. For the moment, all of the previous methods motivate the following basic question:

Question 2.4.1. *What is the length of the shortest addition chain sequence $a_0, a_1, \dots, a_r$ such that $a_0 = 1$ and $a_r = y$? Here, addition chain sequence is defined as a sequence of positive integers with the property that for every index i, there exist indices $0 \leq j, k < i$ such that $a_i = a_j + a_k$.*

The question of the length $\ell(y)$ of the shortest addition chain for the number y is an interesting one. It is difficult to compute the exact value of $\ell(y)$. Yet, Erdös showed that (see [Erd60])

$$\ell(y) = \log y + (1 + o(1))\frac{\log y}{\log\log y}.$$

In practice, it is hard to compute the optimal addition chain, so one typically uses an almost optimal chain (as we see in the examples above).

Lucas chains are special types of addition chains introduced by Peter Montgomery [Mon83], originally as a method of speeding up scalar multiplications on elliptic curves and subsequently, as performing exponentiations. It is an addition chain $a_0, a_1, \dots, a_r$ such that $a_0 = 1$, for each index i, there exist indices $1 \leq j, k < i$ such that $a_i = a_j + a_k$ with either $a_j = a_k$ or $|a_j - a_k| = a_m$ for some m. These are helpful to compute recurrences of the form $X_{m+n} = f(X_m, X_n, X_{m-n})$. Montgomery also computes lower bounds on the lengths of Lucas chains in [Mon83].

The concept of Lucas chains led Montgomery to propose a ladder that ended up being helpful not only for elliptic curve scalar multiplications, but also for general exponentiations. To explain the Montgomery ladder, let $y = (y_{d-1} \dots y_0)_2$ be the binary expansion of the exponent and for each j, let $L_j = (y_{d-1} \dots y_j)_2$. Let $H_j = L_j + 1$. Then

$$L_j = 2L_{j+1} + y_j = L_{j+1} + H_{j+1} + y_j - 1 = 2H_{j+1} + y_j - 2.$$

The idea of the Montgomery ladder is to use to registers z_0 and z_1 in which one would store x^{L_j} and x^{H_j} at any given step. Since

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) & \text{if } y_j = 0 \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) & \text{if } y_j = 1, \end{cases}$$

this leads to the following algorithm:

Even if it might seem *a priori* that the Montgomery ladder has more multiplications than the LR or RL binary ladders, it should be noted that there are three main reasons why the Montgomery ladder is of interest:

1. (Fixed R_1/R_0) - this properly was crucially used by Montgomery for speeding up elliptic curve scalar multiplication (we will see how this work later in the course, so we will not discuss it now).

Algorithm 10 MONTGOMERY LADDER

```
Require: A base $x$ and an exponent $y = (y_{d-1} \dots y_0)_2$.
Ensure: $x^y$.
1: $z_0 := 1$, $z_1 = x$
2: for $j = d-1, \dots 0$ do
3:   if $y_j = 0$ then
4:     $z_1 := z_0 \cdot z_1$, $z_0 := (z_0)^2$
5:   else
6:     $z_0 = z_0 \cdot z_1$, $z_1 := (z_1)^2$
7:   end if
8: end for
9: return $z_0$
```

2. (Parallelization) - note that each pair of multiplications (in Line 4 and Line 6) can be parallelized as the multiplications are independent. For modular exponentiation, this can be an advantage - if the cost of modular multiplication is M, modular squaring is slightly faster, so the cost of each of these lines on a machine with two processors is M.

3. (Common multiplicand) - it should be noted that the two pairs of multiplications (both Lines 4 and 6) have a common multiplicand - if $y_j = b$ then the two multiplications are $z_b \cdot z_{\neg b}$ and $(z_b)^2$. This allows for some practical speedups, originally used for speeding up the above binary ladders [YL93]. The basic idea is to express the two multiplications in terms of logical operations. Since the common multiplicand in both cases is z_b, we write

$$z_{\texttt{com}} := z_0 \wedge z_1, \qquad z_{b,c} := z_{\texttt{com}} \oplus z_b, \qquad z_{\neg b,c} := z_{\texttt{com}} \oplus z_{\neg b},$$

and observe that $z_{\neg b} = z_{\texttt{com}} + z_{\neg b,c}$. We then have

$$z_b \cdot z_{\neg b} = z_b \cdot z_{\texttt{com}} + z_b \cdot z_{\neg b,c},$$

and

$$(z_b)^2 = z_b \cdot z_{\texttt{com}} + z_b \cdot z_{b,c}.$$

Since $z_b \cdot z_{\texttt{com}}$ is common for the two products, it can be computed only once. The gain from the above computation comes from the fact that, on average, the Hamming weights of $z_{\texttt{com}}$, $z_{b,c}$ and $z_{\neg b,c}$ are half the Hamming weights of the inputs, thus, the multiplications $z_b \cdot z_{b,c}$, $z_b \cdot z_{\texttt{com}}$ and $z_b \cdot z_{\neg b,c}$ requiring half less binary additions.

2.5 Exercises

Exercise 2.1 (SAGE)**.** Here, you will do a few basic SAGE exercises related to some bit operations.

1. Write a simple one-line command (in SAGE) that calculates $3^{4324324}$ modulo

$2^{1000000}$ using the Python/SAGE generic `%` (modulo) operator. Time the calculation. You probably notice that it is quite slow. Using the Python "AND" operator `&`, show how to write another one-line command that speeds this up. Give a short justification of why you are getting the same answer. Now, implement a function `myLSB(N, k)` that takes as input integers N and k and outputs the k least significant bits in the binary representation of N.

2. Recall Python's right-shift (`>> k`) and left-shift (`<< k`) operators. Implement a function `myMSB(N, k)` that takes an integer N and an integer k and returns the k most significant bits of the binary representation of N.

Exercise 2.2 (SAGE)**.** Implement `Montgomery_mult` and `Montgomery_exp` corresponding to the Montgomery multiplication and exponentiation respectively. Test the correctness of your functions using the `%` operator and compare the timing.

Exercise 2.3. 1. Write a *recursive* function `Fibo_recursive(n)` (i.e., your function calls itself) that outputs the n-th Fibonacci number (1, 1, 2, 3, 5, 8, 13, ...).

2. Write an *iterative* function `Fibo_iterative(n)` (i.e., using a `for` loop) that outputs the n-th Fibonacci number.

3. Run your two functions on $n = 32$, time and compare the results.

Exercise 2.4. For an integer $n \geq 1$, let $\ell(n)$ be the shortest length of an addition chain $a_0 = 1 < a_1 < ... < a_{\ell(n)} = n$ (i.e., for every integer k such that $1 \leq k \leq \ell(n)$ there are indices $0 \leq i, j < k$ such that $a_i + a_j = a_k$). In this exercise, we are going to show that $\ell(n) \sim \log_2(n)$.

1. Show that if $2^s \leq n < 2^{s+1}$ and $s \geq 1$ then $s \leq \ell(n) \leq 2s$.

2. Prove that if $r \geq 1, s \geq 0$ are integers such that $2^{rs} \leq n < 2^{r(s+1)}$ then there is an addition chain[1] for n of length at most $(r+1)s + 2^r - 2$ and which starts with $a_i = i$ for all $i \in \{0, ..., 2^r - 2\}$, that is:

$$a_0 = 1, \;\; a_1 = 2, \;\; a_2 = 3, \;\; ..., \;\; a_{2^r-2} = 2^r - 1.$$

Hints: proceed by induction on s. In the induction step, you can work with the euclidean division of n by 2^r (and use the induction hypothesis on the quotient).

3. By choosing $r = \lceil \log(\log(n)) \rceil$ for $n \geq 3$ in b), where $\log = \ln$ is the logarithm in base e, deduce that $\ell(n) \leq \log_2(n)(1 + o(1))$ as $n \to +\infty$.

Exercise 2.5. Find an addition chain $a_0 = 1 < a_1 < ... < a_{15} = 2047$ of length 15 for $n = 2047$.

[1] To be very precise for the case $s = 0$, we should say "there is an addition chain for $\max\{n, 2^r - 1\}$".

CHAPTER 3 Modular and Finite Field Arithmetic

3.1 Congruences and Modular Arithmetic

Assuming that most of you have followed a course on abstract algebra, I will spend less time recalling basic facts about congruences, but rather give you some exercises on these. Recall that for integers $a, b, N \neq 0$, we say that *a is congruent to b modulo N* and write $a \equiv b \bmod N$ if $N \mid a - b$. One of the most basic tools from public key cryptography, the RSA algorithm, relies on congruences. We thus need a way of efficiently performing modular arithmetic (modular addition, multiplication and exponentiation). At the same time, cryptographic schemes based on the discrete logarithm problem (either for finite fields or for elliptic curves) heavily rely on finite field arithmetic. In this lecture, we will learn some basics on how one performs efficient modular and finite field arithmetic on modern computer architectures.

3.1.1 Theorems of Fermat and Euler

We recall two basic theorems that will be needed for, e.g., the RSA protocol. You will prove these in one of the exercises:

Theorem 3.1.1 (Fermat's little theorem)**.** *If p is a prime and $a \in \mathbf{Z}$ is coprime to p then*

$$a^{p-1} \equiv 1 \bmod p.$$

Now, if N is an integer, $\varphi(N)$ will denote the number of integers in $\{1, \ldots, N\}$ that are coprime to N. For instance, $\varphi(5) = 4$ and $\varphi(6) = 2$.

Theorem 3.1.2 (Euler's theorem)**.** *If $a \in \mathbf{Z}$ is such that $(a, N) = 1$ then $a^{\varphi(N)} \equiv 1 \bmod N$.*

Remark 1. The main idea behind the RSA protocol comes from the observation that if $N = pq$ for large primes p and q then if one knows only N (without p and q), computing $\varphi(N) = (p - 1)(q - 1)$ is hard (equivalent to factoring N).

3.2 Newton Methods - Barrett's Algorithm

In order to get deeper into how the RSA algorithm is implemented in modern architectures, one needs to understand better modular arithmetic.

The first step for this is to discuss the computations of the classical functions `div` and `mod`. These will be useful in almost any classical public-key cryptographic operation (e.g., RSA encryption/decryption).

The naïve idea for computing these functions resembles schoolbook multiplication. Knuth studies in detail the complexity of division with remainder in multiprecision arithmetic [Knu, Alg.4.3.1.D]. In particular, to compute $ab \mod N$ where $a, b < N$ are two integers, it takes $\mathcal{O}((\log_2 N)^2)$ multiplications and $\mathcal{O}(\log_2 N)$ divisions. But divisions are much more expensive than multiplications. Hence, one may want to look for an algorithm that uses multiplications only.

A method for reducing an integer $a \bmod N$ using only multiplications was proposed by Barrett [Bar86]. To compute $a \bmod N$, the basic idea is to invert N as a floating point number, i.e., $s = 1/N$ and then use that

$$a \bmod N = a - \lfloor as \rfloor N.$$

To better understand the method, let us restrict to single-word arithmetic i.e., we assume that the values fit into a single machine word. In this case, Barrett reduction tries to approximate $1/N$ with a number of the form $m/2^k$ for a given k (here, m corresponds to the mantissa and $-k$ corresponds to the exponent; notice the minus sign corresponding to the normalization in the definition of floating point representation). In order to compute m, we use $m \sim 2^k/N$ and hence, we wish to find the nearest integer to $2^k/N$. Yet, if $m > 2^k/N$, this will create an overflow, hence, we wish to take $m = \lfloor 2^k/N \rfloor$.

With this choice (assuming sufficient precision in the floating-point representation - to be carefully analyzed later), the division is replaced by the simpler operation that first computes (what we would like to call the *quotient*)

$$q := (a \cdot m) \gg k,$$

where $\gg$ represents the right-shift operator. There is a minor technical detail: because of the choice of m above, it could be that q is one less than the value of the actual quotient, i.e., $a - qN$ can be an integer in the interval $[0, 2N)$ instead of $[0, N)$ which means that an extra subtraction might be needed. We formally prove this in the following lemma:

Lemma 3.2.1. *Let N be a modulus and let a be an integer such that $a \leq N^2$. Let k be any integer such that $2^{k-1} > N$, let $m = \left\lfloor \frac{2^k}{N} \right\rfloor$ and let $q = \left\lfloor \frac{am}{2^k} \right\rfloor$. Then*

$$0 \leq a - qN < 2N.$$

Proof. First, note that by the choice of k, $a < 2^k$. We then write

$$\begin{aligned}
a - qN &= a - \left\lfloor \frac{am}{2^k} \right\rfloor N = a - \left(\frac{am - (am \bmod 2^k)}{2^k} \right) N = \\
&= a - \frac{aN}{2^k} m + \frac{(am \bmod 2^k)N}{2^k} = \\
&= a - \frac{aN}{2^k} \left(\frac{2^k - (2^k \bmod N)}{N} \right) + \frac{(am \bmod 2^k)N}{2^k} = \\
&= \frac{a(2^k \bmod N)}{2^k} + \frac{(am \bmod 2^k)N}{2^k} < 2N,
\end{aligned}$$

which proves the lemma. □

The method can be generalized to multi-precision arithmetic as well. It can also be viewed as a generalization of Newton's method. Once the precomputation step has been done, it requires one multiplication of two $\log_2 N$-bit numbers and hence, a complexity of $\mathcal{O}((\log_2 N)^2)$ and we have avoided the expensive divisions.

3.3 Montgomery Arithmetic

Suppose that you want to compute $ab \bmod N$ where N is a large modulus and a and b are large integers. Montgomery arithmetic, introduced by Peter Montgomery in 1985 [Mon85], is another method of performing fast modular multiplication and exponentiation.

The method requires representing integers modulo N in a special form known as *Montgomery form.* For a single multiplication, it is slower than Barrett reduction, yet, if one is performing many subsequent multiplication (e.g., in an exponentiation algorithm) where one can keep the intermediate results in Montgomery form, it outperforms Barrett reduction.

More specifically, let B be the radix (depending on the computer architecture) and choose s such that $R = B^s > N$. We refer to R as the *Montgomery radix.* The Montgomery form of the residue $a \bmod N$ is defined by as $\overline{a} = aR \bmod N$.

Given two integers u and v, define their Montgomery product to be $M(u, v) = uvR^{-1} \bmod N$. Observe that one has $u \bmod N = M(\overline{u}, 1)$ and also $M(\overline{u}, \overline{v}) = \overline{uv}$. The main idea is that, given two integers u and $v \bmod N$, it is very efficient to compute the Montgomery product $M(u, v)$ if the Montgomery radix R is a power of 2. This means that we can efficiently perform Montgomery modular multiplication and exponentiation given the Montgomery forms, assuming that we have performed the conversion of the original inputs into Montgomery form.

As the conversion is not cheap, computing the product of a single pair of integers is not justified using this method and is often slower than more classical modular arithmetic algorithms such as reduction by division or Barrett's method [Bar86]. Yet, if one performs many modular multiplications or a modular exponentiation (the latter requires many many multiplications), it is better to use Montgomery product as this will lead to speedups.

The main observation for computing the Montgomery product fast is summarized in the following lemma (a very simple exercise on congruences):

Lemma 3.3.1. *Suppose that* $(N, R) = 1$ *and let* $N' = N^{-1} \bmod R$. *Then* $y = x - N(xN' \bmod R)$ *is divisible by* R *and*

$$y/R \equiv xR^{-1} \bmod N. \tag{3.1}$$

If $0 \leq x < NR$ *then* $y/R - (xR^{-1} \bmod N)$ *is either* 0 *or* $-N$.

Proof. $y \equiv 0 \bmod R$ is a direct consequence of $NN' \equiv 1 \bmod R$. The second statement is then a consequence of $y \equiv x \bmod N$ and $(R, N) = 1$. □

The lemma indeed shows that computing the Montgomery product $M(u, v)$ is fast if the Montgomery radix is $R = 2^s$ - if $x < NR$ then computing $xN' \bmod R$ is simply $xN' \& (R - 1)$. If we have precomputed N', it reduces to a single multiplication and an & (AND) operation.

3.3.1 Montgomery multiplication

The algorithm that computes efficiently $M(a, b)$ provided $R = 2^s$ and $R > N$ is the following:

Algorithm 11 MONTGOMERY-MULT

```
Require: A modulus $N$, two integers $0 \leq a, b < N$ and a Montgomery radix $R = 2^s > N$.
Ensure: The Montgomery product $M(a, b)$.
1: $x = a \cdot b$
2: $z = x - N \cdot ((x \cdot N')\&(R - 1)) \gg s$  (bit-shift $s$ positions to the right)
3: if $z < 0$ then
4:     $z := z + N$
5: end if
6: return $z$.
```

3.3.2 Montgomery exponentiation

Using Algorithm 11, we can design an efficient modular exponentiation algorithm that computes $x^y \bmod N$ fast.

Note that for Steps 1. and 2. one can use any divide / mod method (these steps are implemented only in the initialization stage). After Step 8., the value of $\overline{r}$ is indeed $\overline{x}^y$, so the algorithm returns the correct result. The upshot is that any of the modular multiplications inside the **for** loop are Montgomery multiplications and hence, efficient (given by Algorithm 11).

3.3.3 Montgomery reduction and Montgomery arithmetic

As we saw, the identity $M(\overline{u}, \overline{v}) = \overline{uv}$ implies that one can compute the Montgomery form of the product of two integers mod$N$ assuming one has the Montgomery form

Algorithm 12 MONTGOMERY-EXP

```
Require: $(N, R)$ as above, a residue $0 < x < N$ and an exponent $y = (y_{t-1}y_{t-2}\dots y_0)_2$ in binary.
Ensure: $x^y \bmod N$.
1: $\overline{x} = xR \bmod N$
2: $\overline{r} = R \bmod N$
3: for $i = t-1, \dots, 0$ do
4:     $\overline{r} = M(\overline{r}, \overline{r})$
5:     if $y_i = 1$ then
6:         $\overline{r} = M(\overline{r}, \overline{x})$
7:     end if
8: end for
9: return $M(\overline{r}, 1)$
```

of the two integers. In order to apply such an algorithm, one clearly needs to compute efficiently the transformation of an integer mod N into its Montgomery form.

The naïve approach for that is to simply compute $xR \bmod N$ (or $xR^{-1} \bmod N$ if we want to go from Montgomery form of the result to the result itself). To compute $xR^{-1} \bmod N$ faster than this naïve method, one uses what is called **Montgomery reduction**. You will learn more about this algorithm in the homeworks/programming assignments, but we will describe the idea here.

To convert an integer $x \bmod N$ to Montgomery form, instead of the naïve idea, we can simply compute (assuming we have precomputed the Montgomery form $\overline{R}$ of R) the Montgomery product $M(x, \overline{R}) = x \cdot R^2 \cdot R^{-1} = xR \bmod N$. To convert from Montgomery form $\overline{x}$ to the integer $x \bmod N$, we simply compute $M(\overline{x}, 1) = xR \cdot R^{-1} = x \bmod N$.

3.3.4 Concurrent Montgomery multiplication

It is interesting to consider how one can speedup Montgomery multiplication using vectorized instructions (SIMD) or even graphic processing units (GPUs). Extensive research has been done to address these two questions which we describe below. The GPU acceleration is based on the Chinese remainder theorem or residue number system (RNS) representation of integers. It enables to replace the expensive operations modulo the large RSA moduli with concurrent modular operations modulo much smaller moduli.

3.3.4.1 Working with multiprecision integers

In the multiprecision case (i.e., the case $s > 1$), Montgomery multiplication interleaves the multiplication and the reduction step, the advantage being that the intermediate results never exceed $s + 1$ machine words.

More precisely, if $x = x_0 + Bx_1 + \cdots + B^{s-1}x_{s-1}$ then computing $xyR^{-1} \bmod N$ amounts to iterating over $i = 0, \dots, s-1$ and performing the following steps at each iteration:

1. $z := z + x_i y$

2. $q := (N^{-1} \bmod B)z \bmod B$

3. $z := (z + Nq)/B$

Here, z is initialized to 0 and at the end of the loop, if necessary, we correct in case $N \leq z < 2N$ by updating $z := z - N$.

3.3.4.2 Concurrent Montgomery multiplication with SIMD instructions

The first work to show that Montgomery multiplication on the 256-bit vector instruction set `AVX2` outperforms the same computation on the ALU of `x86_64` platform was [GK15].

The following algorithm uses two threads running in parallel to compute Montgomery multiplication (a 2-way SIMD vector instructions found on modern processors). References are [BMSZ13] and [SLG+14].

One can try to compute the two multiplications from Section 3.3.4.1 in parallel. The problem is that, *a priori*, these are dependent on each other. The idea to remove the dependency is to compute q first. This changes the order:

1. $q := (N^{-1} \bmod B)(z_0 + a_i b_0) \bmod B$

2. $z := z + x_i y$

3. $z := (z + Nq)/B$

There is a small penalty that one pays for that, namely, computing the first word z_0 of z twice. Indeed, one needs it in order to compute q and then one recomputes it a second time in the second step. While that might seem like an overhead, it is a relatively small penalty that one pays to remove the dependency of the multiplications and thus, compute the two multiplications in parallel (i.e., 2-way SIMD vector instructions). This is essentially the idea of [BMSZ13]. An even more efficient approach is presented in [SLG+14] where, instead of scheduling the two multiplications in parallel, a 2-way SIMD approach is used for each of this multiplications to remove the *read-after-write* dependencies and reduce the number of bubbles (execution delays on the instruction pipeline).

3.3.4.3 Montgomery multiplications via the residue number system representation

A residue number system (RNS) is an integer representation based on the Chinese remainder theorem (CRT) [Gar59]. A large integer x is represented as a vector of small integer values $(x_1, \dots, x_n)$ where x_i is the residue of x modulo r_i for independent and pairwise coprime moduli $r_1, \dots, r_n$. The RNS modulus $R = r_1 \dots r_n$ and, given an integer x modulo R, the representation of x is simply

$$(x_1, \dots, x_n) = (x \bmod r_1, \dots, x \bmod r_n).$$

Converting back to x is done as follows: let $R_i \equiv (R/r_i)^{-1} \bmod r_i$ (i.e., R_i is an integer in $[0, r_i)$). Recovering x is then

$$x = \sum_{i=1}^{n} x_i R_i \cdot \frac{R}{r_i} \bmod R.$$

The advantage is that one can do many small computations concurrently modulo the small integers r_i. We call the system of moduli $\{r_1, \ldots, r_n\}$ an RNS basis.

A classical reference for modular reduction in residue number systems (RNS) is, e.g., [PP95] or [BDK98]. Significant effort has been made to use graphic cards (graphic processing units, or GPUs) in accelerating public-key cryptography [BI04], [MPS07], [SG08].

Here, we present one possible version of a GPU-friendly algorithm.

- Take an RNS basis $\mathcal{B}_n = \{r_1, \ldots, r_n\}$ and define an RNS modulus $R = r_1 \cdot \cdots \cdot r_n$.
- Idea: use the RNS modulus R as the Montgomery radix.
- As N is not an RNS modulus (either a prime or a product of two primes), need an auxiliary RNS basis $\mathcal{B}'_n = \{r'_1, \ldots, r'_n\}$ with RNS modulus $R' = r'_1 \cdot \cdots \cdot r'_n$.

Suppose that we want to compute $xyR^{-1} \bmod N$ on highly parallel architectures like GPUs. Letting $\mathbf{X}$ (resp., $\mathbf{X}'$) and $\mathbf{Y}$ (resp., $\mathbf{Y}'$) be the representations of x and y with respect to $\mathcal{B}_n$ (resp., $\mathcal{B}'_n$), the Montgomery product is computed via the following steps:

1. Compute $\mathbf{Z} := \mathbf{X} \cdot \mathbf{Y}$ and $\mathbf{Z}' := \mathbf{X}' \cdot \mathbf{Y}'$. Here, the product means that at the ith component we compute $x_i y_i \bmod r_i$.
2. Compute $xy(N^{-1} \bmod R) \bmod R$ by computing $\mathbf{Q} = \mathbf{N}^{-1} \cdot \mathbf{Z}$ (with respect to the RNS basis $\mathcal{B}_n$).
3. Convert $\mathbf{Q} := -\mathbf{N}^{-1} \cdot \mathbf{Z}$ to $\mathbf{Q}'$ in basis $\mathcal{B}'_n$.
4. Compute $\mathbf{C}' := (\mathbf{Z}' + \mathbf{Q}' \cdot \mathbf{N}') \cdot \mathbf{R}^{-1}$ in $\mathcal{B}'_n$,
5. Convert $\mathbf{C}'$ to $\mathbf{C}$ in basis $\mathcal{B}_n$ to recover the result.

3.4 Finite Field Arithmetic

Finite fields play an important role in public key cryptographic protocols based on discrete logarithms. If $\mathbf{F} = \mathbf{F}_p$ is a prime field, arithmetic in $\mathbf{F}$ reduces to arithmetic in $\mathbf{Z}/p\mathbf{Z}$. Here, we review basic techniques for finite field arithmetic on prime fields $\mathbf{F}_p$ finite field extensions of $\mathbf{F}_p$.

3.4.1 Basics on finite fields and finite field extensions

We know from abstract algebra that $F = \mathbf{Z}/N\mathbf{Z}$ is a field if and only if N is a prime. A *finite field* is a field F with finitely many elements. For any prime p, we denote by $\mathbf{F}_p$ the finite field with p elements.

Recall that a finite field F has a characteristic $\mathrm{char}(F)$ that is the smallest integer n such that $\underbrace{1+\cdots+1}_{n \text{ times}} = 0$. The characteristic is always a prime number p. In addition, if $\mathrm{char}(F) = p$ then this means that $\mathbf{F}_p \subset F$, i.e., F is a $\mathbf{F}_p$-vector space. This means that $|F| = p^n$ for some $n > 0$. Next, we recall that the multiplicative group of a prime field is cyclic.

Proposition 3.4.1. *The multiplicative group of a finite field $\mathbf{F}_{p^n}^{\times}$ is cyclic.*

Proof. We vaguely recall the idea of the proof. If $a \in \mathbf{F}_{p^n}^*$, one can talk about the order m of a, i.e., the smallest integer m such that $a^m = 1$ in $\mathbf{F}_{p^n}^*$. Note that $m \mid p^n - 1$.

Next, if a has order s and b has order t, and $\gcd(s,t) = 1$ then one can show that ab has order st. To prove then that $\mathbf{F}_{p^n}^{\times}$ is cyclic, we factor $p^n - 1 = q_1^{\alpha_1} \dots q_r^{\alpha_r}$ (where all q_i are primes) and then we show that for each i there is an element g_i such that the order of g_i is $q_i^{\alpha_i}$. Indeed, one can easily show that $X^{q_i^{\alpha_i}} - 1$ has a root that is not a root of $X^{q_i^{\alpha_i - 1}} - 1$. To complete the proof, we take $g = g_1 \dots g_r$. $\square$

One can prove that there is a unique finite field F (up to isomorphism) with $|F| = p^n$ (we will not give a proof here). Yet, we summarize the main results on finite fields in the following theorem:

Theorem 3.4.2. *Let p be a prime and let $q = p^n$ for some $n \geq 1$. Then*

1. *There exists a finite field F with q elements.*
2. *Any two fields of q elements are isomorphic.*
3. *If F is a field of order q then the multiplicative group F^* is cyclic of order $q-1$.*
4. *The elements of F are the roots of the polynomial $X^q - X \in \mathbf{F}_p[X]$ and this polynomial factors into a product of distinct linear factors.*
5. *Every irreducible polynomial of degree n over $\mathbf{F}_p$ is a factor of $X^q - X$. The irreducible factors of $X^q - X$ are precisely the irreducible polynomials over $\mathbf{F}_p$ whose degrees divide n.*
6. *A field F of order q contains a subfield of order $q' = p^{n'}$ if and only if $n' \mid n$.*

The theorem then implies that a way to construct the unique (up to isomorphism) finite field of p^n elements is to choose any irreducible polynomial $f(X)$ of degree n over $\mathbf{F}_p$ and to consider the polynomials in $\mathbf{F}_p[X]$ modulo $f(X)$ under addition and multiplication. More precisely, the quotient ring

$$\mathbf{F}_p[X]/\langle f(X) \rangle \cong \mathbf{F}_{p^n}.$$

For those familiar with the theory of rings and ideals, $\langle f(X) \rangle \subset \mathbf{F}_p[X]$ denotes the ideal generated by $f(X)$.

Example 2. The simples example of a non-prime field is $p = 2$ and $n = 2$. In that case, the only irreducible polynomial of degree 2 over $\mathbf{F}_2$ is $f(X) = X^2 + X + 1$ and hence, $\mathbf{F}_4 \cong \mathbf{F}_2[X] \mod X^2 + X + 1$.

For all practical applications of finite fields, the choice of the polynomial $f(X)$ in the construction of $\mathbf{F}_{p^n}$ matters (a good example is the implementation of AES in symmetric key cryptography where a special choice of $f(X)$ for $\mathbf{F}_{2^{128}}$ allows for very efficient computations.

3.4.2 Irreducible polynomials over $\mathbf{F}_p$

Let $f(d)$ be the number of irreducible polynomials of degree d over $\mathbf{F}_p$. The polynomial $X^{p^n} - X \in \mathbf{F}_p[X]$ factors over $\mathbf{F}_p$ is a product of all irreducible polynomials of degree dividing n. This means that (comparing degrees)

$$\sum_{d|n} f(d)d = p^n \tag{3.2}$$

In order to invert (3.2), one uses the Möbius function $\mu(d)$ (look it up on google) and arrives at the following formula due to Gauss:

$$f(n) = \frac{1}{n}\sum_{d|n} p^d \mu\left(\frac{n}{d}\right).$$

This is asymptotically $\sim \frac{1}{n}p^n$ which means that a randomly chosen monic polynomial of degree n will be irreducible with probability $\frac{1}{n}$. Thus, to compute an irreducible polynomial, one can try choosing a random monic polynomial $f(X)$ of degree n and calculating $\gcd(f(X), X^{p^m} - X)$ for $m = 1, 2, \dots, \lfloor \frac{n}{2} \rfloor$ (if the gcd is trivial for every m then $f(X)$ is irreducible).

Remark 2. One should compare this result with the prime number theorem where the number $\pi(x)$ of primes primes in the interval $[1, x]$ is asymptotically $\frac{x}{\ln x}$. For example, if $x = 2^n$ then

$$\#\text{primes less than or equal to } 2^n \sim c\frac{1}{n}2^n \qquad \#\text{irred. poly's of degree } n = \tilde{c}\frac{1}{n}p^n,$$

for some constants $c, \tilde{c} > 0$.

3.4.3 Normal bases

Let k be an integer and let $\mathbf{F}_{p^k}/\mathbf{F}_p$ be the finite field extension of degree k of $\mathbf{F}_p$. Then $\mathbf{F}_{p^k}$ can be viewed as an $\mathbf{F}_p$-vector space of dimension k.

Definition 3.4.3 (normal basis of a finite field extension)**.** A basis $\mathcal{B}$ of $\mathbf{F}_{p^k}$ (as an $\mathbf{F}_p$-vector space) is called *normal* if

$$\mathcal{B} = \{\alpha, \alpha^p, \alpha^{p^2}, \dots, \dots \alpha^{p^{k-1}}\}$$

for some $\alpha \in \mathbf{F}_{p^k}$ that is a root of an irreducible polynomial $f(X) \in \mathbf{F}_p[X]$ of degree k.

The advantage of normal bases in cryptography is that they speed up arithmetic significantly. The reason for that is that computing the pth power of an element (or, its image under Frobenius) is simply a circular shift of the coefficients. Indeed,

$$\begin{aligned}\left(a_0\alpha + a_1\alpha^p + a_2\alpha^{p^2} + \cdots + a_{k-1}\alpha^{p^{k-1}}\right)^p &= a_0^p\alpha^p + a_1^p\alpha^{p^2} + \cdots + a_{k-1}^p\alpha^{p^k} = \\ &= a_{k-1}\alpha + a_0\alpha^p + \cdots + a_{k-2}\alpha^{p^{k-1}},\end{aligned}$$

since $a_i^p = a_i$ and $(u+v)^p = u^p + v^p$ for any $u, v \in \mathbf{F}_{p^k}$.

3.4.4 Inversion using normal bases

Computing multiplicative inverses in finite fields can be made fast using normal bases. Indeed, note that for any element $x \in \mathbf{F}_{p^k}^*$ and any integer r

$$x^{-1} = (x^r)^{-1}x^{r-1}.$$

Take $r = \dfrac{p^k - 1}{p - 1}$. Then $r - 1 = p + p^2 + \cdots + p^{k-1}$ and hence, one can compute x^{r-1} using at most $\log_2(k-1)$ multiplications in $\mathbf{F}_{p^k}$ and the (fast) Frobenius computations. The $\log_2(k-1)$ comes from the fact that

$$x^{p+p^2+\cdots+p^m} \cdot x^{p^m(p+\cdots+p^m)} = x^{p+\cdots+p^{2m}}.$$

One can thus quickly compute x^{r-1}. To compute $(x^r)^{-1}$, we observe that x^r is an element of $\mathbf{F}_p \subset \mathbf{F}_{p^k}$ and hence, computing its inverse is much faster.

3.4.5 Examples of normal bases

Let p be an odd prime and let q be a prime such that p is a primitive root modulo q, i.e., $p \bmod q$ is a generator of $\mathbf{F}_q^*$. One can show (see homework exercise) that

$$f(X) = \frac{X^q - 1}{X - 1} = X^{q-1} + \cdots + X + 1$$

is an irreducible polynomial of $\mathbf{F}_p[X]$.

In this case, we obtain an optimal normal basis as follows: suppose that α is a root of $f(X)$. Then it follows that α^{p^i} is a root of $f(x)$ as well for $i = 0, 1, \ldots, q-1$. In addition $\alpha^m = \alpha^{m \bmod q}$ for every m. Since $p \bmod q$ is a generator for $\mathbf{F}_q^\times$, it follows that $\{p \bmod q, p^2 \bmod q, \ldots, p^{q-1} \bmod q\}$ is a permutation of $\{1, \ldots, q-1\}$, i.e., $\{\alpha, \alpha^p, \ldots, \alpha^{p^{q-2}}\}$ is a normal basis for the field extension $\mathbf{F}_{p^{q-1}}$. We summarize this in the following:

Proposition 3.4.4. *Let p be an odd prime and let q be a prime such that p is a* *primitive root modulo q, i.e., $p \bmod q$ is a generator of $\mathbf{F}_q^*$. Then the polynomial*

$$f(X) = \frac{X^q - 1}{X - 1} = X^{q-1} + \cdots + X + 1 \in \mathbf{F}_p[X]$$

is irreducible and if α is a root of f then $\{\alpha, \alpha^p, \ldots, \alpha^{p^{q-2}}\}$ is a normal basis of the field extension $\mathbf{F}_{p^{q-1}}/\mathbf{F}_p$.

3.5 Exercises

Exercise 3.1. Let p be a prime. We refer to a set of integers $\{b_1, \ldots, b_{p-1}\}$ as a *complete residue system* mod p if the set $\{b_1 \mod p, \ldots, b_{p-1} \mod p\}$ is a permutation of $\{1, \ldots, p-1\}$.

1. Let $\{b_1, \ldots, b_{p-1}\}$ be a complete residue system. Show that if a is an integer that is relatively prime to p then $\{ab_1, \ldots, ab_{p-1}\}$ is again a complete residue system.

2. Use (a) to deduce Fermat's little theorem, i.e., if $\gcd(a, p) = 1$ then $a^{p-1} \equiv 1 \pmod{p}$.

3. How can you prove Euler's theorem with a similar idea?

Exercise 3.2. Let p be a prime. For $0 \leq k \leq p$, consider the binomial coefficients

$$\binom{p}{k} = \frac{p!}{k!(p-k)!}.$$

1. Prove that if $0 < k < p$ then $\binom{p}{k}$ is divisible by p.

2. Use (a) to show that if x and y are arbitrary integers then

$$(x + y)^p \equiv x^p + y^p \bmod p.$$

3. Use (b) and induction to show that if $x_1, \ldots, x_k$ are integers then

$$(x_1 + x_2 + \cdots + x_k)^p \equiv x_1^p + \cdots + x_k^p \bmod p.$$

   Use that to give another proof of Fermat's little theorem.

Exercise 3.3. Let $\varphi(n)$ be the Euler totient function (i.e., the number of integers in $\{1, 2, \ldots, n\}$ that are coprime to n). Show that $\sum_{d|n} \varphi(d) = n$.

Exercise 3.4. Recall how the RSA algorithm works. Moreover, given an odd integer N which is known to be a product of two distinct primes p, q, prove that computing $\phi(N) := (p-1)(q-1)$ is equivalent to factoring N. Note: this does *not* mean that breaking RSA is equivalent to factoring N.

Exercise 3.5. Let p be an odd prime and let q be a prime such that p (viewed as an element mod q and hence, of $\mathbf{F}_q$) is a primitive element of $\mathbf{F}_q^\times$ (i.e., that p generates the cyclic group $\mathbf{F}_q^\times$). Consider the polynomial

$$f(X) = \frac{X^q - 1}{X - 1} = X^{q-1} + \cdots + X + 1 \in \mathbf{F}_p[X].$$

Let α be a root of $f(X)$ and $\mathbf{F}_p(\alpha)$ be the finite field extension of $\mathbf{F}_p$ generated by α.

1. If $\mathbf{F}_p(\alpha) \cong \mathbf{F}_{p^d}$, show that $q \mid (p^d - 1)$.
2. Show that $d = q - 1$ and deduce that f is irreducible.
3. Show that $f(\alpha^{p^i}) = 0$ for all $i \in \{0, 1, \ldots, q-2\}$.
4. Let α be as above. Show that the set of elements $\{\alpha, \ldots, \alpha^{q-1}\}$ is the same as the set of elements $\{[\}]\alpha^{p^0}, \alpha^{p^1}, \ldots, \alpha^{p^{q-2}}$.
5. Deduce that $\{\alpha^{p^1}, \ldots, \alpha^{p^{q-1}}\}$ is a normal basis for $\mathbf{F}_{p^{q-1}}$ over $\mathbf{F}_p$. In fact, one calls this basis an *optimal normal basis.*

Exercise 3.6. Prove that if g is a primitive element (= multiplicative generator) of $\mathbf{F}_{p^n}^\times$ and if $d \mid n$ then $g^{(p^n-1)/(p^d-1)}$ is a primitive element of $\mathbf{F}_{p^d}^\times$.

Exercise 3.7. We have seen that we can construct $\mathbb{F}_9$ by adjoining to $\mathbb{F}_3$ the roots of an irreducible monic polynomial of degree 2.

1. Check that $f(X) = X^2 - X - 1 \in \mathbf{F}_3[X]$ is an irreducible polynomial and call α a root of this polynomial (in some algebraic closure of $\mathbf{F}_3$). All the elements of $\mathbf{F}_9$ can be written as $a + b\alpha$ for some $a, b \in \mathbf{F}_3$ (you don't need to prove this). Show that α is a generator of the multiplicative group $\mathbf{F}_9^\times$ (explain in detail your computations).

2. Find the discrete logarithm of $\alpha - 2$ to the base $\alpha - 1$, if it exists. That is, find an integer $n \in \mathbf{Z}$ such that $(\alpha - 1)^n = \alpha - 2 \in \mathbf{F}_9$.

Exercise 3.8 (SAGE)**.** Construct with SAGE the finite fields $\mathbf{F}_p$, $\mathbf{F}_{p^n}$ and $\mathbf{F}_p[X]/(P(X))$ for some irreducible polynomial $P(X)$ over $\mathbf{F}_p[X]$.

For various values of prime numbers p and integers $n > 1$ (e.g. $p = 23, n = 10$), find what irreducible polynomial $f(X) \in \mathbf{F}_p[X]$ is used in SAGE to construct the finite field $\mathbf{F}_{p^n}$.

Exercise 3.9 (SAGE)**.** We say that $x \in \mathbf{F}_p^\times$ is a square (or *a quadratic residue*) if there exists $y \in \mathbf{F}_p^\times$ such that $x = y^2$. Write a program counting how many squares there are in $\mathbf{F}_p^*$. Compute this number for all $p < 100$ and give a formula for this number. Prove this formula. Prove that $x \neq 0$ is a square if and only if $x^{\frac{p-1}{2}} = 1$ in $\mathbf{F}_p^*$.

Exercise 3.10 (SAGE)**.** Write in SAGE a function computing the euclidean division between two polynomials in $\mathbf{F}_p[X]$.

Exercise 3.11. In this exercise, we will study the following probabilistic algorithm which outputs a generator of $\mathbf{F}_p^*$, when p is a given odd prime number such that the factorization into prime powers of

$$p - 1 = \prod_{i=1}^{r} q_i^{e_i}$$

is known.

Algorithm 13 Probabilistic generator

```
Require: A prime $p$ and factorization $p - 1 = \prod_{i=1}^{r} q_i^{e_i}$.
Ensure: A generator $\gamma$ of $\mathbf{F}_p^*$.
1: for $i = 1, \ldots, r$ do
2:     $\beta_i := 1$
3:     while $\beta_i = 1$ do
4:         pick $\alpha \in \mathbf{F}_p^\times$ randomly
5:         $\beta_i := \alpha^{(p-1)/q_i}$
6:         $\gamma_i := \alpha^{(p-1)/q_i^{e_i}}$
7:     end while
8: end for
9: return  $\gamma := \prod_{i=1}^{r} \gamma_i$
```

1. Prove that the algorithm indeed returns a generator of $\mathbf{F}_p^*$ when it terminates.
2. At a given step i, what is the probability that $\beta_i = 1$ in the `while` loop? Deduce the expected number of trials before getting $\beta_i \neq 1$. (Hint : you can use the fact that the expected value of a geometric distribution with success probability p_0 is $1/p_0$).
3. Using the previous part, show that the expected running time of the algorithm is $O(\log(p)^d)$ for some integer $d > 0$.
4. (optional) Implement the algorithm in SAGE. You are allowed to use a factorization function from SAGE to pre-compute the factors $q_i^{e_i}$ of $p - 1$.

CHAPTER 4 Factoring Polynomials over Finite Fields

4.1 Polynomial rings and quotient rings of polynomial rings

4.1.1 Commutative rings with unity

Recall the following basic abstract algebra definition to be used throughout:

Definition 4.1.1 (Commutative ring with unity)**.** A commutative ring with unity R is a set, together with two group operations "$+$" (addition) and "$\cdot$" (multiplication) that satisfies the following properties:

1. R is an abelian group under $+$,
2. R is closed under multiplication, $\cdot$ is commutative, associative and there exists a multiplicative identity (unity) $1_R \in R$,
3. Multiplication and addition satisfy the distributive law, namely for any $a, b, c \in R$,
$$a \cdot (b + c) = a \cdot b + a \cdot c.$$

Remark 3. Examples of commutative rings with unity are $R = \mathbf{Z}$, $R = \mathbf{Z}/N\mathbf{Z}$ or $R = \mathbf{Z}[x]$. An example of a commutative ring without unity is $R = 2\mathbf{Z}$.

Remark 4. One easily shows the following two basic facts about commutative rings with unity:

1. The multiplicative identity 1_R is unique. Indeed, if $1'_R, 1''_R \in R$ are two multiplicative identities, then
$$1'_R = 1'_R \cdot 1''_R = 1''_R.$$
2. If 0_R is the zero element of the additive group of R then $0_R \cdot a = 0_R$ for any $a \in R$. Indeed, by the distributive law,
$$a \cdot 0_R = a \cdot (0_R + 0_R) = a \cdot 0_R + a \cdot 0_R \Longrightarrow a \cdot 0_R = 0_R.$$

Given a commutative ring R with unity, one can consider the set $R^\times$ of units in R (in other words those elements that have inverses in R). It is easy to check that $R^\times$ is a group under multiplication. Note that $\mathbf{Z}^\times = \{\pm 1\}$, $(\mathbf{Z}/N\mathbf{Z})^\times = \{a\colon \gcd(a, N) = 1\}$. In particular, $(\mathbf{Z}/N\mathbf{Z})^\times = \mathbf{Z}/N\mathbf{Z} - \{0\}$ if and only if N is prime.

Definition 4.1.2. Let R be a commutative ring with unity. We call a non-zero element $a \in R$ a zero divisor if there exists $a' \in R$, $a' \neq 0_R$ such that $a \cdot a' = 0_R$.

The ring $\mathbf{Z}$ has no zero divisors, whereas $\mathbf{Z}/N\mathbf{Z}$ has zero divisors if and only if N is composite. For instance, if $N = pq$ then $p \bmod N$ and $q \bmod N$ are both zero divisors.

4.1.2 Polynomial rings and quotient rings of polynomial rings

If K is a field then the polynomial ring in one variable over K is the set $K[X]$ of polynomials $p(X) = a_0 + a_1 X + \cdots + a_n X^n$ with coefficients $a_i \in K$. For the purpose of the course, we will be mainly interested in polynomials with coefficients in finite fields. Here, we recall basic facts about finite field extensions. The ring $K[X]$ enjoys several important properties.

First, we know that if $f(X), g(X) \in K[X]$ with $g(X) \neq 0$ then there exist unique polynomials $r(X)$ and $q(X)$ of $K[X]$ with $\deg r(X) < \deg g(X)$ such that

$$f(X) = g(X)q(X) + r(X).$$

This is simply the division with remainder property for polynomials. This makes $R[X]$ into what we call *Euclidean domain.*

4.1.3 The Chinese remainder theorem for quotient rings of polynomial rings

We will state it for polynomial rings only: suppose that K is a field and $f \in K[X]$ is a polynomial that factors as $f(X) = f_1(X)^{e_1} \dots f_r(X)^{e_r}$ where $f_1(X), \dots, f_r(X)$ are distinct irreducible polynomials. We have the following:

Theorem 4.1.3 (Chinese remainder theorem)**.** *The standard projections $K[X] \to K[X]/\langle f_i(X)^{e_i}\rangle$ yield a ring homomorphism*

$$R[X] \to R[X]/\langle f_1(X)^{e_1}\rangle \times \cdots \times R[X]/\langle f_r(X)^{e_r}\rangle,$$

whose kernel is exactly the ideal $\langle f(X)\rangle$. There is thus an isomorphism

$$R[X]/\langle f(X)\rangle \cong R[X]/\langle f_1(X)^{e_1}\rangle \times \cdots \times R[X]/\langle f_r(X)^{e_r}\rangle.$$

4.2 Factorization of polynomials

Let $\mathbf{F}_q$ be the finite field of q elements for some $q = p^k$. We will describe two classical algorithms for factoring polynomials over $\mathbf{F}_q$: Berlekamp's algorithm and Cantor–Zassenhaus algorithm.

4.2.1 Testing polynomials for repeated factors

Let $q = p^k$ for some prime p and let $f(X) \in \mathbf{F}_q[X]$ be a polynomial. Suppose that one wants to test whether there is a repeated factor in the decomposition of $f(X)$ into irreducibles. We start with the following observation:

Lemma 4.2.1. *If $f'(X) = 0$ then there exists a polynomial $g(X)$ such that $f(X) = g(X)^p$.*

Proof. Note that $f'(X) = 0$ if and only if $f(X) = \sum_{i=0}^{n} a_i X^{pi}$, this is to say $f(X) = g(X^p)$ where $g(X) = \sum_{i=0}^{n} a_i^{q/p} X^i$. But $g(X^q) = g(X)^q$. □

The lemma, together with the computation of gcd's of polynomials can be used in a recursive procedure to find the polynomial $h(X)$ that is the (square-free) product of all irreducible factors of $f(X)$ with no multiplicities: set initially $u(X) := f(X)$ and $h(X) := 1$ and keep applying Lemma 4.2.1 to $u(X)$ updating the latter with the corresponding polynomial $g(X)$ (from the Lemma) at each step until $u'(X) \neq 0$. Compute then $v(X) := \gcd(u(X), u'(X))$. If $v(X) = 1$ then $u(X)$ is square-free, update $h(X) := \operatorname{lcm}(h(X), u(X))$ and return $h(X)$. Otherwise, the polynomial $u(X)/v(X)$ is square-free of degree less than $\deg u$ and one updates $h(X) := \operatorname{lcm}(h(X), u(X)/v(X))$, $u(X) := v(X)$ and repeats the process with $u(X)$ and $h(X)$ until termination.

Since Berlekamp's algorithm works only for square-free polynomials, we can recover the irreducible factors of $f(X)$ by computing the irreducible factors of the resulting $h(X)$.

Example 3. Consider $f(X) = X^5(X+1)$ over $\mathbf{F}_5[X]$. Initially, we have $u(X) = X^5(X+1)$ and $h(X) = 1$. Then $u'(X) = X^5$ and $v(X) = \gcd(f(X), f'(X)) = X^5$, so we update $h(X) = (X+1)$ and $u(X) = X^5$. We apply Lemma 4.2.1 to update $u(X) := X$. Repeating the process terminates the algorithm with $h(X) := \operatorname{lcm}(X, X+1) = X(X+1)$.

4.2.2 A brief summary of Berlekamp's algorithm

Let $f(X) \in \mathbf{F}_q[X]$ be a polynomial of degree n that is square-free (i.e., that has no repeated factors). The quotient ring

$$R = \mathbf{F}_q[X]/\langle f(X) \rangle$$

is the set of all polynomials of the form $c_0 + c_1 X + \cdots + c_{n-1}X^{n-1}$, where addition and multiplication are first performed in the usual way and the result is then taken modulo $f(X)$.

The idea of Berlekamp's algorithm is that one can compute factors of $f(X)$ by taking gcd's with special polynomials from the ring R that satisfy the same polynomial equation as the elements of $\mathbf{F}_q$, namely,

$$g(X)^q - g(X) \equiv 0 \bmod f(X). \tag{4.1}$$

The reason this is the case is the following proposition

Proposition 4.2.2. *If $g(X)$ satisfies* (4.1) *then*

$$f(X) = \prod_{c \in \mathbf{F}_q} \gcd(f(X), g(X) - c). \tag{4.2}$$

Proof. Note that the polynomials $g(X) - c \in \mathbf{F}_q[X]$ are pairwise relatively prime as c ranges over $\mathbf{F}_q$. Since $\gcd(f(X), g(X) - c) \mid f(X)$ then

$$\prod_{c \in \mathbf{F}_q} \gcd(f(X), g(X) - c) \mid f(X).$$

Conversely, the polynomial $Y^q - Y \in \mathbf{F}_q[Y]$ factors as $Y^q - Y = \prod_{c \in \mathbf{F}_q} (Y - c)$, so substituting $Y = g(X)$, one gets

$$g(X)^q - g(X) = \prod_{c \in \mathbf{F}_q} (g(X) - c).$$

Now, if $f(X) \mid g(X)^q - g(X)$ then

$$f(X) = \gcd(g(X)^q - g(X), f(X)) = \prod_{c \in \mathbf{F}_q} \gcd(g(X) - c, f(X)).$$

This completes the proof. □

Next, consider the set of all such special polynomials:

$$B(f) = \{g(x) \in R\colon g(x)^q - g(x) \equiv 0 \bmod f(x)\}.$$

It is an easy exercise to show that $B(f)$ is a commutative ring with unity that is also a vector space over $\mathbf{F}_q$ (such a ring is commonly referred to as a commutative $\mathbf{F}_q$-algebra). It is known as the Berlekamp algebra. We now show how to calculate a basis for this algebra and thus, design a factoring algorithm based on (4.2).

Consider the set $\{1, x^q, x^{2q}, \ldots, x^{(n-1)q}\}$ of polynomials in $\mathbf{F}_q[x]$. Each of these polynomials can be written uniquely (division by $f(x)$ with remainder) as

$$x^{iq} = q_{i,0} + q_{i,1}x + \cdots + q_{i,n-1}x^{n-1} + f(x)q_i(x),$$

where $q_{i,j} \in \mathbf{F}_q$ for $j = 0, 1, \ldots, n-1$. Consider the $n \times n$-matrix Q over $\mathbf{F}_q$ whose $(i+1)$th column vector is the vector $v_i = \begin{bmatrix} q_{i,0} \\ q_{i,1} \\ \cdots \\ q_{i,n-1} \end{bmatrix}$. For any polynomial $g(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1} \in R$ one can show that $g(x) \in B(f)$ if and only if

$$(Q - I) \cdot v_g = 0,$$

where $v_g = \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_{n-1} \end{bmatrix}$ and I is the $n \times n$ identity matrix. Indeed, given a vector v representing some polynomial $v(x) = v_0 + v_1 X + \cdots + v_{n-1}X^{n-1}$ in R, consider the polynomial represented by the vector Qv. Then

$$v^q = (v_0 + v_1X + \cdots + v_{n-1}X^{n-1})^q = v_0 + v_1X^q + \cdots + v_{n-1}X^{(n-1)q} \equiv Qv \bmod f$$

If $v \in B(f)$ then $v^q \equiv v \bmod f$, so we get $(Q - I)v = 0$ in R. In other words, $g(x) \in B(f)$ if and only if v_g is in the kernel (null space) of the matrix $Q - I$. Next, we relate the dimension of the kernel to the number of irreducible factors of g.

Lemma 4.2.3. *The dimension of* $\ker(Q - I)$ *is equal to the number of irreducible factors of f in $\mathbf{F}_q$.*

Proof. Let $f = \prod_{i=1}^{r} g_i$ where g_i are irreducible and pairwise distinct. According to the Chinese remainder theorem,

$$\mathbf{F}_q[X]/\langle f(X)\rangle \cong \prod_{i=1}^{r} \mathbf{F}_q[X]/\langle g_i(X)\rangle,$$

where the isomorphism is given by $h \mapsto (h \bmod g_i)_{i=1}^r$. Now, each of the quotient rings $\mathbf{F}_q[X]/\langle g_i(X)\rangle$ is isomorphic to $\mathbf{F}_{q^{d_i}}$ where $d_i = \deg(g_i)$. Now, using

$$h^q = h \in R \Longleftrightarrow h_i^q = h_i,\ i = 1, \dots, r,$$

and $h_i^q = h_i \Longleftrightarrow h_i \in \mathbf{F}_q$, we see that $h \in \ker(Q - I)$ if and only if $(h_i)_{i=1}^r \in \mathbf{F}_q^r$ (the latter means that $\dim_{\mathbf{F}_q}(Q - I) = r$). □

Now, using reduction to reduced row echelon form, one can compute a basis for $B(f)$ by computing a basis for the kernel of $Q - I$. To get an algorithm, one takes a basis vector in the kernel and its corresponding polynomial $g(X)$ in the Berlekamp algebra $B(f)$ and computes $\gcd(f(X), g(X) - c)$, for c running over $\mathbb{F}_q$, until a non-trivial factor $h(X)$ of $f(X)$ is found. Since $\deg g(X) < \deg f(X)$ for any $g(X) \in B(f)$, if $\dim_{\mathbf{F}_q} B(f) > 1$ then there is at least one non-constant polynomial in $B(f)$ and this polynomial would always give us a non-trivial factorization. If one picks a constant polynomial $g(X) = c$ then one of the gcd's will certainly be $f(X)$ as $g(X) - c \equiv 0$. Once a non-trivial factor $h(X)$ of $f(X)$ is found, one then repeats the above algorithm with $f(X)/h(X)$.

If $n = \deg f(X)$, Berlekamp's algorithm can be implemented so as to use $O(n^\omega + n^{1+\varepsilon}q)$ operations in $\mathbf{F}_q$. Here, we assume that two $k \times k$ matrices can be multiplied using $O(k^\omega)$ arithmetic operations $(2 < \omega \leq 3)$. Moreover, we assume that two polynomials of degree k can be multiplied using $O(k^{1+\varepsilon})$ arithmetic operations for some $0 < \varepsilon \leq 1$ that depends on the choice of the multiplication algorithm. Although the running time is linear in q, the algorithm is deterministic. It is possible (using a refined method that we will not discuss here) to implement Berlekamp's algorithm so that it runs in expected polynomial time in $\log q$.

4.2.3 A brief summary of the algorithm of Cantor–Zassenhaus

In order to find the product of all factors of $f(X)$ of degree precisely n, one could use the fact that $\gcd(f(X), X^{q^n} - X)$ is the product of all irreducible factors of $f(X)$ whose degree is a divisor of n.

This leads to an efficient algorithm to compute the polynomials $f_d(X)$ that are the product of all irreducible factors of $f(X)$ of degree d. We therefore have to assume $f(X)$ square-free, which we can achieve in the same way as in the setup of Berlekamp's algorithm. To compute $f_d(X)$, we can proceed via the following steps:

1) Set $r = 1$
2) Calculate $f_r(X) = \gcd(f(X), X^{q^r} - X)$
3) If $r = d$, return $f_r(X)$
4) Update $f(X) := f(X)/f_r(X)$
5) Set $r := r + 1$ and return to 2)

That way, $f_d(X)$ will contain the product of all irreducible factors of $f(X)$ of degree d, so we can assume throughout that $f(X)$ is a square-free product of irreducible factors of degree d. We also assume that p is odd. The basic idea of the Cantor–Zassenhaus algorithm is simple: since $f(X) \mid X^{q^d} - X$ (assuming $f(X)$ has no factor X), we have

$$f(X) \mid X^{q^d-1} - 1 = (X^{(q^d-1)/2} - 1)(X^{(q^d-1)/2} + 1).$$

Thus, by computing $\gcd(f(X), X^{(q^d-1)/2} \pm 1)$, one could hope for obtaining a non-trivial factor of $f(X)$.

In order to make this idea work in practice, suppose that $f(X) = g_1(X) \dots g_s(X)$ where $\deg g_i(X) = d$ then (as in Berlekamp)

$$R := \mathbf{F}_q[X]/\langle f(X)\rangle \cong \mathbf{F}_q[X]/\langle g_1(X)\rangle \times \cdots \times \mathbf{F}_q[X]/\langle g_s(X)\rangle =: S.$$

The isomorphism is given by the map $h(X) \mapsto (h_1(X), \dots, h_s(X))$ that sends $h(X)$ to the s-tuple of reductions $h_i(X) = h(X) \bmod g_i(X)$. The algorithm uses crucially the observation that since g_i's are irreducible then each of the quotient rings $\mathbf{F}_q[X]/\langle g_i(X)\rangle$ is a finite field of q^d elements.

To explain how the algorithm works, we first state the following easy lemma (we omit the proof that is quite straightforward):

Lemma 4.2.4. *Let $h(X) \in R$ be a polynomial that satisfies $h(X) \neq 0, 1, -1$, but $h_i(X) = 0, 1, -1$, where $h_i(X)$ is the reduction of $h(X)$ modulo $g_i(X)$. Then at least two of the following three sets*

$$\begin{aligned} S_0 &= \{i \colon h_i(X) = 0\} \\ S_1 &= \{i \colon h_i(X) = 1\} \\ S_{-1} &= \{i \colon h_i(X) = -1\} \end{aligned}$$

are non-empty and one can get non-trivial factors of $f(X)$ by computing:

$$\begin{aligned}\gcd(f(X), h(X)) &= \textstyle\prod_{i\in S_0} g_i(X)\\ \gcd(f(X), h(X)-1) &= \textstyle\prod_{i\in S_1} g_i(X)\\ \gcd(f(X), h(X)+1) &= \textstyle\prod_{i\in S_{-1}} g_i(X).\end{aligned}$$

To describe the algorithm, let $f(X) = g_1(X)\cdots g_s(X) \in \mathbb{F}_q[X]$ be a product of s irreducible factors, each of degree d, and assume that $\mathbf{F}_q$ has odd characteristic. Set $m := (q^d-1)/2$, and pick a random polynomial $h(X) \neq 0, \pm 1 \in R$, with corresponding $(h_1(X), ..., h_s(X)) \in S$. If for some $1 \leq i \leq s$ we have $h_i(X) \neq 0$, then $h_i(X)^{q^d-1} = 1$, since $h_i(X)$ lives in a finite field with q^d elements. Therefore, $h_i(X)^m = 0, \pm 1$ for all $i = 1, ..., s$. Thus, if a randomly chosen polynomial $h(X) \in R$ happens to satisfy $h(X)^m \not\equiv 0, \pm 1 \mod f(X)$, then $h(X)^m$ is a candidate for Lemma 4.2.4, and we get a proper factor of $f(X)$ by computing $\gcd(f(X), h(X)^m)$, $\gcd(f(X), h(X)^m - 1)$ and $\gcd(f(X), h(X)^m + 1)$.

Thus, one can implement the algorithm as follows: one picks a random polynomial $h(X) \in R$ and computes $h(X)^m \mod f(X)$. If $h(X)^m \equiv 0, \pm 1 \mod f(X)$, we chose another $h(X)$ until we find one that satisfies $h(X)^m \not\equiv 0, \pm 1 \mod f(X)$. One then calculates the three gcd's: $\gcd(f(X), h(X)^m)$, $\gcd(f(X), h(X)^m - 1)$ and $\gcd(f(X), h(X)^m + 1)$. By Lemma 4.2.4, one of the three gcd's necessarily gives a non-trivial factor of $f(X)$. So, in order to estimate the probability of failure, we need an estimate of the probability of $h(X)^m$ being $0, \pm 1$.

Lemma 4.2.5. *The probability that a randomly chosen $h(X) \in R$ would satisfy*

$$h(X)^m \equiv 0, \pm 1 \ mod\ f(X)$$

is less than $1/2$.

Proof. Observing that $X^{q^d-1} - 1 = (X^{(q^d-1)/2} - 1)(X^{(q^d-1)/2} + 1)$, for each i, there are $m := (q^d - 1)/2$ polynomials $b_i(X)$ of degree less than d such that $b_i(X)^m \equiv 1 \pmod{g_i(X)}$. Similarly, there are m polynomials $b_i(X)$ such that $b_i(X)^m \equiv -1 \pmod{g_i(X)}$. Thus, the number of polynomials $b(X)$ such that $b(X)^m \equiv \pm 1 \pmod{f(X)}$ is at most $2m^s$. Excluding the constant polynomials, we get that the number of non-constant ones of degree less than n (with n the degree of $f(X)$, which equals $d \cdot s$) is $q^n - q$ and thus, the probability of failure is bounded by

$$\frac{2m^s - q + 1}{q^n - q} < 2^{1-s} \leq \frac{1}{2},$$

assuming that $s > 1$. □

CHAPTER 5 Integer Factorization

As we have already mentioned, integer factorization is probably the most fundamental tool for public-key cryptanalysis. The security of any RSA-related cryptographic scheme depends on the complexity of factoring large integers. You saw last time that one can efficiently factor polynomials of large degrees over large finite fields. As it is usually the case in number theory, many problems turn out to be harder for integers than for polynomials. This is the case with factorization. We now review some basic and more advanced algorithms (both exponential and subexponential on the number of digits of the integer).

5.1 Exponential-time versus subexponential-time algorithms

5.1.1 Trial division

The simplest factoring algorithm is trial division: try to divide n by the sequence of primes $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$. Since a composite n has always a prime less than or equal to $\sqrt{n}$ then there are at most $\pi(\sqrt{n}) = \sqrt{n}/\log\sqrt{n}$ primes to try. The method is useful for a randomly selected integer, but not useful at all for special integers such as RSA moduli.

5.1.2 Pollard's $p-1$-method

We now cover a basic, but important exponential factoring method, the Pollard's $p-1$ method. Understand it will be important as it will (later in the course) allow us to understand another factoring method, the elliptic curve factoring method (ECM) due to Hendrik Lenstra.

The main idea behind Pollard's $p-1$-method is based on Fermat's little theorem: if p is a prime then $p \mid 2^M - 1$ whenever $p - 1 \mid M$. For instance, given an integer B, if $M(B)$ denote the least common multiple of all integers up to B then (assuming one can compute $M(B)$) one can try to compute $\gcd(2^{M(B)} - 1, n)$ and hope this will produce a non-trivial factor.

The two main questions are thus how to compute efficiently $M(B)$ and then to understand when and how the above method can fail exactly.

For a given B, the complexity of the above algorithm is $\mathcal{O}(B \log B \log^2 n)$. The large the B is, the larger the complexity is, yet, the more likely it is that the algorithm will find a factor. Clearly, the complexity of the algorithm is determined by the largest prime q that divides $p-1$. If this prime happens to be relatively small, one can expect that the method will succeed.

Example 4. Suppose that $n = 10001$.

5.1.3 The L-notation for complexity

We will often need a function that measures precisely subexponential complexity. Let $0 \leq t \leq 1$ be a real number and let $\gamma \in \mathbf{R}_{>0}$. We define the function

$$L_x(t, \gamma) = e^{(\gamma + o(1))(\log x)^t (\log \log x)^{1-t}}, \qquad x \to \infty.$$

Here, x is a complexity parameter and t is used to measure complexity that is between polynomial and exponential. For example, note that $L_x(0, \gamma)$ represents polynomial complexity in x whereas $L_x(1, \gamma)$ represents exponential complexity. Any $0 < t < 1$ represents complexity that is both superpolynomial and subexponential. We will also occasionally use $L_x(t) = L_x(t, \bullet)$ for brevity when the constant γ is irrelevant.

5.2 Methods based on $x^2 \equiv y^2 \bmod n$

Various factoring algorithms are based on a simple idea that dates back to Fermat. We explain the development of this idea.

5.2.1 Fermat's factorization method

If one can write n of the form $n = a^2 - b^2 = (a-b)(a+b)$ for which $a - b > 1$ then one gets a non-trivial factorization. Now, if n is odd and $n = uv$ then n has always such a representation, i.e., $n = \left(\frac{u+v}{2}\right)^2 - \left(\frac{u-v}{2}\right)^2$. This gives rise to the following simple idea: if we find integers x and y such that $x^2 \equiv y^2 \bmod n$ then we can hope to get a non-trivial factor of n by computing $\gcd(x-y, n)$. Fermat's method attempts at trying $x = \lfloor \sqrt{n} \rfloor + 1, \lfloor \sqrt{n} \rfloor + 2, \ldots$ until $x^2 - n$ is a perfect square. In practice, this is quite slow and will only give an exponential time algorithm for factoring. Even worse, it is not even compatible with the basic trial division algorithm.

5.2.2 Dixon's Method and Morrison–Brillhart Approach

This method tries to avoid solving $x^2 \equiv y^2 \bmod n$ directly, but solves it via combining (via linear algebra) some congruences mod n that are easier to generate. Let $B > 0$ be a real number. We call an integer B-smooth if all of its prime factors are less than or equal to B. Let $\mathcal{P}(B)$ be the set of all prime numbers $\leq B$ (the *factor base*) and let $m = \#\mathcal{P}(B)$. The idea of the algorithm is to generate integers v, $1 \le v \le n$ such that

$$v^2 \equiv \prod_{p \in \mathcal{P}(B)} p^{e_p(v)} \bmod n, \tag{5.1}$$

for some exponents $e(p)$, i.e., such that $v^2 \bmod n$ is B-smooth. Once we have generated more than m such congruences, we can try to combine them (using linear algebra over $\mathbf{F}_2$ only) to produce a right-hand side of the congruence that is a perfect square. The phase where we generate the congruences will be called the *relation generation phase*, whereas the phase where we solve the linear system will be called the *linear algebra phase.*

Suppose that sufficiently many relations of type (5.1) have been generated. Each such v gives rise to a vector $(e_p(v))_{p\in\mathcal{P}(B)} \in \mathbf{Z}^m_{\ge 0}$. This step is going to try to find a subset S of the generated v's such that for all $p \in \mathcal{P}(B)$,

$$\sum_{v \in S} e_p(v) = 2s_p, \tag{5.2}$$

for some $s_p \in \mathbf{Z}$. Such a subset S should exist since the vectors $(e_p(v))_{p\in\mathcal{P}(B)}$ are linearly dependent (since the total number of relations is more than $m = \#\mathcal{P}(B)$). Such a subset S can be found using Gaussian elimination reducing (mod 2) the equations (5.2) for all $p \in \mathcal{P}(B)$.

The difficulty in the above method is a rigorous complexity analysis of the algorithm. Obviously, this will depend on the complexity of the individual phases reducing to the following questions:

Question 5.2.1. *Which method is used in the relation generation stage?*

Question 5.2.2. *Which linear algebra improvements are used (e.g., linear algebra with sparse matrices over* $\mathbf{F}_2$*)?*

Here, we will discuss mainly the first question and only mention that there is a large literature on the second question, i.e., the linear algebra phase [Wie86, LO90, Cop94].

For the relation generation phase, Dixon's method [Dix81] is probably the simplest. It consists of generating uniformly random integers v in the interval $[1, n]$ and testing $v^2 \bmod n$ for smoothness.

The method of Morrison–Brillhart is different: it produces smaller values of $v^2 \bmod n$ to test for smoothness than Dixon's method (by using continued fractions convergents rather than choosing random v's) and as such, increases the B-smoothness probabilities. One of the homework problems asks you to develop in detail some aspects of the continued fractions method. The algorithm of Morrison–Brillhart was used to factor the 7th Fermat number $F_7 = 2^{2^7} + 1$. You will study this algorithm in detail in your homework.

5.2.3 Smoothness bounds and smoothness probabilities

Let $x \ge 1$ be a real number, let $y \ge 1$ and let $\psi(x, y)$ be the number of y-smooth integers in the interval $[1, x]$. The function $\psi(x, y)$ has been well-studied. It is also related to a special function known as the Dickman–de Bruijn function[1] ρ.

Dickman first showed that for any fixed $u \geq 1$, $\psi(x, x^{1/u})$ has a non-zero limit as $x \to \infty$ [Dic30]. Later on, de Bruijn [dB51] studied more elaborately the asymptotic behavior of $\psi(x, y)$. We summarize these in the following:

Theorem 5.2.3. *(i) The Dickman–de Bruijn function $\rho(u)$ is non-negative for $u > 0$, decreasing for $u > 1$, and satisfies the asymptotic estimate*

$$\log \rho(u) = -(1 + o(1))u \log u, \qquad u \to \infty.$$

(ii) For any fixed $\epsilon > 0$, we have

$$\psi(x, x^{1/u}) \sim x\rho(u) \sim xu^{-u} \qquad \text{as} \qquad x \to \infty.$$

provided that $u \leq \log(x)^{1-\epsilon}$.

We will not prove it here, but will rather apply it to a basic optimization problem that turns out to be quite useful in the analysis of several cryptographic algorithms (see the exercises from the new Homework 6).

Finally, the function $L_x(t, \gamma)$ is useful when we estimate smoothness probabilities as the following exercise suggests:

Proposition 5.2.4 (smoothness probabilities)**.** *Let $\alpha, \beta, r, s \in \mathbf{R}_{>0}$ be given with $s < r \leq 1$. The probability that a random positive integer less than or equal to $L_x(r, \alpha)$ is $L_x(s, \beta)$-smooth is $L_x(r - s, -\alpha(r - s)/\beta)$ as $x \to \infty$.*

You will prove this on the new homework as well. All of these results will be used throughout the course, so you are encouraged to do the exercises thoroughly.

5.2.4 Analysis of Dixon's method

One method that we can rigorously analyse via Proposition 5.2.4 is Dixon's method. To simplify the analysis, we assume that we use trial division in the test for B-smoothness. To apply Proposition 5.2.4, we write $B = L_n(s, \beta)$ where s and β will be determined subsequently.

Proposition 5.2.5. *The optimal smoothness bound B is obtained for $s = 1/2$ and $\beta = 1/2$. For that B, one obtains opitimal complexity of Dixon's algorithm that is $L_n(1/2, 2)$.*

Proof. The size of the factor base is $\#\mathcal{P} = \pi(B) = L_n(s, \beta)$, so we need to collect that many relations. The time it takes to test a given number for B-smoothness is then $L_n(s, \beta)$. By Proposition 5.2.4, the probability that a random number $\leq n = L_n(1, 1)$

[1]The Dickman–de Bruijn function $\rho(u)$ is a continuous function that satisfies the delay differential equation

$$u\rho'(u) + \rho(u - 1) = 0.$$

with initial conditions $\rho(u) = 1,\ 0 \leq u \leq 1$. A heuristic argument by Dickman shows that $\rho(u) = \frac{\psi(x, x^{1/u})}{x}$

is B-smooth is then $L_n(1-s, -(1-s)/\beta)$ and hence, the run-time for the relation collection stage is

$$\underbrace{L_n(s,\beta)}_{\text{test for smoothness}} \cdot \underbrace{L_n(s,\beta)}_{\text{\# of relations needed}} \cdot \underbrace{L_n(1-s,-(1-s)/\beta)^{-1}}_{\text{inverse of smoothness probability}} = L_n(s,2\beta)\cdot L_n(1-s,(1-s)/\beta).$$

It is not hard to show that the optimal run-time is achieved when $s = 1-s$, i.e., $s = 1/2$. In this case, the runtime is $L_n\left(\frac{1}{2}, 2\beta + \frac{1}{2\beta}\right)$ which is minimized for $\beta = 1/2$, thus, getting a complexity of $L_n(1/2, 2)$. □

5.3 Basics sieving methods - linear and quadratic sieves

The sieving techniques are more efficient alternatives to Dixon's method or the method of Morrison–Brillhart to the relation-generation phase.

The idea is based on the following observation: if $v(i) = i + \lfloor\sqrt{n}\rfloor$ where i is small then we see that

$$v(i)^2 - n \sim 2i\sqrt{n},$$

hence, these values are smaller compared to the values generated by Dixon's method. Moreover, if $p \mid v(i)^2 - n$ then $p \mid v(i+pj)^2 - n$ (again, this assumes that i as well as $i + pj$ are small). This observation gives us a way to much more efficiently test for smoothness via a sieve: let L be a bound and consider all $v(i)$ for $0 < i \leq L$ at once. Take a sequence $\{s(i)\}_{i=1}^{L}$ where we initially set $s(i) = (i + \lfloor\sqrt{n}\rfloor)^2 - n$ originally. We then proceed as follows: for each p in the factor base $\mathcal{P}(B)$, find the roots of the polynomial $f(x) = (x + \lfloor\sqrt{n}\rfloor)^2 - n$ modulo p and for each root r, divide $s(r+pj)$ by p for all possible j's. After that procedure, we will know that for numbers that are B-smooth, $s(i)$ will be 1. The efficiency gain here is that we are not testing individually sampled numbers for smoothness, but test all numbers at the same time with a single loop over the factor base (finding the roots r is easy!).

5.3.1 Linear sieve

Before the above idea evolved, there was a sieve method proposed by Schroeppel called the *linear sieve*. Here, the idea was to test for smoothness numbers of the form

$$\left(i + \lfloor\sqrt{n}\rfloor\right)\left(j + \lfloor\sqrt{n}\rfloor\right) - n \ \sim\ (i+j)\sqrt{n}.$$

Compared to the method of Morrison–Brillhart, the numbers generated by the linear sieve were larger. Yet, their smoothness can be tested via a sieve method (as above). The disadvantage to the above method is that the left-hand side is not a square and hence, each of the numbers $(i + \lfloor\sqrt{n}\rfloor)$ had to go into the matrix in the linear algebra phase. Yet, since the runtime of Morrison–Brillhart back in the days was not analyzed yet, at least in theory, Schroeppel's method was the first one to give a heuristic subexponential time factoring method. Schroeppel's idea was to factor the 8th Fermat number $F_8 = 2^{2^8} + 1$ and he managed to succeed with the relation generation stage. Yet, before he ran the linear algebra phase, the number was factored (somehow by chance) via the Pollard rho method.

5.3.2 Quadratic sieve

This is the basic idea of using $i = j$ in the linear sieve was due to Pomerance [Pom82, Pom85] and led to the development of the quadratic sieve. We should point out the qualitative difference in complexity with Dixon's method (that has optimal complexity $L_n(1/2, 2)$). To analyze the most basic version of this method, choose a smoothness bound $B = L_n(1/2, 1/2)$ and assume that $v^2 \bmod n$ is a number around $\sqrt{n} = L_n(1, 1/2)$. One then calculates naïvely that the smoothness probability is $L_n(1/2, -1/2)$.

Thus, if we are interested in finding $L_n(1/2, 1/2)$ different relations, we need to test

$$L_n(1/2, 1/2) \cdot L_n(1/2, -1/2)^{-1} = L_n(1/2, 1).$$

different integers i. Our sieving array length L is then $L_n(1/2, 1)$. This is already better than the Dixon's method whose complexity was calculated to be $L_n(1/2, 2)$.

Remark 5. We should point out that $v(i)^2 - n$ is not quite behaving as a random number: indeed, if p is a prime dividing $v(i)^2 - n$ then n is a square modulo p, i.e., only about half of the primes $\leq B$ occur in the factorisation of $v(i)^2 - n$. One then expects that the size of the factor base is about $\pi(B)/2$. The smoothness probabilities are not too different from the above naïve smoothness probabilities. **Dimitar** : **Give a reference.**

5.4 Number Field Sieve

The number field sieve is currently the most practical general-purpose integer factorization method. It is based on an idea of Pollard who tried to factor integers of the form $x^3 + d$ for small d. For a detailed account of the method, see [LL93b].

5.4.1 The idea of the general number field sieve

Even if one is not able to analyze it rigorously, the method outperforms the quadratic sieve even for integers of less than 100 digits. The conceptual reason for that is that the numbers tested for smoothness are of order $n^{o(1)}$ as $n \to \infty$ whereas the quadratic sieve methods use numbers of size n^c for a fixed c.

Let $f \in \mathbf{Z}[x]$ be a monic irreducible polynomial of degree d and suppose that m is an integer such that $f(m) \equiv 0 \bmod n$. Let α be a root of f. The map $\alpha \mapsto m$ then gives a homomorphism $\phi\colon \mathbf{Z}[\alpha] \to \mathbf{Z}/n\mathbf{Z}$. Suppose that we find a set $\mathcal{S}$ of polynomials g such that

$$\prod_{g \in \mathcal{S}} g(\alpha) \text{ is a square in } \mathbf{Z}[\alpha] \tag{5.3}$$

and

$$\prod_{g\in\mathcal{S}} g(m) \text{ is a square in } \mathbf{Z}. \tag{5.4}$$

Then we will get the congruence

$$x^2 = \phi\left(\prod_{g\in\mathcal{S}} g(\alpha)\right) = \prod_{g\in\mathcal{S}} g(m) \equiv y^2 \bmod n.$$

Then we have a chance of factoring n. We have already seen the example of the factoring methods based on solving the congruence $x^2 \equiv y^2 \bmod n$, so in order for that to work, there are several questions we need to discuss:

Question 5.4.1. *What is a suitable notion of smoothness in* $\mathbf{Z}[\alpha]$ *so that we can combine various algebraic numbers* $g(\alpha)$ *to get a square? How do we factor* $g(\alpha)$ *intro primes to get a vector of exponents?*

Question 5.4.2. *How do we guarantee that the* $g(m)$*'s are small so that we can study their probabilities of smoothness?*

5.4.2 The choice of m and f

Before we approach Question 5.4.1, we tackle Question 5.4.2. Suppose that we fix the degree of our polynomial f, say d (we will later on discuss how to choose the d). We start by choosing the m by setting $m = \lfloor n^{1/d} \rfloor$. We then write n in base m, i.e.,

$$n = c_d m^d + c_{d-1} m^{d-1} + \cdots + c_0, \qquad 0 \le c_i < m, \ \forall i = 0, 1, \ldots, d.$$

since $m^d \le n < (m+1)^d$, if n is sufficiently large compared to d (see homework exercise), we can show that $c_d = 1$. Consider then the monic polynomial $f(t) = t^d + c_{d-1}t^{d-1} + \cdots + c_0$. Since most of the monic polynomials of a given degree and integer coefficients are irreducible, odds are that f will be irreducible (see the homework for the case when f turns out to be reducible).

Assume that f is irreducible. For the polynomials g we can then take $g(x) = a - bx$ where $0 < b \le B$ and $|a| \le B$ for some bound B. If B is chosen sufficiently small, the numbers $g(m)$ will then be quite small and hence, we may hope to test them efficiently for smoothness. In addition, for a fixed b we have the advantage of being able to sieve over the a's in a similar way as in the quadratic sieve.

Remark 6. Assume that it happens that f is reducible, i.e., $f(t) = g(t)h(t)$ for non-constant polynomials g and h. In this case, we expect that $g(m)h(m)$ will yield a non-trivial factorisation of n. This is indeed proven by Brillhart, Filaseta and Odlyzko [BFO81].

Remark 7. Expressed in terms of ideals, the NFS algorithm is based on the following high-level idea: if α is a root of the (irreducible) polynomial f, one can consider the ideal $I = \langle \alpha - m, n\rangle$. One than wants to find algebraic numbers $u, v \in \mathbf{Q}(\alpha)$ such that $u^2 - v^2 \in I$. If one then considers the ideal $J = \langle u \pm v, \alpha - m, n\rangle$, chances are that computing the gcd of $N(J)$ and n will produce a non-trivial factor.

5.4.3 The norm map $N\colon \mathbf{Q}(\alpha) \to \mathbf{Q}$

We now have to go back to Question 5.4.1 and decide how to test the elements $g(\alpha) \in \mathbf{Z}[\alpha]$ for smoothness. In general, the ring $\mathbf{Z}[\alpha]$ will not be what we call a *unique factorization domain.* For instance, if $f(x) = x^2 + 5$ then $\alpha = \sqrt{-5}$ and we see that

$$6 = 2 \cdot 3 = (1 + \sqrt{-5}) \cdot (1 - \sqrt{-5}),$$

where the elements $2, 3, 1 \pm \sqrt{-5} \in \mathbf{Z}[\sqrt{-5}]$ are what we call irreducible. We thus need a different way of testing for smoothness than just trying to factor $g(\alpha)$.

One thing to try is to use the norm map. Let $\alpha = \alpha_1, \alpha_2, \dots, \alpha_d$ be all the roots of $f(t)$ in $\mathbf{C}$. We define

$$N(h(\alpha)) := \prod_{i=1}^{d} h(\alpha_i), \qquad h(x) \in \mathbf{Q}[x].$$

It is clear that if γ is a square in $\mathbf{Z}[\alpha]$ then $N(\gamma)$ is a square in $\mathbf{Z}$. The latter is a necessary, but not a sufficient condition for γ to be a square in $\mathbf{Z}[\alpha]$. Yet, we start with arranging for that condition to hold first. To do that, note that

$$N(a - b\alpha) = \prod_{i=1}^{d}(a - b\alpha_i) = b^d \prod_{i=1}^{d}(a/b - \alpha_i) = b^d f(a/b) = a^d + c_{d-1}a^{d-1}b + \cdots + c_0 b^d =: F(a, b).$$

Define $F(a,b) = b^d f(a/b)$. Then F is a homogeneous polynomial of degree d in two variables.

If B is a smoothness bound, we call an algebraic integer $a - b\alpha \in \mathbf{Z}[\alpha]$ B-smooth if $N(a - b\alpha)$ is B-smooth. With that, we can (via the same linear algebra method as before) combine smooth algebraic integers so the resulting norm is a perfect square. Yet, even if $N\left(\prod_{g \in \mathcal{S}} g(\alpha)\right)$ is a perfect square in $\mathbf{Z}$, it does not follow that $\prod_{g \in \mathcal{S}} g(\alpha)$ is a perfect square in $\mathbf{Z}[\alpha]$ and this is one of the challenges we will have to resolve.

Remark 8. Consider the Gaussian integers $\mathbf{Q}(i)$ and note that $N(2) = 4$ is a perfect square. Yet, it is easy to check that 2 is not a perfect square in $\mathbf{Q}(i)$.

Before we discuss the question of sufficiency, we note that we will be sieving for tuples (a, b) such that $F(a,b)G(a,b)$ is smooth where $G(a,b) = a - mb$. We will set the linear algebra stage for $F(a,b)$ and $G(a,b)$ separately to ensure that they are both squares in $\mathbf{Z}$. We will thus need to collect twice as many relations, i.e., $> 2\pi(B)$ as opposed to $> \pi(B)$ in the case of the quadratic sieve.

5.4.4 Several remarks on sieving

Here, we discuss the first idea to tackle the problem of the squareness of $a - b\alpha$. Although it will not solve it completely, it will be a good step forward.

Let $p \leq B$ be a prime and let r be an element of $\{0, \dots, p-1\}$ such that $f(r) \equiv 0 \bmod p$.

One can thus define $v_{p,r}(a - b\alpha)$ to be the exponent of p dividing $F(a,b)$ if $a \equiv br \bmod p$ and 0 otherwise. Let $\mathbf{v}(\gamma) = (v_{p,r}(\gamma))_{p,r}$ for all $p \leq B$ and $0 \leq r < p$.

Proposition 5.4.3. *Let $\mathcal{S}$ be a set of coprime pairs (a,b) such that each $a - b\alpha$ is B-smooth in the sense that its norm $N(a - b\alpha)$ is B-smooth as an integer. Let $K = \mathbf{Q}(\alpha)$ be the number field generated by α and let $\mathcal{O}_K$ denote its ring of integers. If $\prod_{(a,b)\in\mathcal{S}} (a - b\alpha)$ is a perfect square in $\mathcal{O}_K$ then we have*

$$\sum_{(a,b)\in\mathcal{S}} \mathbf{v}(a - b\alpha) \equiv 0 \bmod 2.$$

Proof. Here, we will be using an important property of the ring $\mathcal{O}_K$, namely, that it admits a unique factorization of prime ideals (in other words, that $\mathcal{O}_K$ is a Dedekind domain; this may be a new notion for you, but it should not scare you; all we will be using is the unique factorization of ideals property).

Given r and p as above, consider the ideal $\langle p, r - \alpha\rangle \subset \mathcal{O}_K$. First, it is not the unit ideal since the norm of the element $r - \alpha$ is $N(r - \alpha) = f(r) \equiv 0 \bmod p$, i.e., is divisible by p. Denote by $P_1, \ldots, P_k$ all prime ideals of $\mathcal{O}_K$ dividing $\langle p, r - \alpha\rangle$.

Now, take the unique factorization of the principal ideal $\langle a - b\alpha\rangle$ and consider the exponents $a_1, \ldots, a_k$ of the prime ideals $P_1, \ldots, P_k$, respectively in the unique factorization of that principal ideal. One can see that if a given $a_i > 0$ then $a \equiv br \bmod p$ and hence, all the a_i's are positive. In addition, the only prime divisors of $\langle p\rangle$ that divide $\langle a - b\alpha\rangle$ are $P_1, \ldots, P_k$. This tells us that the p-part of the norm of the algebraic integer $a - b\alpha$ is exactly the norm of $P_1^{a_1} \ldots P_k^{a_k}$, i.e.,

$$p^{v_p(a-b\alpha)} = N(P_1)^{a_1} \cdot \cdots \cdot N(P_k)^{a_k}.$$

Let $N(P_i) = p^{e_i}$. Then

$$\sum_{(a,b)\in\mathcal{S}} v_{p,r}(a - b\alpha) = \sum_{(a,b)\in\mathcal{S}} \sum_{i=1}^{k} a_i e_i = \sum_{i=1}^{k} e_i \sum_{(a,b)\in\mathcal{S}} v_{P_i}(a - b\alpha).$$

Since $\prod_{(a,b)\in\mathcal{S}} (a - b\alpha)$ is assumed to be a square in $\mathcal{O}_K$, it follows (again, using the unique factorization of ideals) that $\sum_{(a,b)\in\mathcal{S}} v_{P_i}(a - b\alpha)$ and hence,

$$\sum_{(a,b)\in\mathcal{S}} v_{p,r}(a - b\alpha) \equiv 0 \bmod 2,$$

for all p and all r that satisfy $f(r) \equiv 0 \bmod p$. $\square$

5.4.5 Several remarks on the “sufficiency of the squareness of the norm"

As mentioned above, the squareness of the norm (in $\mathbf{Z}$) of an algebraic integer $\alpha \in \mathbf{Z}[\alpha]$ does not imply the squareness of α as an element of $\mathbf{Z}[\alpha]$. We saw that if we pick any $a \in \mathbf{Z}$ that is a non-square and view it as an element of $\mathbf{Z}[i]$ then $N(a) = a^2$ is a perfect square, but a is not a square in $\mathbf{Z}[i]$. Even if this is the case, the ring $\mathbf{Z}[i]$ has an advantage to a general ring $\mathbf{Z}[\alpha]$, namely, $\mathbf{Z}[i]$ happens to be a unique factorization domain (i.e., we can factor uniquely (up to what we call a unit)) elements of that ring into a product of irreducibles. As mentioned above $\mathbf{Z}[\sqrt{-5}]$, for instance, is not a unique factorization domain.

We saw that even if $\mathcal{O}_K$ is not a unique factorization domain, it is at least a domain with unique factorization of ideals, so we can use that to make a step forward via Proposition 5.4.3. For concreteness, let $\beta = \prod_{(a,b)\in\mathcal{S}} (a - b\alpha)$. We saw that, at least if $\mathbf{Z}[\alpha] = \mathcal{O}_K$, we can ensure that $\langle\beta\rangle = I^2$ for some ideal $I \subset \mathcal{O}_K$.

There are the following obstructions remaining:

1. If $\mathbf{Z}[\alpha] = \mathcal{O}_K$ then we at least have $\langle\beta\rangle$ being the square of an ideal J in $\mathcal{O}_K$. Yet, it may not be the case that $\mathbf{Z}[\alpha] = \mathcal{O}_K$.
2. Even if $\langle\beta\rangle = I^2$ there is no guarantee that I will be a principal ideal.
3. Even if $I = \langle\gamma\rangle$ is principal for some $\gamma \in \mathcal{O}_K$, there is no guarantee that $\beta = \gamma^2$.
4. Even if $\beta = \gamma^2$ for some $\gamma \in \mathcal{O}_K$, there is no guarantee that $\gamma \in \mathbf{Z}[\alpha]$.

Note the following

- Obstruction (i) is related to the index of the quotient group $\mathcal{O}_K/\mathbf{Z}[\alpha]$.
- Obstruction (ii) is related to the class group Cl_K.
- Obstruction (iii) is related to the unit group $\mathcal{O}_K^\times$.

We start by addressing the last issue. One can use the following lemma for that:

Lemma 5.4.4. *Suppose that $f(x) \in \mathbf{Z}[x]$ is a monic irreducible polynomial and let $\alpha \in \mathbf{C}$ be a root of f. Let $K = \mathbf{Q}(\alpha)$ and let $\mathcal{O}_K$ be the ring of algebraic integers of K. Given any $\beta \in \mathcal{O}_K$, we have $f'(\alpha)\beta \in \mathbf{Z}[\alpha]$.*

Proof. □

Having this lemma, we can set the square-finding problem in $\mathcal{O}_K$ as opposed to $\mathbf{Z}[\alpha]$. Indeed, if we setup the problem for $\prod(a - b\alpha)$ being a square γ for $\gamma \in \mathcal{O}_K$ then to get a square in $\mathbf{Z}[\alpha]$, all we do is take $f'(\alpha)\gamma$.

To overcome the rest of the difficulties, one can use an idea of Adleman: let $x = y^2 \in \mathbf{Z}$ be a perfect square and let p a prime number. The Legendre symbol $\left(\dfrac{y^2}{p}\right)$ is $(y^2)^{(p-1)/2} \equiv y^{p-1} \equiv 1 \bmod p$ (by Fermat's theorem). Thus, if $\left(\dfrac{x}{p}\right)$ is equal to 1 for many primes, then x is a square with high probability.

How do we use this observation to test whether an integer $x \in \mathbf{Z}$ is a perfect square: if we select k random primes $q < |x|$ and test whether $\left(\dfrac{x}{q}\right) = 1$ for each of this q's. If this is the case, at least heuristically, the probability that x is not a perfect square will be 2^{-k}.

Proposition 5.4.5. *Let $\mathcal{S}$ be such that $\prod_{(a,b)\in\mathcal{S}} (a - b\alpha)$ is a square in $\mathbf{Z}[\alpha]$. Let q be an odd prime number, $s \in \{0, \ldots, q-1\}$ satisfying $f(s) \equiv 0 \bmod q$, $a - bs \not\equiv 0 \bmod q$ for all $(a,b) \in \mathcal{S}$ and $f'(s) \not\equiv 0 \bmod q$. Then $\prod_{(a,b)\in\mathcal{S}} \left(\frac{a-bs}{q}\right) = 1$.*

5.4.6 On taking square roots

Even if the above methods tell us that both $\prod_{(a,b)\in\mathcal{S}}(a - b\alpha) = \gamma^2$ and $\prod_{(a,b)\in\mathcal{S}}(a - bm) = v^2$ are squares, they do not tell us how to compute the square roots. From that point of view, the rational side of the NFS is easier and is very much the same as in the quadratic sieve.

However, the problem of computing the square root γ is more challenging. In this case, we need to find $\gamma = a_0 + a_1\alpha + \cdots + a_{d-1}\alpha^{d-1}$ and then consider $u = a_0 + a_1 m + \cdots + a_{d-1}m^{d-1}$. This means that we need to compute $a_i \bmod n$.

There are two approaches for doing the latter:

1. Compute $a_i \bmod p$ for lots of small primes p and use the Chinese remainder theorem.

2. Fix a prime p and compute $a_i \bmod p^n$ for many n (Hensel's lifting).

Whereas approach i) is more efficient from the point of view of large integer arithmetic (we do not need to compute with large integers if we keep the p's small), this approach suffers from the drawback that modulo each p, there are two distinct square roots and we do not know which one exactly we should take.

Approach ii) does not have this problem; yet, it requires larger integer arithmetic as the exponent n gets larger.

5.4.7 On the complexity analysis

We first start with explaining some of the intuition behind the algorithm and only formalize that later. The numbers we want to find smooth are all of the form $(a - bm)N(a - b\alpha)$. Here, a and b will run over coprime pairs of integers with $|a| \le B$ and $0 < b \le B$. Our starting point are the following observations:

- In order to keep $a - bm$ small, we want to choose a large d since $m = \lfloor n^{1/d} \rfloor$.

- We can also bound the size of $N(a - b\alpha) = F(a,b)$ by $(d+1)B^d m \le (d+1)B^d n^{1/d}$. The size of $(a - bm)N(a - b\alpha)$ is thus bounded by $(d+1)B^{d+1}n^{2/d}$. This shows that we do not want to choose d too large.

- There is a tradeoff in the choice of B: to keep the smoothness probabilities large, we need a small B; yet, if B is too small then we may not have enough pairs (a,b) such that $(a - bm)N(a - b\alpha)$ is smooth.

- A lower bound for the run-time of the algorithm is B^2 (sieving over all pairs (a,b)).

We thus reduce the question to an optimization problem for d and B in terms of n (you may see some of the rigorous analysis on the homeworks). Overall, the complexity of the number field sieve is $L_n(1/3, (64/9)^3)$ which makes it asymptotically better than the quadratic sieve algorithm.

Remark 9. It should be pointed out that the above analysis is not a rigorous run-time analysis. It makes the heuristic assumption that the norms $N(a - b\alpha)$ behave like random integers.

5.4.8 More details on the run-time analysis

Great survey references for the analytic number theory estimates and their application in the analysis of the sieving algorithms:

- Granville's survey - [Gra08a],
- Pomerance ICM paper - [Pom95],
- Pomerance - paper on multiplicative independence [Pom96],

CHAPTER 6 The Discrete Logarithm Problem over Finite Fields

6.1 Introduction to the Discrete Logarithm Problem

The discrete logarithm problem for a cyclic group is extremely simple to state: let N be a positive integer and let $\mathbf{G}$ be a cyclic group of order N generated by an element g (we write $\mathbf{G} = \langle g \rangle$).

Discrete Logarithm Problem: Given $h \in \mathbf{G}$, find $\log_g h$, that is, find the unique $x \in [0, N-1]$ for which $h = g^x$.

As stated, the problem is rather meaningless unless one specifies the explicit representation of the group $\mathbf{G}$. We take some simple examples to illustrate how different the problem can be on different presentations[1] of the elements of a cyclic group:

1. If $\mathbf{G} = \mathbf{Z}/N\mathbf{Z}$ is represented by $\{0, 1, \dots, N-1\}$ with the addition law being the integer addition taken modulo N and if $g = 1$, the above question is completely trivial since the discrete log of any element x is x itself. What happens is g is not 1?

2. Suppose that p is a prime and that $\mathbf{G} = (\mathbf{Z}/p\mathbf{Z})^\times$ is represented as the non-trivial residues $\{1, 2, \dots, p-1\}$ modulo p where the group law is integer multiplication taken modulo p, the problem is significantly harder. In this case, it is not even obvious how much effort is needed to explicitly find a generator g. Although it is not known how to solve the problem in polynomial time in $\log p$, it is possible to solve this question in subexponential time (in $\log p$).

Note that in ii), one can exponentiate efficiently, i.e., given $g \in \mathbf{G}$ and $x \in [0, p-1]$, one can compute g^x in time polynomial in $\log p$.

The discrete logarithm problem easily reduces to the case when the order of the group is prime. In fact, you will do this in the first exercise for the course:

Exercise 6.1. Suppose that $|\,\mathbf{G}\,| = p_1^{e_1} \dots p_k^{e_k}$. Given a generator $g \in \mathbf{G}$ and $h = g^x \in \mathbf{G}$, use the Chinese remainder theorem to show how to reduce the computation

[1]Note that the word presentation here is not exactly the same as the typical group-theoretic meaning of presentation where we present a group with generators and relations.

of x to the computation of x modulo $p_i^{e_i}$ for every $i = 1, \dots, k$. Explain why it is not a good idea to use cyclic groups whose order is a *smooth integer*, i.e., an integer that factorizes into a product of small primes. This algorithm is basic and is known as the Pohlig–Hellman method.

Yet, the lack of a polynomial time algorithm for solving the problem in ii) is more or less assumed as one of the building axioms of cryptography nowadays allowing for the design of various schemes whose security is built upon this axiom.

Question 6.1.1. *Are there other presentations of a cyclic group that make the problem even harder?*

Any attempt to find such a presentation will mean that this presentation does not allow one to apply **index calculus**, the only known algorithm for solving the discrete logarithm problem for $(\mathbf{Z}/p\mathbf{Z})^\times$ in subexponential time (to be discussed in detail). We will discuss this method in more details in this lecture.

6.2 The Pollard's rho method

6.2.1 Description of the algorithm

The Pollard's rho algorithm was proposed in 1975 as an integer factorization method [Pol75]. A few years later, Pollard realized that this method [Pol78] can be used to solve the *discrete logarithm problem* on a generic cyclic group. Here, *generic* means that the application of the algorithm is independent of representation of the group elements (i.e., it can work for any representation as long as the computation of the group is efficient). Let $\mathbf{G}$ be a cyclic group of prime order N and let $g \in \mathbf{G}$ be a generator. The Pollard's rho method aims to find two distinct pairs (a', b') and (a'', b'') such that $g^{a'} h^{b'} = g^{a''} h^{b''}$. The occurrence of finding these distinct pairs is called a collision. When a non-trivial collision has been found, the solution to the discrete logarithm problem is $x \equiv (a' - a'')(b'' - b')^{-1} \bmod |\,\mathbf{G}\,|$.

The algorithm is based on computing a simple dynamical system: given an iteration function $f\colon \mathbf{G} \to \mathbf{G}$, the Pollard's rho method calculates a sequence of points $x_{i+1} = f(x_i)$, $i \geq 0$, starting from a point $x_0 \in \mathbf{G}$ chosen uniformly at random. Given $x_i = g^{a_i} h^{b_i}$ and $a_i, b_i \in [0, n-1]$, f computes x_{i+1}, a_{i+1} and b_{i+1} such that $x_{i+1} = g^{a_{i+1}} h^{b_{i+1}}$. The original iteration function (dynamical system) proposed by Pollard is

$$x_{i+1} = f(x_i) = \begin{cases} gx_i, & \text{if } x_i \in P_0 \\ x_i^2, & \text{if } x_i \in P_1 \\ hx_i, & \text{if } x_i \in P_2 \end{cases}$$

where $\mathbf{G} = \bigsqcup_{i=0}^{2} P_i$ is partitioned into three disjoint sets P_0, P_1 and P_2. This sequence of points eventually collides (as operations are performed over a finite cyclic group). Denote by λ and $\mu \geq 1$ the smallest numbers such that $x_\lambda = x_{\lambda+\mu}$ holds. The value λ is called the tail and μ the cycle length. The advantage of the Pollard rho method is that it can be implemented using a constant amount of memory by using

Floyd's cycle finding method [Knu97, Ex. 3.1.6] or other elementary cycle detection algorithms. This is an improvement to, e.g., Shanks' baby-step-giant-step method [Sha71] that has the same run-time, but uses non-constant memory.

Typically, we consider a random partition among a set of *admissible* partitions, that is, partitions for which P_0, P_1 and P_2 have approximately the same size $N/3$. The choice of the partition is relevant since it might affect the complexity analysis of the algorithm.

6.2.2 Complexity analysis of Pollard rho

Although deceptively simple, a rigorous complexity analysis of Pollard's rho method for the discrete logarithm problem is currently out of scope. Heuristically, the algorithm runs in $\mathcal{O}(\sqrt{|\mathbf{G}|})$. The reason the run-time is heuristic is that the model of analysis that one uses to rigorously show this bound (Model 2 below) has not been properly compared to Model 1. In both models, one views the collision time T as a random variable[2] and describes the statistics of this random variable. Ideally, one would like to prove a statement according to which with high probability, a collision occurs after $\mathcal{O}(\sqrt{|G|})$ steps. Yet, we need to specify what we mean when we say *with high probability.*

Model 1 (random partitions). Here, it is assumed that the partition is chosen uniformly at random among a set of admissible partitions and thus, the distribution of T is determined by the randomness of that partition only. Let $X_0, X_1, \ldots, X_n, \ldots$ be the random variables indicating the position of the walk after step i.

Model 2 (pseudo-random walks). This aims at approximating Model 1. The problem is that we know very little about quantifying this approximation. The idea behind the approximation is that instead of using the random partition $\mathbf{G} = \bigsqcup_{i=0}^{2} P_i$, at each step k, X_{k+1} will be obtained from X_k by using (with probability $1/3$) a random move out of the three possible moves. This gives us the model of a random walk on what one calls the Pollard rho graph (a 3-regular graph whose vertices are the elements of $\mathbf{G}$ and whose edges are determined by the moves). Of course, once we obtain a collision, the walk should no longer be random, but deterministic. This is why we talk about pseudo-random walks. In this case (for the classical Pollard rho), one can show rigorously that the collision time is $\mathcal{O}(\sqrt{|\mathbf{G}|})$. In fact, this was not known until very recently: the desired bound of $\mathcal{O}(\sqrt{|\mathbf{G}|})$ was rigorously established using Markov chains by Kim, Montenegro, Peres and Tetali [KMPT10] (see also [MV06] for a weaker, but easier bound). Note that the squaring step is essential for obtaining the correct upper bound. Both arguments rely on establishing rapid mixing results for random walks in the Pollard rho graph.

[2]T is a stopping time.

6.3 Index calculus for prime fields and for extension fields

Index calculus is an algorithm for solving discrete log based on the notion of a *factor base*. The idea is that one first computes the discrete logarithm for a subset of *small* elements of the group (the factor base) by collecting sufficiently many relations and then solving a system of linear equations. Then, one computes the logarithm of a particular element using the calculated logarithms of the elements of the factor base. In order to be able to successfully apply such an algorithm, one typically needs a *norm* on the group elements. The algorithm was first discovered by Kraitchik in 1922 and was later on rediscovered by Merkle in 1977 in the context of public-key cryptography. The most commonly used version of the algorithm is due to Adleman. An improvement of Coppersmith for finite fields of small characteristic leads to a subexponential run-time $L_{|\mathbf{G}|}(1/3)$ (currently, the most efficient known algorithm).

6.3.1 Discrete logarithm over prime fields

To illustrate the idea, we consider the case $\mathbf{G} = \mathbf{F}_p^\times$ and the elements of $\mathbf{G}$ viewed as elements of $\{0, 1, \ldots, p-1\}$. We thus have a natural definition norm, so as a factor base, one can try to use the subset $P \subset \mathbf{F}_p^\times$ of primes of norm $\leq B$ for some suitably chosen smoothness bound B. The relations will then be of the form

$$g^u = \prod_{\ell \in P} \ell^{\alpha_{u,\ell}}, \tag{6.1}$$

where g is a generator of $\mathbf{F}_p^\times$ and $u \in \{0, 1, \ldots, p-2\}$. Such a relation results in a relation

$$u \equiv \sum_{\ell \in P} \alpha_{u,\ell} \log_g \ell \mod p-1. \tag{6.2}$$

Thus, if we collect more than $|P|$ relations, we can hope for solving for the values $\log_g p$ by linear algebra. This observation, together with the smoothness probabilities discussed in Section 5.2.3 already give us an algorithm that we can efficiently implement and rigorously analyze: we picks $u \in \{0, 1, \ldots, p-2\}$ at random and compute $g^u \bmod p-1$ and test for smoothness. Since g^u is distributed uniformly at random, we can use the smoothness probabilities to calculate the chance of success. For the second stage, the idea is quite similar except that we choose a random u now test $hg^u \bmod p-1$ for being B-smooth. If it happens that $hg^u = \ell_1^{\beta_1} \ldots \ell_k^{\beta_k} \bmod p$, we calculate

$$\log_g h = \beta_1 \log_g \ell_1 + \cdots + \beta_k \log_g \ell_k - u.$$

For the complexity analysis, one can use smoothness bound $L_p(1/2, \sqrt{1/2})$ as well as the smoothness probabilities discussed above to show that the relation collection stage takes time $L_p(1/2, \sqrt{2})$. Solving the (sparse) linear system takes also time $L_p(1/2, \sqrt{2})$. The runtime for the computation of the discrete logarithm given the discrete logarithms of the primes in the factor base is $L_p(1/2, \sqrt{1/2})$. Thus, the above version of index calculus gives $L_p(1/2, \sqrt{2})$ to solve the discrete logarithm problem.

6.3.2 Discrete logarithm over extension fields

Here, we assume that $q = p^n$ for some $n > 1$ and we look at $\mathbf{G} = \mathbf{F}_q^\times$. Let $g \in \mathbf{F}_q^\times$ will be a generator.

Assumption 1 : We will also assume that p is fixed (e.g., $p = 2$) and $n \to \infty$.

The elements of $\mathbf{F}_{p^n}^\times$ are represented as polynomials of degree at most n over $\mathbf{F}_p$. The key notions in the prime field case were *prime element* and a *norm*. It turns out that if $n > 1$, the norm of an element $\alpha \in \mathbf{F}_{p^n}^\times$ can be defined as $p^{\deg(f)}$ where $f \in \mathbf{F}_p[X]$, $\deg(f) < n$ is the polynomial representing α. One can declare that α is prime if f is irreducible.

Exercise 6.2. Given a polynomial of degree n and the notions developed in the previous problem, discuss how you would test that an element of $\mathbf{F}_{p^n}^\times$ represented by a given polynomial is B-smooth for some B. What is the time complexity of this test?

Let $B = L_{p^n-1}(1/2, \beta)$ for $\beta > 0$ be a smoothness bound and let $P_1, \ldots, P_k$ be irreducible polynomials of $\mathbf{F}_p[X]$ of degrees less than or equal to B. We can use the smoothness test from the previous exercise to do the relations collection stage: pick a random integer $v \in [0, p^n - 1]$ and test g^v for smoothness. If it happens to be smooth (i.e., equal to $P_1^{e_{1,v}} \ldots P_k^{e_{k,v}}$ then you add the relation

$$v = e_{1,v} \log_g P_1 + \cdots + e_{k,v} \log_g P_k \bmod p^n - 1.$$

We have the following result (the latest improvement of which is due to Soundararajan) about smoothness probabilities for polynomials (see [Gra08b, §5.2]):

Theorem 6.3.1. *Suppose that m is a smoothness bound and that $N_p(n, m)$ is the number of monic polynomials of degree n in $\mathbf{F}_p[X]$ that are products of irreducible polynomials each of which has degree at most m. If $u = n/m$ then one has the following asymptotic formula:*

$$N_p(n, m) = \frac{p^n}{u^{(1+o(1))u}},$$

uniformly over all $p \geq (n \log^2 n)^{1/m}$ for $u \to \infty$ and $m \to \infty$.

Exercise 6.3. What is the total time complexity for gathering sufficiently many relations so that you can solve for $\log_g P_1, \ldots, \log_g P_k$? Express your answer as L_n and the number β that appears in the smoothness bound. Observe that if p is fixed, and $m = B = L_{p^n-1}(1/2, \beta)$, the formula is applicable, so you can approximate the smoothness probability with u^{-u} (here, you need to be a bit more careful since your polynomials may have degrees less than n, but you should be able to show that the probability can still be approximated by $(n/m)^{-(n/m)}$).

6.4 Applications of discrete log

Let $\mathbf{F}_q$ be a finite field and let g be a generator for the multiplicative group $\mathbf{F}_q^\times$. Given $h \in \mathbf{F}_q^\times$, the discrete logarithm problem is the problem of computing x such that $g^x = h$. The security of multiple cryptographic schemes rely on the difficulty of solving this problem.

6.4.1 ElGamal encryption

The protocol consists of three components: key generator, encryption algorithm and decryption algorithm.

Key generation: Alice generates a multiplicative cyclic group $\mathbf{G}$ of order q and a generator g. Alice then chooses a random $x \in [0, q-1]$ and computes $h = g^x$. Alice's **public key** is $(\mathbf{F}_q, q, g, h)$. Alice's **private key** is x.

Encryption: To encrypt a message m to Alice under her public key, Bob chooses a random $y \in [0, q-1]$ (the **ephemeral key**) and computes $c_1 = g^y$ and $s = h^y$ (a shared secret). Bob then converts the message m into an element $m' \in \mathbf{G}$ and then calculates $c_2 = m' \cdot s$. Bob sends the ciphertext $c = (c_1, c_2)$ to Alice.

Decryption: The decryption algorithm works as follows: to decrypt the ciphertext $c = (c_1, c_2)$, Alice first calculates the shared secret $s = c_1^x$ with her private key and then computes $c_2 \cdot s^{-1}$. The decryption is correct since

$$c_2 \cdot (c_1^x)^{-1} = m' \cdot h^y \cdot (g^{xy})^{-1} = m' \cdot g^{xy} \cdot g^{-xy} = m'.$$

The ElGamal cryptosystem is typically used as a key encryption key system (i.e., one uses it to encrypt a key that is used to encrypt a long message via a symmetric key encryption protocol). This allows for encryption of messages with arbitrary lengths without directly converting them to elements of $\mathbf{G}$.

6.4.2 Diffie–Hellman key exchange

The Diffie–Hellman key exchange protocol is a basic method for key exchange between two parties, Alice and Bob. As in the case of ElGamal, a cyclic group $\mathbf{G}$ of order q is generated and a generator g of $\mathbf{G}$ is computed. Alice picks a random $a \in [0, q-1]$ and Bob picks a random $b \in [0, q-1]$. Alice sends g^a to Bob and Bob sends g^b to Alice. Both parties can compute g^{ab}. It is clear that an adversary only knows (g, g^a, g^b) and needs to compute g^{ab} (the computational Diffie–Hellman problem or CDH). If one can solve discrete log, one can obviously solve CDH. The most serious limitation of Diffie–Hellman protocol in practice is the lack of authentication. Communications using Diffie–Hellman all by itself are vulnerable to man-in-the-middle attacks. Ideally, Diffie–Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium.

4.2.1 Other applications.

The hardness of the discrete logarithm problem is a fundamental assumption not only for encryption schemes, but also for signature schemes, such as the digital signature scheme (DSA), the elliptic curve digital signature scheme (ECDSA), pairing-based signature schemes. In addition, the Massey–Omura protocol, a three-pass protocol for sending messages that allows one party to securely send a message to a second party without the need to exchange or distribute encryption keys, is also based on DLP.

CHAPTER 7 Introduction to Elliptic Curves

7.1 Why elliptic curve groups or groups of points on Jacobians of curves?

Here, we motivate with a very particular complexity comparison why elliptic curves are attractive for cryptography in practice. In order to do this, we do not need to know much about elliptic curves yet besides the fact that the best currently known attack on ECDLP on a curve E over $\mathbf{F}_p$ (at least on the curves used in practice) is a generic Pollard rho attack that runs with complexity $\mathcal{O}(2^{n/2})$ where n is the number binary digits of p.

7.1.1 Practical implications

Recall from last time that the complexity of the classical Pollard rho method on a generic cyclic group G was $C_{\mathrm{PR}}(n) = 2^{n/2}$, where n is the number of binary digits of $|G|$ (here, we ignore the constant which is rather small in practice - for the classical Pollard rho with three partitions, it is about $C = 1.4$ and decreases, getting closer to the constant $C = \sqrt{\pi/2}$ from the birthday bound as we increase the number of partitions). Moreover, we mentioned (see [Odl84] for details) that index calculus for $\mathbf{F}_p$ has complexity $L_p(1/3, c)$ where $c = (64/9)^{1/3} \sim 1.92$. We should point out that this is the same complexity as the complexity of the general number field sieve method for factoring integer that is used in breaking RSA (see [LL93a]). In terms of the number N of binary digits of p, the complexity is $C_{\mathrm{IC}}(N) = \exp(c(\log N)^{1/3}(\log\log N)^{2/3})$. To compare the security of the elliptic curve discrete log versus the finite field discrete log, we need the two key sizes n and N to satisfy $C_{\mathrm{PR}}(n) = C_{\mathrm{IC}}(N)$. Some algebra shows that this occurs whenever

$$n = \gamma(\log N)^{1/3}(\log\log N)^{2/3},$$

where $\gamma = 2c/(\log 2)^{2/3} \sim 4.91$. This means that for $N = 1024$, the corresponding $n = 173$. This is already a huge improvement making elliptic curves rather attractive as opposed to finite fields when it comes down to the difficulty of discrete log. The shorter key sizes have significant advantages for implementations on constrained devices such as mobile phones, smart cards, RFIDs, sensor networks, etc., thus making elliptic curves rather attractive from a purely practical point of view (see also [BSS00, Ch.I].

7.2 Introduction to elliptic curves

Basic references for this part are [ST92] and [Sil92]. We postpone the more thorough discussion until after we describe the background on algebraic geometry. For the moment, we restrict ourselves to Weierstrass equations only in order to get some basic idea of what is involved in elliptic curve cryptography.

7.2.1 Weierstrass equations

Let F be any field and let $\overline{F}$ be the algebraic closure of F. For the moment (until we properly introduce the notion of genus), an elliptic curve for us will be a projective curve in $\mathbf{P}^2(\overline{F})$ given by the following cubic equation:

$$E\colon Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3, \tag{7.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in F$. We would further require that the curve is *smooth*, i.e., that the three partial derivatives $\partial G/\partial X$, $\partial G/\partial Y$ and $\partial G/\partial Z$ do not simultaneously vanish at any given point $P = (X, Y, Z)$ on E (another ways to say the last condition is to say that the curve does not have a singular point). We can detect this last condition using only the coefficients $a_1, a_2, a_3, a_4, a_6 \in F$ by defining what is called the discriminant of the elliptic curve E. For the moment, we will postpone the formulas and refer to [Sil92, §III.1] or [BSS00, Ch.III] for the precise formulas. We should mention that we are interested not on the Weierstrass equations, but on Weierstrass equations up to isomorphism. We will postpone the definition of an isomorphism until the next lecture. The projective Weierstrass equation (7.1) has a corresponding affine Weierstrass model

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{7.2}$$

obtained (whenever $Z \neq 0$) by the substitution $x = X/Z$ and $y = Y/Z$.

For the moment, we will say that two Weierstrass equations E and E' over a field F with variables (x, y) and (x', y') will be isomorphic if and only if there exist $u \in F^\times$ and $r, s, t \in F$ such that the change of coordinates

$$x = u^2x' + r, \qquad y = u^3y' + su^2x' + t \tag{7.3}$$

transforms E into E'. Clearly, any such transformation can be inverted. Note that the transformation is defined over F. It is possible that two curves are not isomorphic over F, but become isomorphic over a field extension of F.

Exercise 7.1. Read Section III.1 from Silverman and explain why if $\operatorname{char} F \neq 2, 3$, then the general Weierstrass equation (7.2) of E can be transformed via an isomorphism of type (7.3) to a short Weierstrass equation:

$$E\colon y^2 = x^3 + Ax + B, \tag{7.4}$$

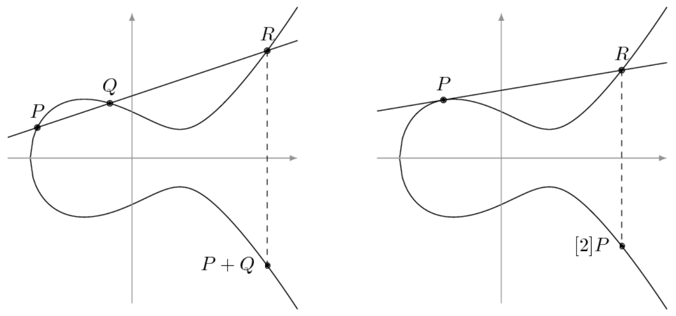

Figure 7.1: Elliptic curve group law on a short Weierstrass model $y^2 = x^3 + ax + b$.

Short Weierstrass equations are convenient since we can explicitly write the discriminant $\Delta = -16(4A^3 + 27B^2)$ of the curve and the j-invariant $j = -1728\frac{(4A)^3}{\Delta}$ in terms of the two coefficients A and B (note the j-invariant is an invariant of the isomorphism class).

Exercise 7.2. Show that the curve defined by (7.4) is smooth if and only if $\Delta \neq 0$. How does one interpret the latter condition in terms of the roots of the cubic polynomial $X^3 + AX + B$?

Finally, here is a simple, but important exercises that you should try:

Exercise 7.3. Suppose that we look at short Weierstrass equations over $\mathbf{F}_q$, that is

$$E\colon y^2 = x^3 + Ax + B, \qquad A, B \in \mathbf{F}_q.$$

Using the isomorphism transformation (7.3), compute the number of short Weierstrass equations that are isomorphic to a given one $y^2 = x^3 + A_0x + B_0$.

7.2.2 The group law

What makes elliptic curves useful for cryptography is their group law. In order to add two points $P, Q \in E(F)$, we draw the line connecting P and Q and consider the third intersection point $P \oplus Q$ with the curve. We then connect O_E and $P \oplus Q$ with a line and define $P + Q$ to be the third intersection point R of that line with the curve (see Fig. 7.2.2). If either P and Q or O_E and $P \oplus Q$ happen to coincide, we simply consider the tangent line through the point (which exists since the curve is smooth by assumption). It is not hard to check that $P + O_E = P$, that $P + Q = Q + P$ and that every element has an inverse.

What is less trivial is showing associativity of the operation '$+$', namely,

$$(P + Q) + R = P + (Q + R). \tag{7.5}$$

The proof can be derived from the following lemma in algebraic geometry about plane cubic curves:

Lemma 7.2.1. *Suppose that C_1, C_2 and C are three cubic curves in projective space such that C passes through eight out of the nine intersection points*[1] *of C_1 and C_2. Then C passes through the ninth point as well.*

The intuition behind the proof is that the family of cubic curves going through a eight distinct points is essentially 1-dimensional and hence, if F_1 and F_2 are the equations of C_1 and C_2, respectively, then the equation for C is $\lambda_1 F_1 + \lambda_2 F_2$ (note that the set of cubics $\lambda_1 F_1 + \lambda_2 F_2$ is 1-dimensional). Thus, if both F_1 and F_2 vanish at the ninth point then F also vanishes at that point[2]. We are now ready to establish the associative law (7.5).

Proposition 7.2.2. *The line through R and $(P+Q)$ intersects the line through P and $Q+R$ at a point S that lies on the curve E.*

Proof. The idea is to construct two cubics C_1 and C_2 such that C_1 and C_2 intersect at the nine points $O_E, P, Q, R, P \oplus Q, Q \oplus R, P+Q, Q+R$ and the point S. Equivalently, the associative law (7.5) holds. To show how to apply Lemma 7.2.1, we construct first C_1 and C_2. Given any two points $P_1, P_2 \in E(F)$, let ℓ_{P_1,P_2} be the line through P_1 and P_2. Consider C_1 to be the cubic that is the product of the linear forms defining $\ell_{P,Q+R}, \ell_{Q,R}$ and $\ell_{O_E,P+Q}$. Similarly, C_2 is the cubic that is the product of the linear forms $\ell_{P+Q,R}, \ell_{P,Q}$ and $\ell_{O_E,Q+R}$. It is clear that the three cubics intersect at the three points $O_E, P, Q, R, P \oplus Q, Q \oplus R, P+Q, Q+R$ and S, and that the Weierstrass E passes through eight of these points. By Lemma 7.2.1, E passes through S as well which proves the statement and thus, the associative law. $\square$

7.2.3 Addition and doubling formulae

One can write formulas for addition and doubling using Weierstrass coordinates for E and following the procedure described in the previous section (we will not spend too much time on that). If $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two points on a Weierstrass equation E given by (7.2), we let

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \qquad \mu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

when $x_1 \neq x_2$ and

$$\lambda = \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3}, \qquad \mu = \frac{-x_1^3 + a_4 x_1 + 2a_6 - a_3 y_1}{2y_1 + a_1 x_1 + a_3}$$

when $x_1 = x_2$. If $P_3 = P_1 + P_2 \neq O_E$ then $P_3 = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \qquad y_3 = -(\lambda + a_1)x_3 - \mu - a_3.$$

Exercise 7.4. Using what you have seen in the previous section, if $P = (x, y)$ is a point on E, compute the coordinates of the point $-P$.

[1] To be fully precise, one has to count intersection points with their multiplicities. For the sake of clarify of the argument, we will ignore these technicalities.

[2] Note that this is far from being a proof as the intuitive argument is not properly considering all possible degeneracies.

7.3 Elliptic curves over finite fields and Hasse–Weil bounds

Let $\mathbf{F}_q$ be the finite field with q elements. We will consider elliptic curves given by Weierstrass equations defined over $\mathbf{F}_q$. A key endomorphism of the curve E is the qth power *Frobenius endomorphism* $\varphi\colon E \to E$ defined on the $\overline{\mathbf{F}}_q$-rational points as follows:

$$\varphi\colon E(\overline{\mathbf{F}}_q) \to E(\overline{\mathbf{F}}_q), \qquad (x, y) \mapsto (x^q, y^q).$$

Although we have not yet introduced isogenies and endomorphisms properly, we introduce this endomorphism now since it is key for detecting which points on E are defined over $\mathbf{F}_q$. The reason is that the $\mathbf{F}_q$-rational points on E are precisely the points that are fixed by φ, i.e., that satisfy $\varphi(P) = P$ (the reason is that the Galois group $\mathrm{Gal}(\overline{\mathbf{F}}_q/\mathbf{F}_q)$ is *topologically* generated by φ). We will now use that to sketch a proof (modulo some facts that we will discuss in more detail in the future) of one of the important results for elliptic curves over finite fields, namely, the Hasse–Weil bounds:

Theorem 7.3.1. *Let $E/\mathbf{F}_q$ be an elliptic curve. Then*

$$|\#E(\mathbf{F}_q) - q - 1| \leq 2\sqrt{q}.$$

Remark 10. Before we explain the proof, we outline an argument for why we expect (heuristically) such a theorem to be true. Indeed, suppose that $x \in \mathbf{F}_q$ is picked at random. If we substitute that x into the Weierstrass equation, we obtain a quadratic equation for y which (if it has a solution) would generically have two solutions. Thus, we expect at most $2q$ points. Yet, if we assume (heuristically) that the quadratic equation is solvable with probability $1/2$ over the choice of x, we get approximately q solutions. The theorem makes the above argument rigorous.

The proof will use a simple, but powerful application of Cauchy–Schwarz inequality for quadratic forms. Before that, we introduce some basics on quadratic forms. Suppose that G is an abelian group. A map $q\colon G \to \mathbf{Z}$ is a quadratic form if the associated form $B\colon G \times G \to \mathbf{Z}$ defined by $B(x, y) = q(x+y) - q(x) - q(y)$ is bilinear, i.e., $B(x_1 + x_2, y) = B(x_1, y) + B(x_2, y)$ and $B(x, y_1 + y_2) = B(x, y_1) + B(x, y_2)$. Furthermore, q is called *positive definite* if $q(x) \geq 0$ for any $x \in G$.

Lemma 7.3.2. *(Cauchy–Schwarz) Let G be an abelian group and let $q\colon G \to \mathbf{Z}$ be a positive definite quadratic form with an associated bilinear form $B(x, y) = q(x+y) - q(x) - q(y)$. Then for all $x, y \in G$, we have*

$$|B(x, y)| \leq 2\sqrt{q(x)q(y)}.$$

Proof. Since q is a positive definite quadratic form, for any $\alpha, \beta \in \mathbf{Z}$, we have

$$0 \leq q(\alpha x + \beta y) = \alpha^2 q(x) + \alpha\beta B(x, y) + \beta^2 q(y).$$

Since this is true for any $\alpha, \beta \in \mathbf{Z}$ (for fixed x and y), the discriminant of the quadratic polynomial $F(z) = q(x)z^2 + L(x,y)z + q(y)^2$ has to be less than or equal to 0 which yields the desired inequality. □

In order to explain the proof, we need to mention a few important notions about endomorphisms of elliptic curves to be introduced and thoroughly discussed throughout the course:

- We will be studying maps (morphisms) $\phi\colon E \to E$ that map the point at infinite to itself and that are given by rational functions on the coordinates. Any such map is necessarily a homomorphism on the group of points. These maps are called endomorphisms and will appear several times for important applications. The endomorphisms (under point addition and composition) form a ring $\mathrm{End}(E)$ called the **endomorphism ring** of the elliptic curve.
- Every endomorphism is either trivial (sending all points to the point at infinity) or surjective with a finite kernel.
- Every endomorphism $\phi\colon E \to E$ has a *degree* $\deg(\phi)$. In order to properly define a degree, we need to introduce fields of rational functions on curves. In certain *nice* cases (the **separable** case), this degree coincides with the cardinality of $\ker(\phi)$. This is NOT the case with the Frobenius endomorphism $\varphi\colon E \to E$ (**purely inseparable** case) in which case the degree is q; yet, the kernel is trivial. Yet, the endomorphism $1 - \varphi$ is always separable, so the kernel is equal to the degree.
- The multiplication-by-m map $[m]\colon E \to E$ is an endomorphism (showing that $\mathbf{Z} \subset \mathrm{End}(E)$ and that $\mathrm{End}(E)$ is a $\mathbf{Z}$-module) and $\deg([m]) = m^2$.
- The degree map $\deg\colon \mathrm{End}(E) \to \mathbf{Z}_{>0}$ is a positive definite quadratic form.

Throughout the course, we will formally introduce and study the ring $\mathrm{End}(E)$ (under addition and composition) of endomorphisms of E.

(Proof of Theorem 7.3.1). As we already mentioned, $P \in E(\mathbf{F}_q)$ if and only if $\varphi(P) = P$, i.e., if and only if $P \in \ker(1 - \varphi)$. Now, the map $1 - \varphi$ is an endomorphism of E that (as mentioned above) is separable. Hence, $\#\ker(1-\varphi) = \deg(1-\varphi)$. We apply Lemma 7.3.2 to $q = \deg$, $x = 1$ and $y = -\varphi$ to obtain

$$|q + 1 - \deg(1-\varphi)| \leq 2\sqrt{q},$$

i.e., to get $|q + 1 - \#E(\mathbf{F}_q)| \leq 2\sqrt{q}$. □

Remark 11. The proof of this very simple, but fundamental statement already shows that it is not only elliptic curves themselves that are important to consider, but also algebraic maps between curves. This will be key for many applications related to discrete log.

Exercise 7.5. Let $\mathbf{F}_q$ be a finite field where q is odd and let $f(x) = ax^3+bx^2+cx+d \in \mathbf{F}_q[x]$ be a cubic polynomial whose roots (over the algebraic closure $\overline{\mathbf{F}}_q$) are distinct.

Let $\chi\colon \mathbf{F}_q^\times \to \{\pm 1\}$ be the quadratic character associated to that finite field, that is, the homomorphism $\chi\colon \mathbf{F}_q^\times \to \{\pm 1\}$ defined by

$$\chi(x) = \begin{cases} 1 & \text{if } x \in \mathbf{F}_q^\times \text{ is a square,} \\ -1 & \text{else.} \end{cases}$$

Extend χ to $\mathbf{F}_q$ by setting $\chi(0) = 0$. Using the Hasse–Weil bounds for an appropriate elliptic curve to prove that

$$\left| \sum_{x \in \mathbf{F}_q} \chi(f(x)) \right| \leq 2\sqrt{q}.$$

CHAPTER 8 Elliptic Curve Cryptography

8.1 Elliptic Curve Cryptosystems

8.1.1 Scalar multiplication of points

A basic arithmetic operation with elliptic curves is a scalar multiplication of a point. Let E be an elliptic curve over $\mathbb{F}_q$, let $P \in E(\mathbb{F}_q)$ be a point and let k be an integer. Computing kP then takes $\mathcal{O}(\log k(\log q)^3)$ operations. Indeed, to compute kP via the repeated doubling method, we have to perform $\log k$ point additions or doublings. Each point addition or doubling requires $\mathcal{O}((\log q)^3)$ operations in $\mathbf{F}_q$ (addition, multiplication, division or subtraction). **Dimitar** : **Fix this!**

8.1.2 The elliptic curve discrete logarithm problem

Let E be an elliptic curve over $\mathbf{F}_q$ and let $P \in E(\mathbf{F}_q)$ be a point. The elliptic curve discrete logarithm problem for E (with respect to base P) is the following problem:

Elliptic Curve DLP: given P and $Q = rP$, compute efficiently the integer r mod N where N is the order of P in $E(\mathbf{F}_q)$.

As we will see below, the security of many of the schemes depends on the difficulty of solving the above problem.

8.1.3 Analogue of Diffie–Hellman key exchange

Alice chooses a secret multiplier a and sends aP to Bob. Bob chooses a secret multiplier b and sends bP to Alice. Both Alice and Bob can then compute abP. In order for an adversary to be able to compute abP, the adversary has to solve the following mathematical problem:

Elliptic Curve Diffie–Hellman (ECDH) Problem: Given $P, aP, bP \in E(\mathbf{F}_q)$, compute abP.

Clearly, if one can solve the ECDL problem then one can solve the ECDH problem, but not necessarily conversely (a difficult open question).

8.1.4 Analogue of ElGamal encryption

Here, we choose an elliptic curve $E/\mathbf{F}_q$ and a point $P \in E(\mathbf{F}_q)$ of large prime order. Bob chooses a random integer b (the secret key) and publishes $Q = bP$. To send a message m to Bob, Alice first encodes m to a point $P_m \in \langle P \rangle$. To encrypt P_m, Alice picks a random integer r, computes (C_1, C_2) where $C_1 = P_m + rQ$ and $C_2 = rP$ and sends it to Bob. In order to decrypt a ciphertext (C_1, C_2), Bob computes $C_1 - bC_2$.

8.1.5 Elliptic curve digital signature algorithm (ECDSA)

The elliptic curve digital signature algorithm (ECDSA) is a classical signature algorithm whose security relies on the difficulty of the discrete logarithm problem. To describe the algorithm, we define the signing key pair as follows: we first fix an elliptic curve $E/\mathbf{F}_q$ and a point $P \in E(\mathbf{F}_q)$ of large order n. The group will be $G = \langle P \rangle$.

Secret key: Alice's secret key is a secret multiplier s_A.

Public key: Alice's public key is the point $Q_A = s_A P$.

In addition, we use a cryptographic hash function $H\colon \{0,1\}^* \to [1, n-1]$. To sign a message m, Alice performs the following steps:

1. Compute $z = H(m)$
2. Pick an invertible random k mod n and compute $(x_1, y_1) = kP$
3. Set $r := x_1 \bmod n$, $s := k^{-1}(z + rs_A) \bmod n$ and define $\sigma := (r, s)$.

In order for Bob to verify the signature $\sigma = (r, s)$ for the message m using only the public key of Alice, Bob performs the following steps:

1. Compute $z := H(m)$
2. Compute $u_1 = zs^{-1} \bmod n$ and $u_2 = rs^{-1} \bmod n$
3. $(x_1, y_1) := u_1 P + u_2 Q_A$
4. The signature is declared valid if $x_1 \equiv r_1 \bmod n$.

Showing correctness is easy - we simply write

$$(x_1, y_1) \quad = \quad u_1 P + u_2 Q = zs^{-1}P + rs^{-1}s_A P = s^{-1}(z + rs_A)P = kP.$$

8.1.6 Encoding plaintexts into points on elliptic curves

The first problem we will discuss is how to embed plaintexts into points on the elliptic curve so that the latter can be encrypted. Note that you never encrypt a "large document" using elliptic curve encryption schemes. What you do to encrypt a document is generate a symmetric key and encrypt with that symmetric key (much faster!) and then encrypt the symmetric key with the elliptic curve public-key encryption scheme. Suppose that your messages m satisfy $0 \leq m < M$ (i.e., are in the interval $[0, M)$.

One way to encode these messages as points on an elliptic curve is the following: pick an integer κ (to be determined later) and suppose that $q > M\kappa$ where q is the size of the finite field over which E is defined. We fix an identification of the set of integers $\{0, \ldots, M-1\}$ and a subset of elements of $\mathbf{F}_q$. Given a message m, consider the integers $m\kappa + j$ for $0 \leq j < \kappa$. Starting with $j = 0$, compute $x = m\kappa + j$ and test whether $f(x) = x^3 + ax + b$ is a square. If it is, then we output (x, y).

Now, given a point $(x, y) \in E(\mathbb{F}_q)$, we determine m by simply taking $m = \lfloor x/\kappa \rfloor$ (here, by abuse of notation, x denotes the integer corresponding to the x-coordinate of the point (an element of $\mathbb{F}_q$). The probability of failing to encode m with a point $(x, y) \in E(\mathbf{F}_q)$ is given by the probability of $f(x)$ being non-square for $x = m\kappa + j$ for all $j = 0, 1, \ldots, \kappa - 1$. Since $f(x)$ is a square for about 50% of the values of x, this algorithm will fail to produce a square with probability $2^{-\kappa}$. If we choose κ sufficiently large, then we will have a very high probability to produce a point $P_m \in E(\mathbf{F}_q)$.

8.2 Elliptic Curve Factorisation Algorithm

8.2.1 Point addition and point doubling formulas

We recall the basic addition and doubling formulas. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on the elliptic curve $E\colon y^2 = x^3 + ax + b$. The coordinates (x_3, y_3) of the point $P_1 + P_2$ are then given by

$$\begin{cases} x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \\ y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3). \end{cases} \tag{8.1}$$

If $P = (x, y)$ then the coordinates (x', y') of the point $2P$ are given by

$$\begin{cases} x' = \left(\frac{3x^2 + a}{2y}\right)^2 - 2x, \\ y' = -y + \left(\frac{3x^2 + a}{2y}\right)(x - x'). \end{cases} \tag{8.2}$$

The formulas themselves will not be so important. What will be important are the denominators appearing in the coordinates of the resulting point.

8.2.2 Arithmetic with points over $\mathbb{Z}/n\mathbb{Z}$

Proposition 8.2.1. *Let n be an odd integer and let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve with $a, b \in \mathbf{Z}$ and $\gcd(4a^3 + 27b^2, n) = 1$. Let P_1, P_2 be two points on E whose coordinates have denominators prime to n. Then $P_1 + P_2$ has coordinates with denominators coprime to n if and only if there is no prime $p \mid n$ such that the points $P_1 \bmod p$ and $P_2 \bmod p$ on the elliptic curve $E \bmod p$ (obtained from E by reducing the coefficients mod p) add up to the point at infinity for $E \bmod p$.*

Proof. Suppose first that P_1, P_2 and $P_1 + P_2$ all have coordinates with denominators prime to n. Let $p \mid n$ be any prime divisor. We have to show that $(P_1 \bmod p) +$

$(P_2 \bmod p) \neq 0 \bmod P$. The statement is clear if $x_1 \not\equiv x_2 \bmod p$. Suppose that $x_1 \equiv x_2 \bmod p$. We will consider two cases:

Case 1: $P_1 = P_2$. In this case, we are using the doubling formulas (8.2) and we will be done if we check that $p \nmid 2y_1$. Suppose that $p \mid 2y_1$. Then the enumerator $3x_1^2 + a$ must necessarily be divisible by p (since the coordinates of $P_1 + P_2$ have denominators coprime to p). This will mean that $p \mid x_1^3 + ax_1 + b$ as well and hence, x_1 is a repeated root of $x^3 + ax + b \bmod p$. This contradicts the fact that the discriminant of the curve is not divisible by p.

Case 2: $P_1 \neq P_2$. In this case, we use the addition formulas (8.1). Writing $x_2 = x_1 + p^r x$ for some $x \not\equiv 0 \bmod p$, we see that $y_2 = y_1 + p^r y$ for some y. We write

$$\begin{aligned} y_2^2 &= x_2^3 + ax_2 + b = (x_1 + p^r x)^3 + a(x_1 + p^r x) + b = x_1^3 + ax_1 + b + p^r x(3x_1^2 + a) = \\ &= y_1^2 + p^r x(3x_1^2 + a) \bmod p^{r+1}. \end{aligned}$$

Now, the points $P_1 \bmod p$ and $P_2 \bmod p$ are equal. If their sum is $0 \bmod p$ then $y_1 \equiv y_2 \equiv 0 \bmod p$ (again, (8.2)). This would imply that $p^{r+1} \mid y_2^2 - y_1^2$ and hence, $3x_1^2 + a \equiv 0 \bmod p$. This again implies that $x^3 + ax + b$ has multiple roots which is a contradiction.

Conversely, suppose that $P_1 \bmod p + P_2 \bmod p \neq 0 \bmod p$ for every $p \mid n$. We would like to show that the denominators of the coordinates of $P_1 + P_2$ are coprime to n. If $x(P_1) \not\equiv x(P_2) \bmod p$ then the denominators of the coordinates of $P_1 + P_2$ are clearly not divisible by p by (8.1). If $x(P_1) \equiv x(P_2) \bmod p)$ there are two cases: 1) $P_1 = P_2$ (in this case, we use doubling formulas); 2) $P_1 \neq P_2$ (in this case, we use addition formulas). In both cases, one proves that the denominators of the rational coordinates of $P_1 + P_2$ are not divisible by p. □

8.2.3 Lenstra's ECM method

Let (E, P) be a pair of an elliptic curve with integer coefficients and a point on E. Suppose that B and C are two bounds (to be determined later) and define

$$k = \prod_{\ell \leq B} \ell^{\alpha_\ell},$$

where α_ℓ is the largest exponent such that $\ell^{\alpha_\ell} \leq C$. The idea of the method is to try to compute kP modulo n. This computation will only be useful if in the process we encounter a denominator in the addition or doubling formulas ($x_2 - x_1$ or $2y_1$) that is not coprime to n. According to Proposition 8.2.1, this will happen if there some partial sum $k_1 P$ occurring on the way of the computation for which $k_1(P \bmod p) = 0$, i.e., the point $P \bmod p$ has order dividing k_1. This is very similar to having a prime factor p in the Pollard's $(p-1)$-method for which $p - 1$ is smooth.

The hope is that the denominator occurring will have a non-trivial common divisor with n. If this divisor were n, that would mean that for every $p \mid n$, $k_1(P \bmod p) = 0$. This is unlikely, especially if we have two large prime divisors of n. Hence, a failure in the addition or doubling formulas in the intermediate computations of kP would mean that one may be computing a non-trivial factor of n.

Just as in Pollard's $(p-1)$-method, the larger B is, the bigger the probability is that $kP \bmod 0 \bmod p$ for some $p \mid n$. The larger B is, the more time it takes to compute kP. The role of the parameter C is to bound the primes $p \mid n$ for which it is likely that $kP \bmod p = 0 \bmod p$. Indeed, suppose that p is a primes such that $p + 1 + 2\sqrt{p} < C$ and that $\#E(\mathbf{F}_p)$ is B-smooth. It then follows from the Hasse bound that k is a multiple of $\#E(\mathbf{F}_p)$ and hence, that $kP \bmod p = 0 \bmod p$ (just as in Pollard's $(p-1)$ method the exponent is a multiple of $p-1$).

Suppose now that p is a prime for which $\#E(\mathbf{F}_p) \leq p+1+2\sqrt{p} < C$ and $\#E(\mathbf{F}_p)$ is B-smooth. Then, $m \mid k$ and hence, $kP = O \bmod p$. To compute $kP \bmod n$ in practice, we do the following: we first compute

$$2P, 2^2P, \ldots, 2^{\alpha_2}P.$$

We then compute

$$3(2^{\alpha_2}P), 3^2(2^{\alpha_2}P), \ldots, 3^{\alpha_3}(2^{\alpha_2}P),$$

then

$$5(2^{\alpha_2}3^{\alpha_3}P), 5^2(2^{\alpha_2}3^{\alpha_3}P), \ldots, 5^{\alpha_5}(2^{\alpha_2}3^{\alpha_3}P).$$

and so on until we go over the primes $p \leq B$. If at some point in the computation we obtain a denominator that is not prime to n, we either get a non-trivial factor of n (if the gcd is not n) or we restart the procedure by choosing another pair (E, P). Finally, if all the computations are successful, we choose a different pair (E, P) and start again.

8.3 Elliptic Curve Primality Testing

8.3.1 Pocklington's test

Just as Lenstra's ECM method uses elliptic curves to generalizes Pollard's $(p-1)$-method (based on the group $(\mathbf{Z}/p\mathbf{Z})^\times$, there is an elliptic curve primality testing method generalizing a classical primality testing method based on the group $(\mathbf{Z}/n\mathbf{Z})^\times$ known as Pocklington's test. We start by describing the latter:

Proposition 8.3.1 ((Pocklington's test)). *Let $n > 1$ be an integer such that $n-1$ has a prime divisor $q > \sqrt{n} - 1$. Suppose that there exists an element $a \in (\mathbf{Z}/n\mathbf{Z})^\times$ such that*

1. $a^{n-1} \equiv 1 \bmod n$,
2. $\gcd\left(a^{(n-1)/q} - 1, n\right) = 1$.

Then n is prime.

Proof. Suppose that n is not prime. Then there exists a prime divisor $p \mid n$ with $p \leq \sqrt{n}$. In particular, $q > p-1$ and hence, $\gcd(q, p-1) = 1$. We can thus find an integer s such that $sq \equiv p-1 \bmod p$. It then follows that

$$a^{\frac{n-1}{q}} \equiv a^{(p-1)\frac{(n-1)}{q}} \equiv a^{\frac{sq(n-1)}{q}} \equiv a^{(p-1)(n-1)} \equiv 1 \bmod p,$$

and hence, $\gcd\left(a^{(n-1)/q}-1, n\right) > 1$, a contradiction with the assumption that n is composite. Hence, n is prime. □

8.3.2 The algorithm of Goldwasser–Killian–Atkin

If we are to generalize Pocklington's test to elliptic curves, it is important to first determine the analogue of the number $n-1$. Recall that when n is prime, this is the order of the group $(\mathbf{Z}/n\mathbf{Z})^\times$. It is thus natural to search for a similar integer m that, when n is prime, is the number of points on an elliptic curve over $\mathbf{Z}/n\mathbf{Z}$. Suppose that you have any point-counting method $\mathcal{A}$ (we just mentioned the algorithm of Schoof–Elkies–Atkin that is used in practice although we did not go over the algorithm itself).

To find such a number, we will be taking elliptic curves over $\mathbf{Z}/n\mathbf{Z}$ and try to run $\mathcal{A}$ on E. If n is prime, $\mathcal{A}$ is guaranteed to compute the number $m = \#E(\mathbf{F}_n)$. If n is not prime, $\mathcal{A}$ (run on the "set" $E(\mathbf{Z}/n\mathbf{Z})$) either returns some number m or it encounters an undefined expression (in either the addition or doubling formulas). In the latter case (just as in Lenstra's factoring method), this means that we have encountered a denominator having a non-trivial common factor with n.

To make things more precise, consider the following proposition that serves as a basis for the primality test using elliptic curves due to Goldwasser, Killian and Atkin:

Proposition 8.3.2. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbf{Z}/n\mathbf{Z}$ and let m be an integer. Suppose that there is a prime q dividing m that is greater than $(n^{1/4}+1)^2$. If there exists a point P on E such that: i) $mP = O$; ii) $(m/q)P$ is defined and not equal to O then n is prime.*

Proof. If n is composite, let $p \le \sqrt{n}$ be a prime divisor. Consider the elliptic curve E' that is the curve E taken modulo p and let $m' = \#E'(\mathbf{F}_p)$. By the Hasse–Weil bounds,

$$m' \le p + 1 + 2\sqrt{p} = (p^{1/2}+1)^2 \le (n^{1/4}+1)^2.$$

This means that $\gcd(q, m') = 1$ and hence, there exists an integer s, such that $sq \equiv 1 \bmod m'$. It then follows that

$$(m/q)P' = sq(m/q)P' = mP' = 0 \in E(\mathbf{F}_p),$$

which is a contradiction with the assumption that n is composite. □

The idea of the algorithm is then to choose three random numbers (a, x, y) and compute

$$b = y^2 - x^3 - ax \bmod n.$$

The point $P = (x, y)$ is then a point on the curve $E\colon y^2 = x^3 + ax + b \bmod n$. We then try the point-counting algorithm $\mathcal{A}$ on E. If it does abort, that means that n is not a prime number (we have found a non-trivial factor of n). Suppose that the algorithm outputs m. If one can write $m = kq$ for a small k and probable prime q (probable in the sense of any probabilistic primality test) then we keep the curve and compute mP and kP. If $mP \neq 0$ then n cannot be prime (since if it were, then mP would have been 0 as the order of the group $E(\mathbb{F}_n)$ would have been m). If $kP = 0$ then we discard E and start over.

Suppose now that $mP = 0$ but $kP \neq 0$. Then, if q were indeed prime, then Proposition 8.3.2 would tell us that n is prime. To test the primality of q, we apply this same algorithm recursively.

CHAPTER 9

Lattice-basis Reduction Algorithms

9.1 History of Lattices

Lattices are interesting mathematical structures extensively studied in mathematics, physics, chemistry, computer science and many other subjects. Back in the 19th century, Gauss and Lagrange studied lattices in order to give proofs of the quadratic reciprocity and the four-square theorem. It was Minkowski who developed *geometry of numbers* in the 19th century via the study of lattices.

Computational aspects of lattices did not come until the early 1980's with the famous work of Lenstra, Lenstra and Lovász (LLL) on lattice basis reduction algorithms [LLL82]. The LLL algorithm was originally used for factoring polynomials over $\mathbf{Q}$ and for breaking several cryptosystems.

Later on (mid-1990's), Ajtai [Ajt96] proved an unexpected worst-case to average-case reduction theorem for lattice problems, a statement that allowed for constructing one-way functions based on worst-case hardness conjectures. In subsequent joint work with Dwork, he constructed the first lattice-based cryptosystem, the Ajtai–Dwork cryptosystem [AD97]. As the scheme was not secure for any realistic keys, it was primarily of theoretical interest.

Subsequently, Goldreich, Goldwasser and Halevi [GGH97] proposed a public-key encryption scheme based on the closest-vector problem (CVP). Around that time, the `NTRUEncrypt` scheme was proposed [HPS98].

9.2 Lattices and Lattice-based Cryptography

9.2.1 Lattices in R^n

By a lattice of $\mathbf{R}^n$ we mean a discrete subgroup of $\mathbf{R}^n$. Here, discrete means that for each point x in the lattice, there is a neighborhood of x in $\mathbf{R}^n$ which intersects the lattice only at x. One can show that any discrete subgroup of $\mathbf{R}^n$ is free of rank at most n. By a (full-rank) lattice $\mathcal{L}$ of $\mathbf{R}^n$, we mean discrete subgroup of $\mathbf{R}^n$ of rank exactly n. Unless explicitly specified, for the rest of the chapter, we will use the term *lattice* to mean a *full-rank lattice.*

Given a basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of $\mathbf{R}^n$, we define the lattice

$$\mathcal{L}(\mathcal{B}) := \mathbf{Z}\,\mathbf{b}_1 + \mathbf{Z}\,\mathbf{b}_2 + \cdots + \mathbf{Z}\,\mathbf{b}_n\,.$$

Moreover, associated to $\mathcal{B}$ is the n-by-n basis matrix B whose columns are the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ written in terms of the standard basis.

If $\mathcal{B}$ and $\mathcal{B}'$ are two bases of the same lattice $\mathcal{L} \subset \mathbf{R}^n$ then one sees immediately that the change of basis matrix U satisfies $\det(U) = \pm 1$ (indeed, both U and its inverse have integer determinants and hence which forces $\det(U)$ to be ± 1).

9.2.2 The discriminant of a lattice

Given a basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbf{R}^n$, we define the discriminant $\operatorname{disc}(\mathcal{L}(\mathcal{B}))$ of the lattice $\mathcal{L}(\mathcal{B})$ as the volume of the *fundamental parallelepiped* $\mathcal{P}(\mathcal{B})$ associated to $\mathcal{B}$. The latter is defined as

$$\mathcal{P}(\mathcal{B}) := \{x_1\,\mathbf{b}_1 + \cdots + x_n\,\mathbf{b}_n : x_i \in [0,1),\ i = 1, \ldots, n\}.$$

One can prove that

$$\operatorname{disc}(\mathcal{L}(\mathcal{B})) := \det(B^t B)^{1/2} = |\det(B)|,$$

where B is the associated basis matrix.

9.2.3 Theory of short vectors in lattices and Minkowski's theorems

Given a lattice $\mathcal{L}$, we use $\lambda_1(\mathcal{L})$ to denote the length of a shortest vector of $\mathcal{L}$. Scaling $\mathcal{L}$ by a constant scales $\lambda_1(\mathcal{L})$ by the same constant. One can thus scale the lattice so that $\operatorname{disc}(\mathcal{L}) = 1$. A question studied in the 19th century was whether $\lambda_1(\mathcal{L})$ can be arbitrary large for such a lattice. It turns out that the answer was negative as the following theorem of Hermite shows:

Theorem 9.2.1 (Hermite'1870)**.** *For each n, there exists a constant C_n such that for any lattice $\mathcal{L}$ of dimension n,*

$$\lambda_1(\mathcal{L}) \leq C_n \cdot \operatorname{disc}(\mathcal{L})^{1/n}.$$

In fact, one defines the Hermite constant as follows:

Definition 9.2.2 (Hermite constant)**.** For each n, define

$$\gamma_n := \sup_{\mathcal{L} \subset \mathbf{R}^n} \left(\frac{\lambda_1(\mathcal{L})}{\operatorname{disc}(\mathcal{L})^{1/n}} \right)^2,$$

where the infimum is taken over all lattices $\mathcal{L} \subset \mathbf{R}^n$ of dimension n.

Hermite's theorem shows that this supremum is finite. Note the square in the definition of γ_n - it appears in the definition for historical reasons as most of the theory was developed in terms of positive-definite quadratic forms instead of lattices.

One of the first results of Minkowski's theory provides an upper bound for γ_n. We start by the following famous theorem of Minkowski:

Theorem 9.2.3 (Minkowski)**.** *Let $\mathcal{L} \subset \mathbf{R}^n$ be a lattice. Any convex and centrally symmetric body $S \subset \mathbf{R}^n$ that satisfies vol$(S) > 2^n \operatorname{disc}(\mathcal{L})$ has a non-zero lattice point.*

Proof. **Dimitar : Give a proof for completeness.** □

As a corollary, we obtain

Corollary 9.2.4 (Minkowski's first theorem)**.** *For every n, $\gamma_n < n$.*

Proof. **Dimitar : Give a proof for completeness.** □

Remark 12. Let $\mathcal{L} := \mathcal{L}(\mathcal{B})$ where $\mathcal{B}$ is the basis consisting of the columns of the matrix

$$B = \begin{pmatrix} 1 & 0 \\ \alpha & p \end{pmatrix},$$

where $\alpha \in \mathbf{Z}$ is such that $\alpha^2 \equiv -1 \bmod p$ - note that such an α exists since $p \equiv 1 \bmod 4$. It follows by Minkowski's first theorem (Corollary 9.2.4) that $\gamma_2 < 2$. Since $\operatorname{disc}(\mathcal{L}) = p$ then the square-norm of the shortest vector in $\mathcal{L}$ will be less than $2p$. But notice that any vector $\mathbf{v} = \begin{pmatrix} a \\ b \end{pmatrix} \in \mathcal{L}$ satisfies $p \mid (a^2 + b^2)$. Thus, Minkowski's theorem implies that there exists (a, b) such that $p = a^2 + b^2$. This proves that any prime number of the form $p = 4k + 1$ is a sum of two squares. We will see later how to efficiently find such a pair (a, b) that represent p using the LLL algorithm. See proposition 9.4.1.

The Hermite constants γ_n are only known for $n \leq 8$. Yet, for arbitrary n, one knows the following asymptotic lower and upper bounds (valid for large n):

$$\sqrt{\frac{n}{2\pi e}} \leq \gamma_n \leq \sqrt{\frac{n}{\pi e}}.$$

Shortest Vector Problem (SVP): Given a basis $\mathcal{B}$ of a lattice $\mathcal{L} \subset \mathbf{R}^n$, find a vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.

The SVP in its most basic form is an NP-hard problem under randomized reductions meaning that an algorithm for this problem would yield a randomized algorithm for any problem in NP. As such, one often talks about an approximate SVP where the goal is to find a vector $\mathbf{v}$ whose norm is bounded by $\gamma\lambda_1(\mathcal{L})$ for some approximation factor $\gamma > 0$. We call this problem the *γ-approximate SVP* problem (sometimes, one calls this the `GapSVP` problem).

Closest Vector Problem (CVP): Given a basis $\mathcal{B}$ of a lattice $\mathcal{L} \subset \mathbf{R}^n$ and a vector $\mathbf{w} \in \mathbf{R}^n$, find a vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{w} - \mathbf{v}\| = \operatorname{dist}(\mathbf{w}, \mathcal{L})$.

Similarly to the SVP problem, we talk about the γ-approximate CVP problem.

Finally, there is one more related problem to consider, the Shortest Independent Vectors Problem (`SIVP`).

9.2.4 Lattice-based cryptography

- Ajtai–Dwork cryptosystem
- Goldreich–Goldwasser–Halevi (GGH) cryptosystem
- NTRU cryptosystem

9.3 The LLL Algorithm

9.3.1 The case of $\dim(\mathcal{L}) = 2$

Given a lattice $\mathcal{L} = \mathbf{Z}\,\mathbf{b}_1 + \mathbf{Z}\,\mathbf{b}_2 \subset \mathbf{R}^2$, one way to reduce the basis is as follows: if $\|\,\mathbf{b}_1\,\| > \|\,\mathbf{b}_2\,\|$ then swap $\mathbf{b}_1$ and $\mathbf{b}_2$; else, we are tempted to find the component of the component of $\mathbf{b}_2$ orthogonal to $\mathbf{b}_1$, i.e., $\mathbf{b}_2' := \mathbf{b}_2 - \mu\,\mathbf{b}_1$ for $\mu = \dfrac{\langle \mathbf{b}_2, \mathbf{b}_1\rangle}{\langle \mathbf{b}_1, \mathbf{b}_1\rangle}$. The family $\{\mathbf{b}_1, \mathbf{b}_2'\}$ is orthogonal, yet, $\mathbf{b}_2'$ need not be in $\mathcal{L}$ since μ need not necessarily an integer. Instead, we try to update $\mathbf{b}_2 := \mathbf{b}_2 - \lfloor \mu \rceil\,\mathbf{b}_1$. If the updated $\mathbf{b}_2$ is still longer than $\mathbf{b}_1$, we terminate; else, we repeat the above process. Gauss proved that the above algorithm always terminates and also, the vector $\mathbf{b}_1$ obtained upon termination is a shortest non-zero vector.

For instance, if $\mathbf{b}_1 = (3,0)$ and $\mathbf{b}_2 = (4,1)$ then $\mu = 4/3$. Hence, we reduce to $\mathbf{b}_1 = (3,0)$ and $\mathbf{b}_2 = (4,1) - (3,0) = (1,1)$. But now, the vector $\mathbf{b}_1$ that we started might have been too large. At this point, we can swap the two vectors to obtain $\mathbf{b}_1 = (1,1)$ and $\mathbf{b}_2 = (3,0)$. We repeat the process computing $\mu = 3/2$, thus, reducing to $\mathbf{b}_1 = (1,1)$ and $\mathbf{b}_2 = (-2,1)$.

9.3.2 The notion of a reduced basis

Definition 9.3.1 (Gram–Schmidt orthogonalization)**.** Given linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \subset \mathbf{R}^n$, the *Gram–Schmidt orthogonalization* are the vectors

$$\widehat{\mathbf{b}}_i := \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \widehat{\mathbf{b}}_j,$$

where $\mu_{i,j} = \dfrac{\langle \mathbf{b}_i, \widehat{\mathbf{b}}_j\rangle}{\langle \widehat{\mathbf{b}}_j, \widehat{\mathbf{b}}_j\rangle}$.

Definition 9.3.2 (Reduced basis)**.** A basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbf{R}^n$ is called δ-LLL reduced if the following conditions are satisfied

1. (size reduced) $|\mu_{i,j}| \le 1/2$ for all $1 \le j < i \le n$.
2. (Lovász condition) $\delta \|\widehat{\mathbf{b}}_i\|^2 \le \|\mu_{i+1,i}\widehat{\mathbf{b}}_i + \widehat{\mathbf{b}}_{i+1}\|^2$ for all $1 \le i < n$.

The intuition behind the last definition is the following: if we write the basis $\mathcal{B}$ in terms of the Gram–Schmidt basis, the first property bounds the size of the off-diagonal entries. More precisely, the Gram–Schmidt basis satisfies:

$$\begin{pmatrix} \|\widehat{\mathbf{b}}_1\| & \leq \frac{1}{2}\|\widehat{\mathbf{b}}_1\| & \dots & \leq \frac{1}{2}\|\widehat{\mathbf{b}}_1\| \\ 0 & \|\widehat{\mathbf{b}}_2\| & \dots & \leq \frac{1}{2}\|\widehat{\mathbf{b}}_2\| \\ & & \ddots & \\ 0 & 0 & \dots & \|\widehat{\mathbf{b}}_n\| \end{pmatrix}$$

The second property tells us that $\widehat{\mathbf{b}}_{i+1}$ is not much shorter than $\widehat{\mathbf{b}}_i$. Indeed,

$$\delta\|\widehat{\mathbf{b}}_i\|^2 \leq \|\mu_{i+1,i}\widehat{\mathbf{b}}_i + \widehat{\mathbf{b}}_{i+1}\|^2 = \mu_{i+1,i}^2\|\widehat{\mathbf{b}}_i\|^2 + \|\widehat{\mathbf{b}}_{i+1}\|^2 \leq \frac{1}{4}\|\widehat{\mathbf{b}}_i\|^2 + \|\widehat{\mathbf{b}}_{i+1}\|^2,$$

hence,

$$\|\widehat{\mathbf{b}}_{i+1}\|^2 \geq \left(\delta - \frac{1}{4}\right)\|\widehat{\mathbf{b}}_i\|^2 \tag{9.1}$$

This allows us to show that the vector $\mathbf{b}_1$ is not too long:

Lemma 9.3.3. *One has* $\|\,\mathbf{b}_1\,\| \leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \lambda_1(\mathcal{L})$.

Proof. It follows from (9.1) that

$$\|\widehat{\mathbf{b}}_i\| \geq \left(\delta - \frac{1}{4}\right)^{\frac{i-1}{2}} \|\,\mathbf{b}_1\,\|.$$

Next, we show that

$$\lambda_1(\mathcal{L}) \geq \min\{\|\widehat{\mathbf{b}}_1\|, \dots, \|\widehat{\mathbf{b}}_n\|\}.$$

Indeed, let $\mathbf{x} \in \mathcal{L}$ be a (shortest) vector of length λ_1. One can write $\mathbf{x} = c_1\,\mathbf{b}_1 + \cdots + c_r\,\mathbf{b}_r$ for some $c_i \in \mathbf{Z}$ and $r \leq n$ such that $c_r \neq 0$. We now express the $\mathbf{b}_i$'s in terms of the Gram–Schmidt $\widehat{\mathbf{b}}_i$'s to obtain (for some coefficients $\nu_1, \dots, \nu_r \in \mathbf{R}$)

$$\|\mathbf{x}\|^2 = c_r^2\|\widehat{\mathbf{b}}_r\|^2 + \|\nu_1\widehat{\mathbf{b}}_1 + \cdots + \nu_{r-1}\widehat{\mathbf{b}}_{r-1}\|^2 \geq \|\widehat{\mathbf{b}}_r\|^2 \geq \min\{\|\widehat{\mathbf{b}}_1\|^2, \dots, \|\widehat{\mathbf{b}}_n\|^2\}.$$

Note that $\nu_r = c_r$ since $\widehat{\mathbf{b}}_r - \mathbf{b}_r$ lies in the span of $\mathbf{b}_1, ..., \mathbf{b}_{r-1}$. Hence,

$$\lambda_1(\mathcal{L}) \geq \left(\delta - \frac{1}{4}\right)^{\frac{n-1}{2}} \|\,\mathbf{b}_1\,\|,$$

and hence,

$$\|\,\mathbf{b}_1\,\| \leq \left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}} \lambda_1(\mathcal{L}),$$

which proves the lemma. □

9.3.3 The algorithm

The algorithm of Lenstra–Lenstra–Lovász (Algorithm 14) alternates two phases - a reduction phase and a swapping phase. We will show below that, if it terminates, then it produces a correct LLL-reduced basis. We also show that it always terminates via a potential argument that helps us also analyze its runtime.

Algorithm 14 LLL

```
Require: $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ - a lattice basis
Require: $\delta$ - parameter (typically, $\delta = 3/4$)
Ensure: A $\delta$-LLL reduced basis
    Reduction stage:
 1: $\{\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_n\} := \texttt{GramSchmidt}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$
 2: for $i = 2, \ldots, n$ do
 3:     for $j = i-1, \ldots, 1$ do
 4:         $\mu_{i,j} \leftarrow \langle \mathbf{b}_i, \widehat{\mathbf{b}}_j \rangle / \langle \widehat{\mathbf{b}}_j, \widehat{\mathbf{b}}_j \rangle$
 5:         $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lfloor \mu_{i,j} \rceil \, \mathbf{b}_j$
 6:     end for
 7: end for
    Swap stage:
 8: for $i = 1, \ldots, n-1$ do
 9:     if $\delta \|\widehat{\mathbf{b}}_i\|^2 > \|\mu_{i+1,i}\widehat{\mathbf{b}}_i + \widehat{\mathbf{b}}_{i+1}\|^2$ then
10:         $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$
11:         goto Step 1
12:     end if
13: end for
14: return $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$
```

9.3.4 Correctness

Lemma 9.3.4. *Assuming that the above procedure terminates, then the output is a δ-LLL reduced basis.*

Proof. Assuming that the procedure terminates, checking the Lovász condition (condition 2.) is straightforward: if it fails for some i then the algorithm swaps $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$ and keeps iterating. We thus only need to check condition 1. But this is an easy consequence of Steps 4. and 5. of the reduction stage: indeed,

$$\mu_{i,j}^{\text{new}} = \mu_{i,j}^{\text{old}} - \lfloor \mu_{i,j}^{\text{old}} \rceil \leq 1/2.$$

Since there are no more swaps before termination of the algorithm, the updated $\mu_{i,j}$'s satisfy condition 1. □

9.3.5 Termination

The argument for termination of the LLL algorithm is based on a potential function. Given a basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, define

$$\Phi(\mathcal{B}) := \Phi_1(\mathcal{B}) \cdot \cdots \cdot \Phi_n(\mathcal{B}),$$

where $\Phi_i(\mathcal{B}) := \det(\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_i)) = \|\widehat{\mathbf{b}}_1\| \cdot \cdots \cdot \|\widehat{\mathbf{b}}_i\|$.

Lemma 9.3.5. *(i) The reduction step does not change the potential function.*
(ii) The swap step lowers the potential function by a constant factor.

Proof. (i) This is obvious since $\Phi_i = \|\widehat{\mathbf{b}}_1\| \cdot \cdots \cdot \|\widehat{\mathbf{b}}_i\|$ and the reduction step does not change the Gram–Schmidt basis.
(ii) Note that changing the order of $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ changes only Φ_i (since changing the order of the vectors in a basis does not change the determinant).

$$\Phi_i(\mathcal{B}^{\text{old}}) = \det(\mathbf{b}_1, \ldots, \mathbf{b}_i), \qquad \text{and} \qquad \Phi_i(\mathcal{B}^{\text{new}}) = \det(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}).$$

Applying Gram–Schmidt orthogonalization to the (old) basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_i\}$ and the (new) basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}\}$ does not alter the first $i-1$ orthogonal vectors. For the second basis, the ith vector is precisely the component of $\mathbf{b}_{i+1}$ orthogonal to $\{\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_{i-1}\}$, i.e., $\widehat{\mathbf{b}}_{i+1} + \mu_{i+1,i}\widehat{\mathbf{b}}_i$. In addition, the swapping condition implies that

$$\|\widehat{\mathbf{b}}_{i+1}\|^2 + \mu_{i+1,i}^2\|\widehat{\mathbf{b}}_i\|^2 \leq \delta\| \mathbf{b}_i \|^2 \implies \|\widehat{\mathbf{b}}_{i+1} + \mu_{i+1,i}\widehat{\mathbf{b}}_i\| \leq \sqrt{\delta}.$$

Thus,

$$\frac{\Phi_i(\mathcal{B}^{\text{new}})}{\Phi_i(\mathcal{B}^{\text{old}})} = \frac{\|\widehat{\mathbf{b}}_{i+1} + \mu_{i+1,i}\widehat{\mathbf{b}}_i\|}{\|\widehat{\mathbf{b}}_i\|} \leq \sqrt{\delta}.$$

On the other hand, $\Phi(\mathcal{B}) \geq 1$ since we are working with integral lattices. □

9.3.6 Complexity analysis

To estimate the complexity, it suffices to understand the size of the initial value of $\Phi(\mathcal{B})$.

Lemma 9.3.6. *The initial value $\Phi(\mathcal{B})$ is bounded by 2^N where N is polynomial in the dimension n and the number of bits of the basis $\mathcal{B}$. In addition, at each intermediate step of the algorithm, $\Phi(\mathcal{B}) \geq 1$.*

Proof. The lemma follows from

$$\Phi(\mathcal{B}) \leq \max_i \| \mathbf{b}_i \|^{\mathcal{O}(n^2)},$$

which is a direct consequence of the definition of $\Phi(\mathcal{B})$ that. □

This lemma, together with Lemma 9.3.5 shows that the number of iterations in the LLL algorithm is at most polynomial in n and the number of bits of the original basis $\mathcal{B}$.

9.4 Applications

9.4.1 Basic number theoretic applications

The original idea behind lattice-basis reduction (going back to Gauss and Lagrange) was to prove quadratic reciprocity and the four-square theorem, respectively. Let us illustrate these types of applications by proving the following basic and well-known fact:

Proposition 9.4.1. *(i) Every prime of the form $p = 4k+1$ is a sum of two squares.*

(ii) There exists an efficient algorithm (polynomial in $\log p$) to find such a representation.

Proof. Part (i) was already proved in Remark 12. We did this by considering the lattice $\mathcal{L}$ generated by the columns of the the matrix

$$B = \begin{pmatrix} 1 & 0 \\ \alpha & p \end{pmatrix},$$

where $\alpha^2 \equiv -1 \bmod p$. A shortest vector $v = \begin{pmatrix} a \\ b \end{pmatrix}$ was then shown to satisfy $p \mid (a^2 + b^2)$. For part (ii), we need to find a shortest vector efficiently. We apply the LLL algorithm to the original basis $\mathcal{B}$ to get a LLL-reduced basis $\mathcal{B}' = \{\mathbf{b}_1, \mathbf{b}_2\}$ for $\mathcal{L}$. We know that $\mathbf{b}_1$ is not too long, i.e.,

$$\| \mathbf{b}_1 \| \leq \left(\delta - \frac{1}{4}\right)^{-1/2} \operatorname{disc}(B)^{1/2}.$$

For $\delta > 3/4$, we obtain $\| \mathbf{b}_1 \|^2 < 2p$ which means that LLL finds such a representation. □

9.4.2 Finding exact shortest vectors

The LLL can give us an algorithm running in time $2^{\mathcal{O}(n^2)}$ to compute an exact shortest vector in a lattice $\mathcal{L}$ of dimension n.

Corollary 9.4.2. *Let $\mathcal{L}$ be a lattice of dimension n in $\mathbf{R}^n$. Then the shortest vector problem for $\mathcal{L}$ can be solved in time $2^{\mathcal{O}(n^2)}$.*

Proof. Consider a shortest vector $\mathbf{v} = \sum_{i=1}^{n} c_i \, \mathbf{b}_i$ where $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is a LLL-reduced basis. Consider also the Gram–Schmidt orthogonalization $\{\widehat{\mathbf{b}}_1, \ldots, \widehat{\mathbf{b}}_n\}$. Let i be the smallest index such that $c_i \neq 0$.

We claim that $|c_i| \leq 2^{\mathcal{O}(n)}$. To see this, first note that the argument of Lemma 9.3.3 yields

$$\|\widehat{\mathbf{b}}_\mathbf{i}\| \geq \left(\delta - \frac{1}{4}\right)^{i-1} \|\widehat{\mathbf{b}}_1\| = \left(\delta - \frac{1}{4}\right)^{i-1} \| \mathbf{b}_1 \|.$$

Next, since $c_i = \widehat{c}_i$, where $\mathbf{v} = \sum_{i=1}^{n} \widehat{c}_i \widehat{\mathbf{b}}_i$, one can write

$$\| \mathbf{b}_1 \| \geq \|\mathbf{v}\| \leq c_i \|\widehat{\mathbf{b}}_\mathbf{i}\| \leq c_i \left(\delta - \frac{1}{4}\right)^{i-1} \| \mathbf{b}_1 \|.$$

Hence,

$$c_i \leq \left(\frac{4}{4\delta - 1}\right)^{\frac{i-1}{2}}.$$

This yields an algorithm for finding an exact short vector with complexity $2^{\mathcal{O}(n^2)}$: once we have LLL-reduced the basis, we try all possibilities for c_1 (including $c_1 = 0$) and for each one of these possibilities, recursively find the shortest vector in the translated lattice $c_1 \mathbf{b}_1 + \mathcal{L}(\mathcal{B}')$ where $\mathcal{B}' = \{\mathbf{c}_2, \ldots, \mathbf{b}_n\}$. This yields an algorithm with complexity $2^{\mathcal{O}(n^2)}$. □

9.4.3 Attack on RSA with low public exponent

9.4.3.1 Coppersmith's method for finding small roots.

Coppersmith proposed a method based on LLL for finding all small roots of a polynomial modulo N even if the factorization of N is unknown [Cop01] (see also [Bon99]). This leads to an attack on the RSA cryptosystem with small exponents (*Coppersmith's attack*) which we explain later.

Theorem 9.4.3 (Coppersmith's small root finding). *Suppose that N is a modulus and $f(X) \in (\mathbf{Z}/N\mathbf{Z})[X]$ be a monic polynomial of degree d. One can compute efficiently all $x \in \mathbf{Z}$, $|x| \leq N^{1/d}$ such that $f(X) \equiv 0 \bmod N$.*

Proof. Let $B = N^{1/d}$ and let $f(X) = a_d X^D + \cdots + a_1 X + a_0$. If

$$|a_i B^i| < \frac{N}{d+1}, \; i = 0, \ldots, d, \tag{9.2}$$

then it follows that every solution x, $|x| \leq B$ of $f(x) \equiv 0 \bmod N$ is an integer solution of $f(x) = 0$. Indeed, for any such x, we have

$$|f(x)| \leq \sum_{i=0}^{d} |a_i B^i| < N.$$

This allows us to compute the roots of f by using any standard root-finding technique over the reals (e.g., Newton's method).

The problem is that (9.2) need not hold for f. To solve this, we can consider multiples of f taken modulo N. More precisely, any linear combination of the polynomials $\{N, NX, NX^2, \ldots, Nx^{d-1}, f(x)\}$ have the same roots as f modulo N. Thus, one can hope to find a linear combination g that satisfies (9.2). This is equivalent to finding a short vector in the following lattice:

$$\mathcal{L} := \begin{pmatrix} N & & & & a_0 \\ & BN & & & Ba_1 \\ & & \ddots & & \vdots \\ & & & B^{d-1}N & B^{d-1}a_{d-1} \\ & & & & B^d \end{pmatrix}$$

To find a short vector, we will apply the LLL algorithm. First, we bound the first successive minima of the lattice. Since

$$\det(\mathcal{L}) = N \cdot BN \cdot \cdots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2},$$

we obtain that LLL yields a vector whose length is bounded by

$$\|v\|_2 \leq C\lambda_1(\mathcal{L}) \leq C\sqrt{d+1}\det(\mathcal{L})^{\frac{1}{d+1}} = C\sqrt{d+1}N^{\frac{d}{d+1}}B^{d/2},$$

where C depends only on d and not on N. In the second inequality, we have used Minkowski's theorem to bound the first successive minima in terms of the determinant of the lattice. We can now take B sufficiently small, namely $B \leq C'N^{\frac{2}{d(d+1)}}$ to obtain

$$\|v\|_2 \leq C(C')^{d/2}\sqrt{d+1}N^{\frac{d}{d+1}}N^{\frac{1}{d+1}} = C(C')^{d/2}\sqrt{d+1}N.$$

Now, if we choose $(C')^{d/2}C\sqrt{d+1} \leq \frac{1}{d+1}$, i.e.,

$$C' \leq \left(\frac{1}{C(d+1)^{3/2}}\right)^{\frac{2}{d}},$$

we obtain inequality (9.2). Note that this method does not recover the roots up to $N^{1/d}$ but only up to $cN^{2/(d(d+1))}$ for some constant c. We need to write a slightly different lattice problem to get to $N^{1/d}$. There are two major ways to improve the exponent:

1. Consider a larger class of polynomials, namely,

$$\mathcal{L}_2 := \{N, Nx, Nx^2, \ldots, Nx^{d-1}, f(x), f(x)x, \ldots, f(x)x^{d-1}\}.$$

   This yields a lattice of dimension $2d$ whose determinant is

$$\det(\mathcal{L}_2) = N^d B^{2d(2d-1)/2}.$$

   The enabling condition is then of the form

$$B \leq C'N^{1/(2d-1)},$$

   i.e., the exponent is improved from $2/d(d+1)$ to $1/(2d-1)$.

2. Considering high powers of f and N. **Dimitar** : **To be completed.**

□

9.4.3.2 Coppersmith's attack on RSA.

Early proposals for instantiating the RSA cryptosystems used small public exponents. The smallest possible value is $e = 3$ for obvious efficiency reasons.

In addition, it is well-known that deterministic encryption with the RSA method is not secure since an eavesdropper can recognize chosen ciphertexts. A typical example is the following - if an attacker knows that a ciphertext corresponds to an interesting plaintext message, the attacker might try to learn information any time the message is transmitted. By performing statistical analysis, an attacker might try to correlate a ciphertext of a plaintext message with other actions or information. More precisely, deterministic encryption with RSA can never be semantically secure.

To remedy this problem, one uses random paddings of plaintext messages. To encrypt a message M, one concatenates a random m-bit pad to the message M, i.e., encrypts a message $M' = 2^m M + r$ for a random $r \in \{0, 2^m - 1\}$.

Theorem 9.4.4. *Let the modulus N be an n-bit number. Let C_1 and C_2 be two ciphertexts corresponding to the same message M padded with two different m-bit pads r_1, r_2 and RSA-encrypted with public exponent $e = 3$. If $m \leq \lfloor n/e^2 \rfloor$ then one can efficiently recover M.*

Before we prove the theorem, we record an important attack known as the *related message attack*:

Lemma 9.4.5 (related message attack)**.** *Let $e = 3$ and let $M_1, M_2 \in (\mathbf{Z}/N\mathbf{Z})^\times$ be two plaintext messages related via $M_2 = \ell(M_1)$ for a known linear function $\ell(x) = ax + b$ with $a, b \neq 0$. If one has access to $C_1 = M_1^e \bmod N$ and $C_2 = M_2^e \bmod N$ together with the RSA public information, one can efficiently recover M_1 and M_2.*

Proof. Consider the polynomials

$$g_1(x) = x^e - C_1 \qquad \text{and} \qquad g_2(x) = \ell(x)^e - C_2 \in (\mathbf{Z}/N\mathbf{Z})[x].$$

The condition of the lemma implies that $x = M_1$ is a common root of the two polynomials. We can thus try to find this common root using the Euclidean algorithm. In order for that strategy to work, we only need to check that $\gcd(g_1(x), g_2(x)) = x - M_1$. Write $g_1(x) = (x - M_1)h_1(x)$, where $h_1(x)$ is a quadratic polynomial. Since the function $g_1\colon \mathbf{Z}/N\mathbf{Z} \to \mathbf{Z}/N\mathbf{Z}$ is bijective, g_1 has a unique root M_1, i.e., $h_1(X)$ is irreducible in $(\mathbf{Z}/N\mathbf{Z})[x]$. It follows that $\gcd(g_1, g_2)$ is either $x - M_1$ or $g_1(x)$ itself. Since $b \neq 0$, this greatest common divisor cannot be $g_1(x)$ and hence, it is $x - M_1$. Thus, the Euclidean algorithm will find the root. □

Proof of Theorem 9.4.4. Consider the two bivariate polynomials

$$g_1(x, y) = x^e - C_1 \qquad \text{and} \qquad g_2(x, y) = (x + y)^e - C_2.$$

Here, x is supposed to represent M_1 and y is supposed to represent $r_2 - r_1$ (the difference between the two pads). The idea will be to use Coppersmith's method for finding small roots modulo N (without knowing the factorization of N) in order to find $y = r_2 - r_1$. To do that, we take the resultant of the two bivariate polynomials with respect to x, i.e., compute the univariate polynomial

$$h(y) = \mathrm{res}_x(g_1, g_2).$$

Recall that the resultant of two polynomials $f_1(x), f_2(x)$ is defined as

$$\mathrm{res}(f_1, f_2) := \prod_{(\alpha,\beta)} (\alpha - \beta),$$

where the product is taken over all pairs (α, β) of a root of f_1 and a root of f_2.

To complete the attack, we claim that $\Delta = r_2 - r_1$ is a root of the resultant polynomial. Indeed, this is a consequence of the fact that $g_1(M, \Delta) = 0 = g_2(M, \Delta)$, i.e., the two polynomials $g_1(x, \Delta)$ and $g_2(x, \Delta)$ have a common root $x = M$. Now, Δ is a root of $h(y)$ and $|\Delta| < 2^m < N^{1/e^2}$, hence, we can run Coppersmith's method (Theorem 9.4.3) to find a list of candidates Δ of polynomial size. For each such candidate, note that $M_2 = \ell(M_1)$ where $\ell(x) = x - \Delta$ is a linear function. Hence, the related message attack of Lemma 9.4.5 yields M. □

9.4.4 Approximate GCD problem.

The approximate GCD problem is the following problem: given a set of large integers $x_0, x_1, \ldots, x_t$ randomly chosen close to multiples of a large integer p, find p. We can parametrize it more formally as follows:

(ρ, η, γ)-approximate GCD problem. Given t samples from the distribution $\mathcal{D}_{\gamma,\rho}(p)$ for a η-bit number p, find p. Here, $\mathcal{D}_{\gamma,\rho}(p)$ is the distribution that chooses a uniformly random $q \leftarrow_R \mathbf{Z} \cap [0, 2^\gamma/p)$, $r \leftarrow_R \mathbf{Z} \cap (-2^\rho, 2^\rho)$ and outputs the number $pq + r$.

1. Simultaneous Diophantine Approximation (SDA)

2. Lagarias' algorithm based on LLL

For each $i = 0, \ldots, t$, let $x_i = pq_i + r_i$. Consider the lattice $\mathcal{L}$ given by the columns of the following $(t + 1) \times (t + 1)$ matrix:

$$M = \begin{pmatrix} 2^\rho & & & & \\ x_1 & -x_0 & & & \\ x_2 & & -x_0 & & \\ & & & \cdots & \\ x_t & & & & -x_0 \end{pmatrix}$$

The high-level idea is that the vector

$$v = M \cdot \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_t \end{pmatrix} = \begin{pmatrix} 2^\rho q_0 \\ x_0 q_0 \left(\frac{x_1}{x_0} - \frac{q_1}{q_0}\right) \\ \vdots \\ x_0 q_0 \left(\frac{x_t}{x_0} - \frac{q_t}{q_0}\right) \end{pmatrix}$$

will be a short vector in $\mathcal{L}$ for a certain range of parameters (ρ, η, γ). Indeed, we estimate

$$|2^\rho q_0| \leq 2^{\rho+\gamma-\eta}, \qquad \text{and} \qquad \left| x_0 q_0 \left(\frac{x_1}{x_0} - \frac{q_1}{q_0}\right) \right| \leq 2^{\rho+\gamma-\eta},$$

and hence, $\|v\| \leq 2^{\rho+\gamma-\eta}\sqrt{t+1}$.

Dimitar : Discuss range of parameters.

9.5 The NTRU Cryptosystem

The NTRU cryptosystem was originally developed by Hoffstein and further developed by Hoffstein, Pipher and Silverman [HPS98]. The basic algebraic operations of NTRU are:

1. Multiplication of polynomials in the ring $R := \mathbf{Z}[X]/\langle X^N - 1\rangle$ for a suitable N,
2. Convolution product in $\mathbf{Z}^N$ denoted by $\star$.

9.5.1 Key generation

There are three public parameters:

- N - degree of the polynomials,
- p - small modulus,
- q - large modulus.

The private key is generated as follows: generate random polynomials $\mathbf{F}, \mathbf{G} \in \{-1,0,1\}^N$ and define $\mathbf{f} = 1 + p\mathbf{F}$ and $\mathbf{g} = p\mathbf{G}$.

The public key is defined as $\mathbf{h} = \mathbf{f}^{-1} \star \mathbf{g}$

9.5.2 Encryption

To encrypt a message $\mathbf{m} \in \{-1,0,1\}^N$, we choose a random $\mathbf{r} \in \{-1,0,1\}^N$ and define the encryption of $\mathbf{m}$ as

$$\mathbf{c} = \mathbf{m} + \mathbf{r} \star \mathbf{h}.$$

9.5.3 Decryption

To decrypt a ciphertext $\mathbf{e}$, compute

$$\mathbf{a} = \mathbf{f} \star \mathbf{c} \bmod q,$$

and take $\mathbf{m} := \texttt{centermod}(\mathbf{a}) \bmod p$, where `centermod` is the unique lift from $(\mathbf{Z}/q\mathbf{Z})^N$ to $\mathbf{Z}^N$ that sends each coordinate $a \bmod q$ to the unique integer $\widetilde{a}$ in $(-q/2, q/2)$.

9.5.4 Correctness

To check the correctness, write

$$\begin{aligned} \mathbf{a} &= \mathbf{f} \star \mathbf{c} \bmod q = \mathbf{f} \star (\mathbf{m} + \mathbf{r} \star \mathbf{h}) \bmod q = \\ &= \mathbf{f} \star \mathbf{m} + \mathbf{f} \star \mathbf{r} \star \mathbf{f}^{-1} \star \mathbf{g} \bmod q = \mathbf{f} \star \mathbf{m} + \mathbf{r} \star \mathbf{g} \bmod q. \end{aligned}$$

But the coordinates of the vectors $\mathbf{f} \star \mathbf{m} + \mathbf{r} \star \mathbf{g}$ are small, hence,

$$\texttt{centermod}(a) = \mathbf{f} \star \mathbf{m} + \mathbf{r} \star \mathbf{g} \in \mathbf{Z}^N,$$

and hence,

$$\texttt{centermod}(a) \bmod p = (1 + p\mathbf{F}) \star \mathbf{m} + p\mathbf{r} \star \mathbf{G} \bmod p = \mathbf{m},$$

thus, proving the correctness.

9.5.5 Analysis

To analyze the security of `NTRUEncrypt`, we consider the so-called *convolution modular lattice*, that is, the lattice $\mathcal{L}_\mathbf{h}$ that is the $\mathbf{Z}$-span of the columns of the following $2N$-by-$2N$ matrix:

$$\mathcal{L}_\mathbf{h} := \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & & \ddots & & & & & \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ h_0 & h_{N-1} & \dots & h_1 & q & 0 & \dots & 0 \\ h_1 & h_0 & \dots & h_2 & 0 & q & & 0 \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \\ h_{N-1} & h_{N-2} & \dots & h_0 & 0 & 0 & \dots & q \end{pmatrix}$$

Another way of describing it is as as

$$\mathcal{L}_\mathbf{h} = \{(\mathbf{a}, \mathbf{b}) \colon \mathbf{b} \equiv \mathbf{a} \star \mathbf{h} \bmod q\}.$$

Note that the `NTRUEncrypt` private key $(\mathbf{f}, \mathbf{g})$ is a short vector in this lattice. The LLL algorithm can thus help us find a short vector in the lattice.

9.6 Block Korkine–Zolotarev (BKZ) Algorithm

The Korkine–Zolotarev (or Hermite–Korkine–Zolotarev) algorithm is another lattice basis reduction algorithm. Unlike the LLL algorithm, it runs in exponential time, yet, it yields a basis with a much lower orthogonality defect (at most $\mathcal{O}(n^n)$ as opposed to the $\mathcal{O}(2^{n^2})$ in LLL). To define the notion of a Korkine–Zolotarev reduced basis we use the same notation as before for the Gram–Schmidt matrix and additionally, write $\pi_i \colon \mathbf{R}^n \to \mathbf{R}^{n-i+1}$ for the projection map onto the span of $\{\mathbf{b}_i, \dots, \mathbf{b}_n\}$. We also use $\mu_{i,j} := \dfrac{\langle \mathbf{b}_i, \widehat{\mathbf{b}}_j \rangle}{\langle \widehat{\mathbf{b}}_j, \widehat{\mathbf{b}}_j \rangle}$.

Definition 9.6.1 (Korkine–Zolotarev reduced basis)**.** A basis $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ is called *KZ-reduced* if the following two conditions are satisfied

1. $\widehat{\mathbf{b}}_i$ is the shortest vector in $\pi_i(\mathcal{L}(\mathcal{B}))$ for every $i = 1, \dots, n$,

2. For all $j < i$, $|\mu_{i,j}| < 1/2$.

The notion of a KZ-reduced basis was proposed as a strengthening of Hermite reduction. The first algorithm to construct a KZ-reduced basis was proposed by Kannan [Kan83].

CHAPTER 10 Fully Homomorphic Encryption

10.1 Homomorphic encryption schemes - basic notions

10.2 Additively homomorphic encryption schemes

We start with some simpler schemes that are homomorphic with respect to addition and not multiplication, yet, they still have powerful practical applications.

10.2.1 Paillier's scheme

KeyGen: An RSA modulus $N = pq$, $\lambda = \operatorname{lcm}(p-1, q-1)$ (secret key).

Encrypt: To encrypt a message $0 \leq m < N$, select a random r, $0 < r < N$ such that $(r, N) = 1$ and compute

$$C = \mathtt{Enc}_{\mathtt{PK}}(m, r) := g^m \cdot r^N \bmod N^2.$$

Decrypt: To decrypt a ciphertext C, we compute

$$L(C^\lambda \bmod N^2) \cdot \mu \bmod N,$$

where $L(x)$ denotes the largest integer value $v \geq 0$ such that $x - 1 \geq vN$ and $\mu := (L(g^\lambda \bmod N^2))^{-1} \bmod N$.

Homomorphic addition: Given two plaintexts m_1, m_2 with ciphertexts $C_1 = \mathtt{Enc}$ and C_2, respectively, one can define the homomorphic addition as

$$C_1 \oplus C_2 = C_1 \cdot C_2 \bmod N^2 = g^{m_1+m_2}(r_1 r_2)^N \bmod N^2.$$

The decryption is then

$$\mathtt{Dec}_{\mathtt{SK}}(C_1 \oplus C_2) = m_1 + m_2 \bmod N.$$

The security of the scheme relies on the *decisional composite residuosity assumption.* **Dimitar : State the assumption.**

Decisional Composite Residuosity Assumption (DCRA): Given a composite number N and an integer x, it is hard to decide whether x is an N-residue modulo N^2, i.e., whether there exists a number y such that

$$x \equiv y^N \bmod N^2.$$

10.2.2 The scheme of Damgaard–Jurik

The scheme of Damgaard–Jurik [DJ] works modulo n^{s+1} for an RSA modulus n and a positive integer s. It uses that the group $(\mathbf{Z}/n^{s+1}\mathbf{Z})^\times \cong G \times H$ for a cyclic group G of order n^s and $H \cong (\mathbf{Z}/N\mathbf{Z})^\times$. The case $s = 1$ recovers exactly Paillier's scheme.

Yet, the scheme provides some advantage to the scheme of Paillier: 1) the choice of g; 2) the block size.

KeyGen: An RSA modulus $n = pq$, $\lambda = \mathrm{lcm}(p-1, q-1)$. In this scheme, we take $g = 1 + N \bmod N^{s+1}$

Encrypt: Given a message m, we generate a random nonce r and define a ciphertext

$$C(m, r) := (1+N)^m \cdot r^{N^s} \bmod N^{s+1}.$$

Decrypt:
Dimitar : Complete the section!

10.3 Learning with errors (LWE) problem

The learning-with-errors problem (LWE) is an important problem (conjectured to be hard by Regev [Reg09]) that has significant implications in public-key cryptography, including the constructions of almost all of the practical fully-homomorphic encryption schemes. We start with a more in-depth discussion of this problem,

10.3.1 Learning parity with noise (LPN) problem

- Parity learning problem (easy with Gaussian elimination),
- Learning parity with noise (LPN) - significantly harder,
- The subexponential algorithm of Blum–Kalai–Wasserman [BKW00].

For a given integer n, one is interested in finding $\mathbf{s} \in (\mathbf{Z}/2\mathbf{Z})^n$ given a set of equations

$$\begin{aligned}\langle \mathbf{s}, \mathbf{a}_1 \rangle &= b_1 \\ &\dots \\ \langle \mathbf{s}, \mathbf{a}_m \rangle &= b_m\end{aligned}$$

where $\mathbf{a}_i$ are chosen uniformly at random from the uniform distribution on $(\mathbf{Z}/2\mathbf{Z})^n$.

A significantly harder problem is when each equation holds with probability with probability $1-\varepsilon$ for some $\varepsilon > 0$ (the Learning Parity with Noise (LPN) problem). In this case, a straightforward application of Gaussian elimination would give us uncertainty as the following simple probability lemma shows:

Lemma 10.3.1. *If the samples $(\mathbf{a}_1, b_1), \dots, (\mathbf{a}_r, b_r)$ are corrupted by noise $\varepsilon < \frac{1}{2}$ then the probability that $\langle \mathbf{s}, \mathbf{a}_1 \rangle + \cdots + \langle \mathbf{s}, \mathbf{a}_r \rangle$ is equal to $b_1 + \cdots + b_r$ is equal to $\frac{1}{2} + \frac{1}{2}(1-2\varepsilon)^r$.*

Proof. Follows by straightforward induction on r. □

Assuming that we want to guess the first bit of $\mathbf{s}$, the above lemma shows that if we naïvely perform Gaussian elimination, the probability of guessing it correctly is $\frac{1}{2} + 2^{-\Theta(n)}$, i.e., we only get a negligible advantage over a random guess. To obtain a non-negligible advantage over a random guess (i.e., guessing the bit with probability $\frac{1}{2} + \texttt{polylog}(n)^{-1}$), one should repeat the above procedure $\Theta(2^n)$ times. The naïve algorithm is thus exponential in the dimension n.

The first proposed subexponential algorithm was due to Blum, Kalai and Wasserman [BKW00] (known as the BKW algorithm). The idea of the algorithm is simple: by drawing many more samples than the minimum needed for Gaussian elimination, one can hope to write the basis vector $(1, 0, \dots, 0)$ as the sum of $O(\log n)$ instead of n sample vectors $\mathbf{a}_i$, thus, keeping the advantage polylogarithmic in n.

This last idea is applicable to other important hardness problems - it was used by Kumar and Sivakumar [KS01] as well as by Ajtai et al. [AKS01] to find the first algorithm for solving the shortest vector problem in time $O(2^n)$.

Finally, it is worth mentioning that there has been progress on quantum algorithms for solving LPN:

- The quantum attack of Esser, Kübler and May [EKM17] revisiting the attack of Blum, Kalai and Wasserman (BKW),
- Subsequent improvements of [Jia20] and most recently [TV22].

10.3.2 LWE problem

The LWE problem generalizes the LPN problem to any modulus q [Reg09]. The original motivation of Regev was to understand the difficulty of solving the LPN problem. The main result of Regev [Reg09, Thm.1.1] was that if one can solve LWE for a certain noise distribution then there exists an efficient quantum algorithm to approximate the decisional version of the approximate SVP and the SIVP problem. Of course, this qualitative result needs to be made quantitatively precise.

There are two major versions - a continuous one and a discrete one which we present below. In addition to the search problem, there is also a decisional problem.

10.3.2.1 Continuous version.

Let $\mathbf{s} \in (\mathbf{Z}/q\mathbf{Z})^n$ be a vector. Let χ be an arbitrary distribution over the torus $\mathbf{T} = \mathbf{R}/\mathbf{Z}$. Define a distribution $\mathcal{A}_{\mathbf{s},\chi}$ over $(\mathbf{Z}/q\mathbf{Z})^n \times \mathbf{T}$ as follows:

1. Choose a uniformly random $\mathbf{a} \in (\mathbf{Z}/q\mathbf{Z})^n$,
2. Choose a random sample $e \leftarrow \chi$ from the distribution χ.
3. Compute $b = \phi(\langle \mathbf{a}, \mathbf{s} \rangle) + e \in \mathbf{T}$. Here, $\langle \mathbf{a}, \mathbf{s} \rangle / q$. Here, $\phi \colon \mathbf{Z}/q\mathbf{Z} \to q^{-1}\mathbf{Z}/\mathbf{Z} \subset \mathbf{T}$ is the identification sending u mod q to $\widetilde{u}/q$ mod $\mathbf{Z}$ where $\widetilde{u} \in \mathbf{Z}$ is an arbitrary element lifting u to $\mathbf{Z}$.
4. Output $(\mathbf{a}, s)$.

The LWE problem is the problem of finding $\mathbf{s}$ given access to polynomially many samples from the distribution $\mathcal{A}_{\mathbf{s},\chi}$. More formally, given a number of samples m, we define the distribution

Definition 10.3.2 (continuous LWE distribution)**.** The continuous LWE distribution on the set $(\mathbf{Z}/q\mathbf{Z})^{mn} \times \mathbb{T}^m$ is defined as

$$\texttt{cLWE}(q,n,m,\chi) := \{(A,\mathbf{b}) \colon \; \mathbf{b} = [\phi(A\mathbf{s}) + \mathbf{e}]_q, A \leftarrow_R (\mathbf{Z}/q\mathbf{Z})^{m \times n}, \mathbf{s} \leftarrow_R (\mathbf{Z}/q\mathbf{Z})^n, \mathbf{e} \leftarrow \chi^m\}.$$

10.3.2.2 Discrete version.

To describe the discrete version, consider the following parameters:

- q - modulus
- n - the dimension
- m - the number of samples
- χ - distribution on $\mathbf{Z}/q\mathbf{Z}$ supported on small integers, i.e., a distribution such that
$$\Pr[x \leftarrow \chi \colon |x| > \alpha q] < \texttt{negl}(n),$$
for some $\alpha \ll 1$. Example is a Gaussian distribution supported on the small integers **Dimitar : Be more precise** .

Definition 10.3.3 (Discrete LWE distribution)**.** The discrete LWE distribution on the set $(\mathbf{Z}/q\mathbf{Z})^{m(n+1)}$ is defined as follows:

$$\texttt{LWE}(q,n,m,\chi) := \{(A,\mathbf{b}) \colon \; \mathbf{b} = [A\mathbf{s} + \mathbf{e}]_q, A \leftarrow_R (\mathbf{Z}/q\mathbf{Z})^{m \times n}, \mathbf{s} \leftarrow_R (\mathbf{Z}/q\mathbf{Z})^n, \mathbf{e} \leftarrow \chi^m\}.$$

10.3.3 Decisional LWE problem

The search problem of computing $\mathbf{s}$ from the pair $(A, \mathbf{b})$ can be reduced (for various ranges of parameters) to the problem of distinguishing $(A, \mathbf{b})$ from a uniform pair from $(\mathbf{Z}/q\mathbf{Z})^{(m+1)\times n}$ [Reg09, Lem.4.2], [MP12]. The Decision LWE hardness assumption $\texttt{DLWE}(q, n, m, \alpha)$ is defined as follows:

10.3.4 Reductions

The first statement shows that if one can solve DLWE in the average case (for a random value of $\mathbf{s}$) then one can solve DLWE in the worst case (for any value of $\mathbf{s}$):

Lemma 10.3.4 (worst-case to average-case reduction)**.** *Suppose that one has an algorithm $\mathcal{D}$ (called* distinguisher*) that accepts samples from $\mathcal{A}_{\mathbf{s},\chi}$ with probability close to 1 and rejects samples from the uniform distribution $\mathcal{U}$ on $(\mathbf{Z}/q\mathbf{Z})^{n+1}$ with probability close to 1 for a non-negligible fraction of all possible $\mathbf{s} \in (\mathbf{Z}/q\mathbf{Z})^n$. Then there exists a distinguisher $\mathcal{D}'$ that distinguishes $\mathcal{A}_{\mathbf{s},\chi}$ from $\mathcal{U}$ for all $\mathbf{s}$.*

Regev [Reg09] also describes a worst-case to average-case quantum reduction from solving the shortest vector problem on an n-dimensional lattice to solving a random instance of LWE (the approximation factor is n/α).

Lemma 10.3.5 (decision-to-search/discrete)**.** *Suppose that one has a distinguisher $\mathcal{D}$ that distinguishes samples from $\mathcal{A}_{\mathbf{s},\chi}$ from samples from the uniform distribution $\mathcal{U}$ for all $\mathbf{s}$. Then there exists an algorithm $\mathcal{A}$ that, given samples from $\mathcal{A}_{\mathbf{s},\chi}$, outputs $\mathbf{s}$ with probability exponentially close to 1.* **Dimitar** ***: Be a bit more clear on the complexities.***

Finally, given a distribution χ on $\mathbb{T}$, one can discretize it to obtain a discrete distribution $\overline{\chi}$ on $(\mathbf{Z}/q\mathbf{Z})$ in the natural way: given a sample $x \leftarrow \chi$, we consider the sample

$$\overline{x} = \lfloor qx \rceil \bmod q \in \mathbf{Z}/q\mathbf{Z}.$$

More formally, if $\chi\colon \mathbb{T} \to \mathbf{R}_{>0}$ is the probability density function of χ, we define the probability density function $\overline{\chi}\colon \mathbf{Z}/q\mathbf{Z} \to \mathbf{R}_{>0}$ as

$$\overline{\chi}(u) := \int_{(u-1/2)/q}^{(u+1/2)/q} \chi(t)dt.$$

The following is an easy lemma that shows how to reduce the discrete version to a continuous one.

Lemma 10.3.6 (discrete-to-continuous)**.** *Let n, q and χ (on $\mathbf{T}$) be parameters as above for the continuous LWE problem. Suppose that there is an algorithm $\mathcal{W}$ to solve the continuous LWE problem in polynomial time. Then there is an algorithm $\mathcal{W}'$ that solves the discrete LWE problem for n, q and the distribution $\overline{\chi}$ on $(\mathbf{Z}/q\mathbf{Z})$ that is the discretization of χ.*

10.3.5 Quantum reduction to approximate SVP

10.3.6 Regev's public-key cryptosystem

Regev [Reg09] proposes a cryptosystem based on the hardness of LWE. The parameters are m, q and a probability distribution χ on T.
Dimitar : Conditions on the parameters?

KeyGen: The secret key SK is a vector $\mathbf{s} \in (\mathbf{Z}/q\mathbf{Z})^n$ chosen uniformly at random. To define the public key PK, choose m vectors $\mathbf{a}_i \in (\mathbf{Z}/q\mathbf{Z})^n$ independently and uniformly at random as well as errors $\mathbf{e}_1, \dots, \mathbf{e}_m \in \mathbf{T}$ independently according to the distribution χ. The public key PK is then $(\mathbf{a}_i, b_i := \langle \mathbf{a}_i, \mathbf{s}\rangle/q + \mathbf{e}_i)$.

Encrypt: To encrypt a bit $x \in \{0,1\}$, choose a random subset $S \subset \{1, \dots, m\}$ and define

$$\mathtt{Enc}(x) = \left(\sum_{i\in S} \mathbf{a}_i, x/2 + \sum_{i \in S} b_i\right)$$

Decrypt: To decrypt a ciphertext $(\mathbf{a}, b)$, compute $e = b - \langle \mathbf{a}, \mathbf{s}\rangle/q$ and define

$$\mathtt{Dec}(\mathbf{a}, b) := \begin{cases} 0 & \text{if } |e| < 1/4, \\ 1 & \text{otherwise.} \end{cases}$$

The correctness is established in the following lemma:

Lemma 10.3.7. *Assume that for any $k \in \{0, \dots, m\}$, $\chi^{\star k}$ satisfies*

$$\Pr_{e \leftarrow \chi^{\star k}} [|e| < 1/4] > 1 - \delta.$$

Then the probability of a decryption error is at most δ, that is, for any $x \in \{0,1\}$, if we use the above scheme to encrypt x and then decrypt the ciphertext, the result will be x with probability at least $1 - \delta$.

Proof. □

10.4 Ring Learning with Errors (Ring-LWE) Problem

10.4.1 Motivation

We start by considering a very simple hash function based on another classical problem, the *Short Integer Solution* (SIS) problem:

Short Integer Solution (SIS) Problem: Given parameters, n, m, q and an input matrix $A \in (\mathbf{Z}/q\mathbf{Z})^{n\times m}$, the goal is to find a non-zero vector $\mathbf{e} \in \{-1, 0, 1\}^m$ such that

$$A \cdot \mathbf{e} \equiv 0 \bmod q.$$

This yields a very simple hash function: $h_A(\mathbf{e}) = A \cdot \mathbf{e}$ for $\mathbf{e} \in \{0,1\}^m$ which is provably secure if worst-case lattice problems are hard.

The main issue with the construction is efficiency: reading the public description requires time $O(nm \log q)$ while collision is found in $2^{\mathcal{O}(m)}$ time. Can we modify the construction to optimize the performance? More specifically, is it possible to obtain a construction whose public description requires time linear in m.

10.4.2 Formulation and basic properties

We first formulate the `RingLWE` problem:

RingLWE Problem: Given $n = 2^s$, and integers m and a modulus q, the input consists of samples $a_1, \dots, a_m$ chosen uniformly at random from the ring $\mathcal{R}_q := \mathbf{Z}[x]/\langle x^n + 1, q\rangle$ together with $b_1, \dots, b_m \in \mathcal{R}_q$ such that $a_i = b_i s + e_i$ where $s \in \mathcal{R}_q$ is secret and $e_1, \dots, e_m \in \chi$, where χ is a distribution over short elements of $\mathcal{R}_q$. The goal is to compute s.

Note that we take $s \in R_q$ not to be random, but worst-case. If needed, one can randomize. There is a corresponding decisional version of the problem:

Decisional RingLWE Problem: Given $n = 2^s$, and integers m and a modulus q as well as $a_1, \dots, a_m$ chosen uniformly at random from the ring R_q together with $b_1, \dots, b_m \in \mathcal{R}_q$ such that $a_i = b_i s + e_i$ where $s \in R_q$ is secret and $e_1, \dots, e_m \in \chi$, where χ is a distribution over short elements of $\mathcal{R}_q$. The goal is to distinguish $\{(a_i, b_i) \colon i = 1, \dots, m\}$ from uniformly random pairs from $\mathcal{R}_q^2$.

10.4.3 Encryption schemes based on `RingLWE`

We now present two basic encryption schemes (a symmetric one and a public-key encryption one) based on the difficulty of solving `RingLWE`.

10.4.3.1 Symmetric encryption scheme

KeyGen: the secret key is a uniformly random element $\texttt{SK} \in \mathcal{R}_q$.

Encrypt: Plaintext messages are polynomials in $\mathcal{R}_q$ whose coefficients are elements of $\{0,1\}$. We denote the set of such messages by $\mathcal{B}_q$. To encrypt a plaintext message $\mu \in \mathcal{B}$, one chooses random polynomials $a \in \mathcal{R}_q$, $e \in \chi$ and defines the ciphertext

$$C_m = (a, b := a \cdot \texttt{SK} + e + \lfloor q/2 \rceil \mu).$$

Decrypt: To decrypt a ciphertext C, the owner of $\texttt{SK}$ simply computes $b - a \cdot \texttt{SK}$ and round to the nearest multiple of $\lfloor q/2 \rceil$. The decryption is correct as long as $e < q/4$.

10.4.3.2 Public-key encryption scheme

There is also a public key encryption scheme:

KeyGen: The secret key is a short vector $\mathtt{SK} \in \mathcal{R}_q$ drawn from a distribution χ. The public key is a pair $\mathtt{PK} := (a, b)$ where $a \in_R \mathcal{R}_q$ is uniformly random and $b = a \cdot \mathtt{SK} + e$ where $e \leftarrow \chi$ is a drawn from a noise distribution.

Encrypt: To encrypt a message $\mu \in \mathcal{B}_q$, choose $u, e_0, e_1 \leftarrow \chi$ and compute the ciphertext

$$C = au + e_0, bu + e_1 + \lfloor q/2 \rceil \mu \bmod q \in \mathcal{R}_q^2.$$

Decrypt: To decrypt a ciphertext $C = (C_0, C_1)$, compute $C_1 - \mathtt{SK} \cdot C_0 = eu + e_1 - \mathtt{SK} \cdot e_0$ and perform the rounding procedure. The scheme will be correct as long as $|eu + e_1 - \mathtt{SK} \cdot e_0| < q/4$.

10.5 Fully homomorphic encryption schemes

10.5.1 Historical overview

- First generation: Gentry (ideal lattices) [Gen09], van Dijk (integer arithmetic) [vDGHV10] - issues with growing noise. Ad hoc hardness assumptions (not standard classical assumptions) needed for bootstrapping.
- Second generation: Brakerski–Gentry–Vaikuntanathan [BGV12], Brakerski–Vaikuntanathan [BV14] - slowing noise growth from linear to logarithmic in the degree of the function, hence evaluation of circuits of fixed polynomial depth (levelled schemes) as well as bootstrappable schemes. More standard hardness assumptions.
- Third generation: GSW [GSW13] - asymmetric multiplication (i.e., encryption of $\mu_1 \times \mu_2$ is not the same as the encryption of $\mu_2 \times \mu_1$ allowing for even slower noise rates. Less efficient than second generation schemes, yet, conceptually simpler.

10.5.2 Gentry's original idea

Here, we present the integral version of [vDGHV10]: the secret key is a large odd number p. A bit $b \in \{0, 1\}$ will be encrypted into a ciphertext that is close to a multiple of p: more precisely, b gets encrypted to $c = pq + 2r + b$ so that $c \bmod p$ is congruent to b modulo 2, where the numbers q and r are random and $r \ll |p|$. To decrypt the ciphertext, one reduces it modulo p to get an integer in $[-p/2, p/2)$ and outputs the parity of the result, i.e., $b = (c \bmod p) \bmod 2$.

Given two ciphertexts, $c_1 = pq_1 + 2r_1 + b_1$ and $c_2 = pq_2 + 2r_2 + b_2$, the homomorphic sum and product are given by

$$c_+ = p(q_1 + q_2) + 2(r_1 + r_2) + b_1 \oplus b_2$$

and

$$c_\times = p(q_1 c_2 + q_2 c_1) + 2(b_1 r_2 + b_2 r_1 + 2 r_1 r_2) + b_1 b_2.$$

respectively.

1. The scheme of [vDGHV10] is proven to be secure if the Approximate GCD problem described in Section 9.4.4 is assumed to be hard.
2. Two problems: 1) non-compactness of the ciphertext, and 2) homomorphic evaluation (capacity) bound to low-degree polynomials;

10.5.3 Bootstrapping

To solve the problem of the limited homomorphic capacity, Gentry observed that one can turn any homomorphic scheme that can homomorphically evaluate its own decryption circuit (for any ciphertext) as well as a NAND gate. Gentry calls a scheme with these properties *bootstrappable.*

The idea is simple: if one publishes $\mathtt{Enc}_{\mathtt{SK}}(\mathtt{SK})$ then for any two ciphertexts c_1 and c_2, it is possible to homomorphically evaluate the function

$$F_{c_1,c_2}(\mathtt{SK}) := \mathtt{NAND}(\mathtt{Dec}_{\mathtt{SK}}(c_1), \mathtt{Dec}_{\mathtt{SK}}(c_2)).$$

This function depends on two fixed ciphertexts, c_1 and c_2 and it takes as input a secret key, decrypts the two ciphertexts and computes the NAND of the resulting plaintexts. If the homomorphic capacity of the scheme is sufficient to evaluate (homomorphically) the functions F_{c_1,c_2} for any two ciphertexts c_1, c_2 then the scheme is called *bootstrappable.*

The property of being *bootstrappable* guarantees that as long as the ciphertexts are properly decryptable, the homomorphic NAND is properly evaluated. Since it is applied to fresh encryption of the secret key SK and the ciphertexts are used in the definition of the function, the homomorphic NAND can be applied an arbitrary number of times while keeping the ciphertext compact (i.e., not increasing the size of the ciphertext), thus, obtaining a compact fully homomorphic encryption scheme.
Dimitar : Explain better. Not very clear.

10.5.4 Modulus and key switching

10.5.4.1 Modulus switching

Modulus switching allows to convert a system of approximate linear equations for one modulus into a system of approximate linear equations for another modulus.

Lemma 10.5.1. *Suppose that* $\mathbf{s}, \mathbf{u} \in (\mathbf{Z}/q\mathbf{Z})^n$ *are such that*

$$[\langle \mathbf{s}, \mathbf{u} \rangle]_q = b \left\lfloor \frac{q}{2} \right\rfloor + \delta,$$

where δ *.... Then if* $\mathbf{u}' = \left\lfloor \frac{q'u}{q} \right\rceil$ *then*

$$[\langle \mathbf{s}, \mathbf{u}' \rangle]_{q'} = b \left\lfloor \frac{q'}{2} \right\rfloor + \delta',$$

where δ' *...*

Proof. □

It is useful for replacing arithmetic modulo a large modulus q with arithmetic modulo a smaller modulus p at the expense of a slight noise increase when the secret key $\mathbf{s}$ is a vector of small norm. **Dimitar : Explain which schemes would benefit from reducing the modulus.**

10.5.4.2 Key switching

Key switching allows one to convert approximate linear relations for one secret key $\mathbf{s}$ into approximate linear relations for another secret key $\mathbf{s}'$.

10.5.5 The Brakerski–Fan–Vercauteren (BFV) scheme

A commonly known second-generation scheme is the scheme of Brakerski/Fan–Vercauteren (BFV) which we now describe.

The plaintext space in the BFV scheme is the ring $\mathcal{R}_p = (\mathbf{Z}/p\mathbf{Z})[X]/\langle X^N+1\rangle$ and the ciphertext space is $\mathcal{R}_q \times \mathcal{R}_q$ where $\mathcal{R}_q = (\mathbf{Z}/q\mathbf{Z})[X]/\langle X^N+1\rangle$ for a larger modulus $q > p$. Typically, p is either a power of 2 or a prime. We also let $\mathcal{R} = \mathbf{Z}[X]/\langle X^N+1\rangle$.

KeyGen: the secret key $\mathtt{SK}$ is a random polynomial from $\mathcal{R}_p$ with small coefficients (i.e., coefficients in $\{-1, 0, 1\}$). The public key $\mathtt{PK} = (\mathtt{PK}_1, \mathtt{PK}_2)$ is generated as follows: $\mathtt{PK}_2 \in_R \mathcal{R}_q$ is a random element and $\mathtt{PK}_1 = -(\mathtt{PK}_2 \cdot \mathtt{SK} + e) \in \mathcal{R}_q$. **Dimitar : Explain how $\mathtt{SK}$ is lifted as well as how e is sampled.**

Encrypt: To encrypt a plaintext message $\mu \in \mathcal{R}_p$, one chooses random polynomials $u \in \mathcal{R}$, $e_1, e_2 \in \mathcal{R}_q$ and defines the ciphertext $C = (a, b)$ where

$$a = [\mathtt{PK}_1 \cdot u + e_1 + \Delta\mu]_q, \qquad b = [\mathtt{PK}_2 \cdot u + e_2]_q.$$

Here $\Delta = \lfloor q/p \rfloor$ is the scaling factor.

Decrypt: To decrypt a ciphertext $C = (a, b)$, the owner of $\mathtt{SK}$ simply computes

$$\left\lfloor \frac{p}{q}\,(a + b \cdot \mathtt{SK}) \right\rceil \bmod p.$$

To verify that the decryption is correct, note that

$$\begin{aligned} a + b \cdot \mathtt{SK} &= \mathtt{PK}_1 \cdot u + e_1 + \Delta\mu + (\mathtt{PK}_2 \cdot u + e_2) \cdot \mathtt{SK} = \\ &= -(a \cdot \mathtt{SK} + e) \cdot u + e_1 + \Delta\mu + (a \cdot u + e_2) \cdot \mathtt{SK} = \\ &= \Delta\mu - e \cdot u + e_1 + e_2 \cdot \mathtt{SK}\,. \end{aligned}$$

The error term $v = -e \cdot u + e_1 + e_2 \cdot \mathtt{SK}$ has small norm since the polynomials e, e_1, e_2 and $\mathtt{SK}$ all have small coefficients. The decryption will be correct as long as

$$\frac{p}{q} \cdot \|v\| \leq \frac{1}{2}.$$

EvalAdd: To perform homomorphic addition, consider two ciphertexts $C_1 = (a_1, b_1)$ and $C_2 = (a_2, b_2)$. Adding them component-wise yields the ciphertext $C = C_1 + C_2 = (a, b)$ where

$$\begin{aligned} a &= a_1 + a_2 = [\mathtt{PK}_1 \cdot (u_1 + u_2) + (e_1^{(1)} + e_1^{(2)}) + \Delta(\mu_1 + \mu_2)]_q, \\ b &= b_1 + b_2 = [\mathtt{PK}_2 \cdot (u_1 + u_2) + (e_2^{(1)} + e_2^{(2)})]_q. \end{aligned}$$

Dimitar : Discuss how the noise grows.

EvalMult/Relinearization: To perform homomorphic multiplication, consider two ciphertexts $C_1 = (a_1, b_1)$ and $C_2 = (a_2, b_2)$. We express the plaintexts as

$$\mu_1 = \frac{p}{q}\left(\widetilde{a}_1 + \mathtt{SK} \cdot \widetilde{b}_2\right) \qquad \text{and} \qquad \mu_2 = \frac{p}{q}\left(\widetilde{a}_2 + \mathtt{SK} \cdot \widetilde{b}_2\right).$$

The homomorphic multiplication is achieved via a technique known as *relinearlization* that works as follows: write the plaintext product as

$$\mu_1\mu_2 \simeq \frac{p}{q}\left(\underbrace{\widetilde{b}_1\widetilde{b}_2}_{C_0} + \mathtt{SK}(\underbrace{\widetilde{a}_1\widetilde{b}_2 + \widetilde{a}_2\widetilde{b}_1}_{C_1}) + \mathtt{SK}^2 \underbrace{\widetilde{a}_1\widetilde{a}_2}_{C_2}\right).$$

The $\mathtt{SK}^2$-term now needs to be relinearized in order for us to interpret the above ciphertext as a linear function in $\mathtt{SK}$. The main idea is to mask $\mathtt{SK}^2$ by its encryption under another classical RLWE homomorphic encryption scheme **Dimitar : MAKE PRECISE!** , i.e., generate a random element $a_0 \in \mathcal{R}_q$ and consider the relinearization key

$$\mathtt{RK} = (\mathtt{RK}_0, \mathtt{RK}_1) = ([-a_0 \cdot \mathtt{SK} + e + \mathtt{SK}^2]_q, [a_0]_q),$$

where e is a small noise. We can then write a valid ciphertext for $\mu_1 \cdot \mu_2$ as

$$c = (a_1, a_2) \boxtimes_{\mathtt{BFV}} (a_2, b_2) := (C_0 + C_2 \cdot \mathtt{RK}_0, C_1 + C_2 \cdot \mathtt{RK}_1).$$

To check that this is a valid ciphertext, we write the decryption function

$$\begin{aligned} \mathtt{BFVDec}(c) &= \frac{p}{q}\left(C_0 + C_2 \cdot \mathtt{RK}_0 + \mathtt{SK} \cdot (C_1 + C_2 \cdot \mathtt{RK}_1)\right) = \\ &= \frac{p}{q}\left(C_0 + C_1 \,\mathtt{SK} + C_2(\mathtt{RK}_0 + \mathtt{SK} \cdot \mathtt{RK}_1)\right) = \mu_1\mu_2 - C_2 e. \end{aligned}$$

Now, the problem is that even if e is small, C_2 can be any element mod$q$ and hence, $C_2 e$ can be arbitrary large, thus blowing up the noise in the decryption. To remedy this issue, Fan–Vercauteren [FV12] originally propose two ideas (variants) which we explain in detail below:

1. *Slicing the relinearization key.*
2. *Modulus switching.*

10.5.5.1 Slicing the relinearization key.

10.5.5.2 Modulus switching.

10.5.6 BGV scheme

The BGV scheme is a second generation levelled scheme that that does not require Gentry's bootstrapping [BGV12]. Instead, it uses modulus switching for noise management.

Dimitar : Complete!

10.5.7 Cheon–Kim–Kim–Song (CKKS) scheme

The Cheon–Kim–Kim–Song (CKKS) scheme [CKKS17] is a homomorphic encryption scheme for approximate arithmetic. There is an original version based on the hardness of LWE, but as usual, this version is not efficient in terms of the ciphertext size. For an efficient version, we use the version based on `RingLWE`.

We will explain what the latter means by just looking at the way it encrypts and decrypts messages: the secret key is an element $\mathtt{SK} \in \mathcal{R}$ while the public key is a pair $(a, b := -a \cdot \mathtt{SK} + \mathbf{e}) \in \mathcal{R}_q^2$ where $\mathbf{e} \in \mathcal{R}$ is an error (chosen according to some error distribution).

(CKKS) Encryption: Using the public key $\mathtt{PK} = (a, b)$, we encrypt a message $\mu \in \mathcal{R}$ as follows: choose a random $u \in \mathcal{R}$ according to a distribution χ_{enc} and two error terms $\mathbf{e}_0, \mathbf{e}_1 \in \mathcal{R}$ and output the ciphertext

$$C := (bu + \mu, au).$$

(CKKS) Decryption: The owner of the secret key $\mathtt{SK}$ can decrypt a ciphertext $C = (C_0, C_1)$ by simply computing

$$C_0 + C_1 \cdot \mu.$$

Notice that this yields $\mu + e \sim \mu$ - an approximate plaintext, the reason why the scheme is an HE scheme for approximate arithmetic.

10.5.8 The GSW scheme

The GSW scheme [GSW13] is a third generation asymmetric scheme that we now review in detail.

10.5.8.1 High level intuition

Dimitar : Missing quantifiers - where does the key belong to, the message, C_u? Be more precise.

If $\mathbf{s}$ is the secret key (a vector), to encrypt a message u, construct a matrix C_u (the ciphertext) that has $\mathbf{s}$ as an eigenvector with eigenvalue u. Then homomorphic addition and multiplication is given by matrix addition and matrix multiplication:

$$(C_{u_1} + C_{u_2})\mathbf{s} = (u_1 + u_2)\mathbf{s}, \qquad (C_{u_1} C_{u_2})\mathbf{s} = u_1 u_2 \mathbf{s}.$$

This is insecure as one can compute the eigenvectors (one of them being the secret key) given the matrix (ciphertext) C.

10.5.8.2 Approximate eigenvectors

To remedy for this, we use approximate eigenvectors, i.e., the public key will be an approximate eigenvector such that

$$C_u \mathbf{s} = u\mathbf{s} + \mathbf{e}, \tag{10.1}$$

where $\mathbf{e}$ is a small noise. This idea will not help over the reals as computing eigenvectors is robust against small errors. Yet, over a discrete field (e.g., modulo an integer q) that can help.

What is the problem with the above scheme? It is the accumulation of noise after performing several multiplications. Let's do a calculation: if $C_{u_i}\mathbf{s} = u_i\mathbf{s} + \mathbf{e}_i$ then

$$C_{u_2}(C_{u_1}\mathbf{s}) = C_{u_2}\left(u_1\mathbf{s} + \mathbf{e}_1\right) = u_2 \cdot u_1 \mathbf{s} + u_1 \mathbf{e}_2 + C_{u_2}\mathbf{e}_1,$$

and hence, the noise of the product is $u_1\mathbf{e}_2 + C_{u_2}\mathbf{e}_1$ which can be large even if $\mathbf{e}_1$ and $\mathbf{e}_2$ are small.

10.5.8.3 Small plaintext scalars and ciphertext matrices

To fix the problem with the growing noise, we want to ensure that both the plaintext scalars as well as the ciphertext matrices are small.

To ensure that plaintexts are small, one can use an important property from logic (or digital electronics) called *functional completeness* of the `NAND` gate, that is, the property that any other logic function (e.g., `AND`, `OR`, `NOT`) can be expressed as a combination of `NAND` gates. If we treat the plaintext inputs as bits, a `NAND` gate can be expressed mathematically as $\texttt{NAND}(x, y) = 1 - xy$ over $\mathbb{Z}/q\mathbb{Z}$, thus ensuring that the output is also small.

For the ciphertext, we use a useful tool from lattice-based cryptography known as the *flattening gadget* on the second ciphertext C_2: a flattening gadget takes a vector of high norm and represents it in higher dimensions by a vector of low norm while preserving some basic linear-algebraic properties. More precisely, we want a function $f\colon \mathbb{Z}/q\mathbb{Z} \to (\mathbb{Z}/q\mathbb{Z})^{\ell}$ for some ℓ such that

1. For every $x \in \mathbb{Z}/q\mathbb{Z}$, the ℓ_∞-norm of $f(x)$ is much smaller than q,

2. Recovering x from $f(x)$ is a linear operation, that is, there exists a *gadget vector* $\mathbf{g}$ such that $z = \langle \mathbf{g}, f(z) \rangle$.

The important point is that f is a non-linear function, yet, f^{-1} is a linear function. For that reason, the literature often uses the notation g^{-1} for f to indicate that g (the inverse of f) is a linear function.

Example 5. A natural example of flattening gadget comes from the binary representation of x: if $x = \overline{x_{\ell-1}x_{\ell-2}\dots x_0}_{(2)}$ then we can define $f(x) = (x_0, \dots, x_{\ell-1})$ and $\mathbf{g} = (1, 2^1, \dots, 2^{\ell-1})$. It then follows that $\mathbf{x} = \langle \mathbf{g}, f(x)\rangle$. This example naturally generalizes to vectors of size n: if $\mathbf{v} = (x_1, \dots, x_n)$ then we define $G^{-1}(\mathbf{x}) = (f(x_1)|f(x_2)|\dots|f(x_n)) \in (\mathbb{Z}/q\mathbb{Z})^{n\ell}$. In this case, instead of a gadget vector, one uses a gadget matrix

$$G := \begin{pmatrix} 1 & 2 & \dots & 2^{\ell-1} & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & 2 & \dots & 2^{\ell-1} \end{pmatrix} \in (\mathbb{Z}/q\mathbb{Z})^{n\times n\ell}.$$

In this case, $G \cdot G^{-1}(\mathbf{x}) = \mathbf{x}$ and $\|G^{-1}(\mathbf{x})\| \leq 1$. Similarly, one can generalize the flattening gadget to matrices, namely, if $M = [\mathbf{c}_1|\mathbf{c}_2|\dots|\mathbf{c}_m] \in (\mathbb{Z}/q\mathbb{Z})^{n\times m}$, define

$$G^{-1}(M) := (G^{-1}(\mathbf{c}_1)|\dots|G^{-1}(\mathbf{c}_m)).$$

We can thus modify the GSW scheme as follows: instead of using the version from Section 10.5.8.2, we integrate G into (10.1), that is, we require that the ciphertext C_u satisfies

$$C_u\mathbf{s} = uG\mathbf{s} + \mathbf{e}, \tag{10.2}$$

for a small e.

We modify the homomorphic multiplication accordingly: if we now define the homomorphic product as

$$C_1 \boxtimes C_2 := C_1G^{-1}C_2 \tag{10.3}$$

then

$$C_{u_1} \boxtimes C_{u_2}\mathbf{s} \quad := \quad u_2C_{u_1}\mathbf{s} + G^{-1}C_{u_2}\mathbf{e}_2 = u_2u_1\mathbf{s} + u_2\mathbf{e_1} + G^{-1}C_{u_2}\mathbf{e}_2.$$

Since $u_2\mathbf{e_1}$ is small (assuming u_2 is small, e.g., a bit) and $G^{-1}C_{u_2}$ has low norm, then the vector $u_2\mathbf{e_1} + G^{-1}C_{u_2}\mathbf{e}_2$ is of low norm, i.e., $C_{u_1} \boxtimes C_{u_2}$ can be a valid ciphertext for u_1u_2.

10.5.8.4 Leveled homomorphic encryption

10.5.8.5 GSW can be made bootstrappable

10.5.9 TFHE scheme

- FHEW - Micciancio–Ducas [DM15]
- TFHE - [CGGI20]
- Scheme switching - CHIMERA [BGGJ20]

Bibliography

[AD97] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997 (Frank Thomson Leighton and Peter W. Shor, eds.), ACM, 1997, pp. 284–293. (↑ 85.)

[Ajt96] M. Ajtai, *Generating hard instances of lattice problems (extended abstract)*, Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996 (Gary L. Miller, ed.), ACM, 1996, pp. 99–108. (↑ 85.)

[AKS01] M. Ajtai, R. Kumar, and D. Sivakumar, *A sieve algorithm for the shortest lattice vector problem*, Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece (Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, eds.), ACM, 2001, pp. 601–610. (↑ 103.)

[Bar86] P. Barrett, *Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings (Andrew M. Odlyzko, ed.), Lecture Notes in Computer Science, vol. 263, Springer, 1986, pp. 311–323. (↑ 28, 29.)

[BDK98] J.-C. Bajard, L. Didier, and P. Kornerup, *An RNS montgomery modular multiplication algorithm*, IEEE Trans. Computers **47** (1998), no. 7, 766–776. (↑ 33.)

[BFO81] J. Brillhart, M. Filaseta, and A. Odlyzko, *On an irreducibility theorem of A. Cohn*, Canad. J. Math. **33** (1981), no. 5, 1055–1059. (↑ 55.)

[BGGJ20] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, *CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes*, J. Math. Cryptol. **14** (2020), no. 1, 316–338. (↑ 114.)

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, *(leveled) fully homomorphic encryption without bootstrapping*, Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012 (Shafi Goldwasser, ed.), ACM, 2012, pp. 309–325. (↑ 108, 112.)

[BI04] J.-C. Bajard and L. Imbert, *A full RNS implementation of RSA*, IEEE Trans. Computers **53** (2004), no. 6, 769–774. (↑ 33.)

[BKW00] A. Blum, A. Kalai, and H. Wasserman, *Noise-tolerant learning, the parity problem, and the statistical query model*, Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA (F. Frances Yao and Eugene M. Luks, eds.), ACM, 2000, pp. 435–440. (↑ 102, 103.)

[BMSZ13] J. W. Bos, P. L. Montgomery, D. Shumow, and G. Zaverucha, *Montgomery multiplication using vector instructions*, Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers (Tanja Lange, Kristin E. Lauter, and Petr Lisonek, eds.), Lecture Notes in Computer Science, vol. 8282, Springer, 2013, pp. 471–489. (↑ 32.)

[Bon99] D. Boneh, *Twenty years of attacks on the rsa cryptosystem*, NOTICES OF THE AMS **46** (1999), 203–213. (↑ 93.)

[BSS00] I. Blake, G. Seroussi, and N. Smart, *Elliptic curves in cryptography*, Cambridge University Press, Cambridge, 2000. (↑ 70.)

[BV14] Z. Brakerski and V. Vaikuntanathan, *Efficient fully homomorphic encryption from (standard) LWE*, SIAM J. Comput. **43** (2014), no. 2, 831–871. (↑ 108.)

[CGGI20] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, *TFHE: fast fully homomorphic encryption over the torus*, J. Cryptol. **33** (2020), no. 1, 34–91. (↑ 114.)

[CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, *Homomorphic encryption for arithmetic of approximate numbers*, Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Tsuyoshi Takagi and Thomas Peyrin, eds.), Lecture Notes in Computer Science, vol. 10624, Springer, 2017, pp. 409–437. (↑ 112.)

[Cop94] D. Coppersmith, *Solving homogeneous linear equations over* GF(2) *via block Wiedemann algorithm*, Math. Comp. **62** (1994), no. 205, 333–350. (↑ 51.)

[Cop01] ______, *Finding small solutions to small degree polynomials*, Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers (Joseph H. Silverman, ed.), Lecture Notes in Computer Science, vol. 2146, Springer, 2001, pp. 20–31. (↑ 93.)

[CT65] J. Cooley and J. Tukey, *An algorithm for the machine calculation of complex fourier series*, Mathematics of Computation **19** (1965), no. 90, 297–301. (↑ 15.)

[dB51] N. G. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factors $> y$*, Nederl. Acad. Wetensch. Proc. Ser. A. **54** (1951), 50–60. (↑ 52.)

[Dic30] K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Ark. Mat. Astr. Fys. **22** (1930), 1–14. (↑ 52.)

[Dix81] J. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981), no. 153, 255–260. (↑ 51.)

[DJ] I. Damgård and M. Jurik, *A generalisation, a simplification and some applications of paillier's probabilistic public-key system*, Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings (Kwangjo Kim, ed.), Lecture Notes in Computer Science, vol. 1992, pp. 119–136. (↑ 102.)

[DM15] L. Ducas and D. Micciancio, *FHEW: bootstrapping homomorphic encryption in less than a second*, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I (Elisabeth Oswald and Marc Fischlin, eds.), Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 617–640. (↑ 114.)

[EKM17] A. Esser, R. Kübler, and A. May, *LPN decoded*, Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II (Jonathan Katz and Hovav Shacham, eds.), Lecture Notes in Computer Science, vol. 10402, Springer, 2017, pp. 486–514. (↑ 103.)

[Erd60] P. Erdös, *Remarks on number theory. III. On addition chains*, Acta Arith. **6** (1960), 77–81. (↑ 24.)

[Für07] M. Fürer, *Faster integer multiplication*, Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007 (David S. Johnson and Uriel Feige, eds.), ACM, 2007, pp. 57–66. (↑ 19.)

[FV12] J. Fan and F. Vercauteren, *Somewhat practical fully homomorphic encryption*, IACR Cryptol. ePrint Arch. (2012), 144. (↑ 111.)

[Gar59] H. Garner, *The residue number system*, Papers presented at the the 1959 western joint computer conference, IRE-AIEE-ACM 1959 (Western), San Francisco, California, USA, March 3-5, 1959 (R. R. Johnson, ed.), ACM, 1959, pp. 146–153. (↑ 32.)

[Gen09] C. Gentry, *Fully homomorphic encryption using ideal lattices*, Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009 (M. Mitzenmacher, ed.), ACM, 2009, pp. 169–178. (↑ 108.)

[GGH97] O. Goldreich, S. Goldwasser, and S. Halevi, *Public-key cryptosystems from lattice reduction problems*, Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings (Burton S. Kaliski Jr., ed.), Lecture Notes in Computer Science, vol. 1294, Springer, 1997, pp. 112–131. (↑ 85.)

[GK15] S. Gueron and V. Krasnov, *Fast prime field elliptic-curve cryptography with 256-bit primes*, J. Cryptogr. Eng. **5** (2015), no. 2, 141–151. (↑ 32.)

[Gra08a] A. Granville, *Smooth numbers: computational number theory and beyond*, Algorithmic number theory: lattices, number fields, curves and cryptography, Math. Sci. Res. Inst. Publ., vol. 44, Cambridge Univ. Press, Cambridge, 2008, pp. 267–323. (↑ 60.)

[Gra08b] ______, *Smooth numbers: computational number theory and beyond*, Algorithmic number theory, vol. 4, 2008. (↑ 65.)

[GSW13] C. Gentry, A. Sahai, and B. Waters, *Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based*, Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I (Ran Canetti and Juan A. Garay, eds.), Lecture Notes in Computer Science, vol. 8042, Springer, 2013, pp. 75–92. (↑ 108, 112.)

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman, *NTRU: A ring-based public key cryptosystem*, Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings (Joe Buhler, ed.), Lecture Notes in Computer Science, vol. 1423, Springer, 1998, pp. 267–288. (↑ 85, 97.)

[HvdH21] D. Harvey and J. van der Hoeven, *Integer multiplication in time $O(n \log n)$*, Ann. of Math. (2) **193** (2021), no. 2, 563–617. (↑ 19.)

[JF09] S. G. Johnson and M. Frigo, *Implementing ffts in practice **, 2009. (↑ 16.)

[Jia20] L. Jiao, *Specifications and improvements of LPN solving algorithms*, IET Inf. Secur. **14** (2020), no. 1, 111–125. (↑ 103.)

[Kan83] R. Kannan, *Improved algorithms for integer programming and related lattice problems*, Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA (David S. Johnson, Ronald Fagin, Michael L. Fredman, David

Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, eds.), ACM, 1983, pp. 193–206. (↑ 99.)

[KMPT10] J. Kim, R. Montenegro, Y. Peres, and P. Tetali, *A birthday paradox for Markov chains with an optimal bound for collision in the Pollard rho algorithm for discrete logarithm*, Ann. Appl. Probab. **20** (2010), no. 2, 495–521. (↑ 63.)

[Knu] Donald E. Knuth, *The art of computer programming. Vol. 2*, third ed., Addison-Wesley Publishing Co., Reading, Mass., Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing. (↑ 28.)

[Knu97] D. E. Knuth, *Seminumerical algorithms*, 3rd ed., The Art of Computer Programming, Addison-Wesley, Reading, Massachusetts, USA, 1997. (↑ 63.)

[KO63] A. Karatsuba and Y. Ofman, *Multiplication of multidigit numbers on automata*, Soviet Physics Doklady **7** (1963), 595–596. (↑ 11.)

[KS01] R. Kumar and D. Sivakumar, *On polynomial approximation to the shortest lattice vector length*, Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA (S. Rao Kosaraju, ed.), ACM/SIAM, 2001, pp. 126–127. (↑ 103.)

[LL93a] A. K. Lenstra and H. W. Lenstra, *The development of the number field sieve*, Springer-Verlag, Berlin, 1993. (↑ 69.)

[LL93b] A.K. Lenstra and H.W. Lenstra, *The development of the number field sieve*, Lecture Notes in Math., vol. 1554, Springer, Berlin, 1993, pp. 103–126. (↑ 54.)

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), 515–534. (↑ 85.)

[LO90] B. LaMacchia and A. Odlyzko, *Solving large sparse linear systems over finite fields*, Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings (A. Menezes and S. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 109–133. (↑ 51.)

[Mon83] P. L. Montgomery, *Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via Lucas chains*, unpublished (1983). (↑ 24.)

[Mon85] P. Montgomery, *Modular multiplication without trial division*, Math. Comp. **44** (1985), no. 170, 519–521. (↑ 29.)

[MP12] D. Micciancio and C. Peikert, *Trapdoors for lattices: Simpler, tighter, faster, smaller*, Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings (David Pointcheval and Thomas Johansson, eds.), Lecture Notes in Computer Science, vol. 7237, Springer, 2012, pp. 700–718. (↑ 105.)

[MPS07] A. Moss, D. Page, and N. P. Smart, *Toward acceleration of RSA using 3d graphics hardware*, Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings (Steven D. Galbraith, ed.), Lecture Notes in Computer Science, vol. 4887, Springer, 2007, pp. 364–383. (↑ 33.)

[MV06] S. Miller and R. Venkatesan, *Spectral analysis of Pollard rho collisions*, Algorithmic number theory, Lecture Notes in Comput. Sci., vol. 4076, Springer, Berlin, 2006, pp. 573–581. (↑ 63.)

[Odl84] A. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, EUROCRYPT, 1984, pp. 224–314. (↑ 69.)

[Pol75] J. M. Pollard, *A Monte Carlo method for factorization*, BIT Numerical Mathematics **15** (1975), 331–334. (↑ 62.)

[Pol78] ______, *Monte Carlo methods for index computation (mod p)*, Math. of Comp. **32** (1978), 918–924. (↑ 62.)

[Pom82] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, Computational methods in number theory, Part I, Math. Centre Tracts, vol. 154, Math. Centrum, Amsterdam, 1982, pp. 89–139. (↑ 54.)

[Pom85] ______, *The quadratic sieve factoring algorithm*, Advances in cryptology (Paris, 1984), Lecture Notes in Comput. Sci., vol. 209, Springer, Berlin, 1985, pp. 169–182. (↑ 54.)

[Pom95] ______, *The role of smooth numbers in number-theoretic algorithms*, Proceedings of the International Congress of Mathematicians, Vol. 1, 2 (Zürich, 1994), Birkhäuser, Basel, 1995, pp. 411–422. (↑ 60.)

[Pom96] ______, *Multiplicative independence for random integers*, Analytic number theory, Vol. 2 (Allerton Park, IL, 1995), Progr. Math., vol. 139, Birkhäuser Boston, Boston, MA, 1996, pp. 703–711. (↑ 60.)

[PP95] K. C. Posch and R. Posch, *Modulo reduction in residue number systems*, IEEE Trans. Parallel Distributed Syst. **6** (1995), no. 5, 449–454. (↑ 33.)

[Reg09] O. Regev, *On lattices, learning with errors, random linear codes, and cryptography*, J. ACM **56** (2009), no. 6, 34:1–34:40. (↑ 102, 103, 105, 106.)

[SG08] R. Szerwinski and T. Güneysu, *Exploiting the power of gpus for asymmetric cryptography*, Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings (Elisabeth Oswald and Pankaj Rohatgi, eds.), Lecture Notes in Computer Science, vol. 5154, Springer, 2008, pp. 79–99. (↑ 33.)

[Sha71] D. Shanks, *Class number, a theory of factorization, and genera*, Symposia in Pure Mathematics, vol. 20, 1971, pp. 415–440. (↑ 63.)

[Sil92] J. H. Silverman, *The arithmetic of elliptic curves*, Springer-Verlag, New York, 1992, Corrected reprint of the 1986 original. (↑ 70.)

[SLG$^+$14] H. Seo, Z. Liu, J. Großschädl, J. Choi, and H. Kim, *Montgomery modular multiplication on ARM-NEON revisited*, Information Security and Cryptology - ICISC 2014 - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers (Jooyoung Lee and Jongsung Kim, eds.), Lecture Notes in Computer Science, vol. 8949, Springer, 2014, pp. 328–342. (↑ 32.)

[SS71] A. Schönhage and V. Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing (Arch. Elektron. Rechnen) **7** (1971), 281–292. (↑ 16.)

[ST92] J. H. Silverman and J. Tate, *Rational points on elliptic curves*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1992. MR 93g:11003 (↑ 70.)

[TV22] B. Tran and S. Vaudenay, *Solving the learning parity with noise problem using quantum algorithms*, Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings (Lejla Batina and Joan Daemen, eds.), Lecture Notes in Computer Science, Springer Nature Switzerland, 2022, pp. 295–322. (↑ 103.)

[vDGHV10] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, *Fully homomorphic encryption over the integers*, Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 24–43. (↑ 108, 109.)

[Wie86] D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986), no. 1, 54–62. (↑ 51.)

[YL93] S.-M. Yen and C.-S. Laih, *Common-multiplicand multiplication and its applications to public key cryptography*, Electronics Letters **29** (1993), 1583–1584. (↑ 25.)