GROUP CRYPTOGRAPHY AND DISCRETE LOGARITHMS

Enes KALECI

MATH 411 Capstone Project 1
Advisor: Doga Can Sertbas

Istinye University
Faculty of Engineering and Natural Sciences
Mathematics Department
January 2025

GROUP CRYPTOGRAPHY AND DISCRETE LOGARITHMS

Enes KALECI

21/01/2025

Abstract

In my report, the fundamental topics of group cryptography and the discrete logarithm problem are discussed. The properties of groups in cryptography and their importance in secure communication are addressed.

In my report, the widely used cryptographic method known as the ElGamal encryption scheme is introduced, along with an examination of methods such as the baby-step giant-step algorithm, Pollard's rho algorithm, and the Pohlig-Hellman algorithm.

Additionally, cryptographic attacks such as types of birthday attacks and how calculations are carried out using the method of index computation are also covered. Furthermore, the use of targeted mathematical techniques like the Chinese distribution for discrete logarithms is discussed. The section also examines how modular arithmetic systems based on discrete logarithms contribute to making cryptographic systems faster and more secure.

Overall, this section explains the basic concepts of group cryptography and the applications of the discrete logarithm problem in cryptography in a simple and comprehensible manner.

Contents

1 Introduction

Cryptography is a field of study that aims to secure information by employing mathematical methods. Throughout human history, the transmission of secret messages has always been of significant importance. Today, cryptography is widely used to fulfill this purpose.

The history of cryptography dates back to the Caesar cipher used in the Roman Empire. This method is considered the first symmetric encryption technique. However, modern cryptography offers much more complex and secure systems compared to the Caesar cipher. With the help of mathematical techniques, modern cryptography achieves a high level of security that is difficult to breach.

Group cryptography is an example of modern cryptographic techniques. By leveraging the mathematical concept of "groups," it ensures the accuracy and security of encryption and decryption processes. The primary element that provides security in this system is the difficulty of the discrete logarithm problem. To decrypt the information, one must solve this challenging problem. Although various algorithms and techniques have been developed to solve the discrete logarithm, it remains computationally infeasible in practice, offering robust security.

2 The birthday paradox

The birthday paradox states that in an environment with 23 different people, the probability that two of them share the same birthday is greater than 50%. This paradox implies that a collision can be detected in $O(\sqrt{m})$ time.

Theorem 1. *We consider random choices, with replacement, among p labeled items. The expected number of choices until a collision occurs is $O(\sqrt{m})$.*

Proof. The probability that no collision occurs in j chances is given by:

$$P(s > j) = \prod_{i=1}^{j-1} \left(1 - \frac{i}{m}\right)$$

Using $1 - x \leq e^{-x}$, we approximate:

$$P(s > j) \approx \prod_{i=1}^{j-1} e^{-\frac{i}{m}} = e^{-\frac{1}{m}\sum_{i=1}^{j-1} i}$$

$$P(s > j) \approx e^{-\frac{(j-1)j}{2m}}$$

The expected value of s is given by:

$$\mathbb{E}[s] = \sum_{j=1}^{\infty} P(s > j) \approx \sum_{j=1}^{\infty} e^{-\frac{(j-1)j}{2m}}$$

We approximate the sum as an integral:

$$\mathbb{E}[s] = \int_0^{\infty} e^{-\frac{x(x-1)}{2m}}\, dx$$

Let $u = \frac{x^2}{2m}$, $x = \sqrt{2mu}$, and $dx = \frac{\sqrt{m}}{\sqrt{2u}}\, du$. Substituting:

$$\int_0^{\infty} e^{-\frac{x(x-1)}{2m}}\, dx = \int_0^{\infty} e^{-u} u^{-1/2}\, du$$

The integral evaluates to:

$$\int_0^{\infty} e^{-u} u^{-1/2}\, du = \Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$$

Thus:

$$\mathbb{E}[s] = 2 + \sqrt{\frac{\pi m}{2}}$$

□

3 The Diffie-Hellman key exchange

Using the Diffie-Hellman key exchange, secure communication can be established even over an insecure channel. This system does not transmit the secret key directly; instead, each party generates it themselves. Diffie-Hellman operates on a prime number p and the group $\mathbb{Z}_p^*$. It supports the system with its multiplicative inverses and cyclic structure. The group G is defined using a generator g, and group operations are performed in polynomial time.

Protocol 1. Diffie-Hellman key exchange.

Set-up:

- **Input:** Security parameter n.
- **Output:** $G, g,$ and d as below.
- Determine a description of a finite cyclic group $G = \langle g \rangle$ with $d = \#G$ elements and a generator g of G, where d is an n-bit integer.

Key exchange:

1. Alice chooses her secret key $a \leftarrow \mathbb{Z}_d$. She computes her public key $A \leftarrow g^a \in G$.
2. Bob chooses his secret key $b \leftarrow \mathbb{Z}_d$. He computes his public key $B \leftarrow g^b \in G$.
3. Alice and Bob exchange their public keys A and B.
4. Alice computes the shared secret key $k_A = B^a$.
5. Bob computes the shared secret key $k_B = A^b$.

Remark 1. In the Diffie-Hellman key exchange, Alice and Bob choose their secret keys $a, b \in \mathbb{Z}_d$. They compute their public keys as follows:

$$A = g^a \mod d, \quad B = g^b \mod d.$$

After exchanging public keys, they compute the shared secret:

$$k_A = B^a \mod d = g^{ab} \mod d, \quad k_B = A^b \mod d = g^{ab} \mod d.$$

Thus, $k_A = k_B$, and both parties derive the same shared secret.

4 Group Cryptography and Discrete Logarithms

Important tools for asymmetric cryptography are protocols that operate on groups. These protocols are generally introduced in the multiplicative groups of finite fields ($\mathbb{Z}_p^*$).

The ElGamal encryption protocol is introduced, and attacks on the security of group cryptography as well as discrete logarithm algorithms are discussed. Groups of large prime order are recommended as they are more secure against efficient attacks.

Additionally, there are specific algorithms related to solving discrete logarithms, and these algorithms will be examined.

4.1 Groups for cryptography

Groups of interest in cryptography:

1. the multiplicative group $G = \mathbb{Z}_p^*$ of units modulo a prime p,
2. the multiplicative group $G = \mathbb{F}_q^*$ of a finite field $\mathbb{F}_q$,
3. elliptic curves and their subgroups,
4. large subgroups of $\mathbb{Z}_p^*$ for a huge prime p

The Diffie-Hellman key exchange uses a group G and a generator g. In this method, both parties choose random secret numbers, create public keys g^a and g^b, and exchange them. Using their secret numbers, they calculate a shared secret key g^{ab}. The security of this method relies on the difficulty of solving the discrete logarithm problem.

The discrete logarithm problem is finding a in the equation $x = g^a$ where $x \in G$. This problem is very hard to solve.

Definition 2. Let G be a finite group.

(i) The general discrete logarithm problem in G is, given two elements g and x of G, to decide whether $x \in \langle g \rangle$, and if so, compute some $a \in \mathbb{Z}$ with $x = g^a$.

(ii) We are often given additional information:

- Guaranteed membership: $x \in \langle g \rangle$ is true,
- Cyclic group: we have $G = \langle g \rangle$,
- Group order: we are given $\#G$,
- Inverse logarithm: we are given $b \in \mathbb{Z}$ with $g = x^b$.

(iii) When the group order $d = \#G$ is given, we usually require the smallest nonnegative value for a, that is, $a \in \mathbb{Z}_d = \{0, 1, \ldots, d-1\}$.

(iv) We write DL_G for the computational version of the most common form of the discrete logarithm problem in the finite group G: a generator g of $G = \langle g \rangle$ and $d = \#G$ are given, the input is some $x \in G$, and the output is the unique integer a with $x = g^a$ and $0 \leq a < d$.

There are some algorithms to solve discrete logarithms, some of them are:

1. **Baby-step giant-step attack:** works for any G and takes time and space $\sqrt{d}$.
2. **Pollard rho:** works for any G and takes time $\sqrt{d}$ and constant space.
3. **Chinese distribution:** reduces from general d to prime power d.
4. **Pohlig-Hellman algorithm:** reduces from prime power d to prime d.
5. **Index calculus:** works for $G = \mathbb{Z}_p^*$, p a prime, and is more efficient than the methods above. It can be generalized to $G = \mathbb{F}_p^*$ with q a prime power.

4.2 The ElGamal encryption scheme

ElGamal enables secure message exchange using the Diffie-Hellman protocol. The following protocol explains this cryptosystem.

Protocol 2. ElGamal Encryption Scheme
Set-up:

- **Input:** 1^n.
- Choose a finite cyclic group $G = \langle g \rangle$ with d elements, where d is an n-bit integer, and a generating element g. These data, including a description of G, are made public.

Key Generation:

- **Output:** $(\mathrm{sk}, \mathrm{pk})$.
- Secret key $\mathrm{sk} = b \leftarrow \mathbb{Z}_d$. Public key $\mathrm{pk} = B \leftarrow g^b \in G$.

Encryption:

- **Input:** Plaintext $x \in G$.

- **Output:** Ciphertext $\text{enc}_{\text{pk}}(x) \in G^2$.
- Choose a secret session key $a \leftarrow \mathbb{Z}_d$ at random.
- Public session key $A \leftarrow g^a \in G$, and $k \leftarrow B^a$.
- Compute $y \leftarrow x \cdot k \in G$.
- Return $\text{enc}_{\text{pk}}(x) = (y, A)$.

Decryption:

- **Input:** Arbitrary $(y, A) \in G^2$.
- **Output:** Decryption $\text{dec}_{\text{sk}}(y, A) \in G$.
- Compute session key $k \leftarrow A^b$, and its inverse $k^{-1} \in G$.
- Compute $z \leftarrow y \cdot k^{-1}$.
- Return z.

ElGamal encryption works correctly because the decryption part of the protocol takes (y, A) as input, where y is the encrypted message and A is Alice's public key. Bob can easily calculate the shared secret key using Alice's public key A, as

$$A^b = (g^a)^b = k.$$

Since $k \in G$, it has an inverse, and

$$z = y \cdot k^{-1}.$$

This is well-defined, and

$$z = y \cdot k^{-1} = x \cdot B^a \cdot (B^a)^{-1} = x,$$

proving that the system works correctly.

The protocol completes in $O(n^3)$ operations because the repeated squaring algorithm is used for computing exponentials.

4.3 Baby-step giant-step algorithm

In this section, we will examine an algorithm used for DLG. This algorithm operates in the group G and uses $\sqrt{d}$ time, where $d = \#G$. The required steps in the algorithm are also $\sqrt{d}$. The main idea of the algorithm is to detect a collision. The identified collision provides information about the discrete logarithm.

Algorithm 1. Baby-step giant-step algorithm for the discrete logarithm.
Input:

- A cyclic group $G = \langle g \rangle$ with d elements.
- A group element $x \in G$.

Output: $\mathrm{dlog}_g x$.
Algorithm:

1. $m \leftarrow \lceil \sqrt{d} \rceil$.
2. **Baby steps:** Compute and store $x, xg, xg^2, \ldots, xg^m$ in a table.
3. **Giant steps:** Compute $g^{-m}, x \cdot g^{-m}, x \cdot g^{-2m}, x \cdot g^{-3m}, \ldots$ until one of them, say xg^j, equals an element in the table.
4. Return $im - j$ in $\mathbb{Z}_d$.

Theorem 3. *For any group G with d elements, the baby-step giant-step method solves DL_G with at most $2m$ group operations and space for m elements of G, where $m = \lceil \sqrt{d} \rceil$.*

Proof. The algorithm works correctly because if a collision is found, $g^{im} = xg^j$, and from this, $x = g^{im-j}$ is obtained. This means $\mathrm{dlog}_g(x) = im - j$. Therefore, the algorithm is correct for this reason.
The Baby-step Giant-step method solves the discrete logarithm problem by dividing $a = im + j$, where i and j must satisfy $0 \leq i, j < m$, and $m = \sqrt{d}$.

In the Baby-step phase, g^{im} values are computed and stored; this requires m group operations.

In the Giant-step phase, g^{im} values are computed and compared with the Baby-step table, also requiring m group operations.

As a result, the algorithm completes in $2m$ group operations.

□

In the Baby-Step and Giant-Step algorithms, the calculations are performed using the repeated squaring algorithm. So, each computation is carried out in $O(\log d)$ bit operations.

4.4 The birthday attack

This algorithm takes its name from the birthday paradox. For random values of i and j, it computes xg^i and g^j, waiting for a collision. When a collision $xg^i = g^j$ is detected, we obtain $x = g^{j-i}$. This algorithm serves as a preparation for Pollards Rho method.

Algorithm 2. Birthday algorithm for discrete logarithm
Input: A cyclic group $G = \langle g \rangle$ with d elements, and a group element $x \in G$.
Output: $\mathrm{dlog}_g x$.

1. $X, Y \leftarrow \emptyset$.
2. Do step 3 until a collision of X and Y occurs.
3. Choose uniformly at random a bit $b \leftarrow \{0, 1\}$ and $i \leftarrow \{0, \ldots, d-1\}$.
   - Add xg^i to X if $b = 0$.
   - Add g^i to Y if $b = 1$, and remember the index i.
4. If $xg^i = g^j$ for some $xg^i \in X$ and $g^j \in Y$, then return $j - i$ in $\mathbb{Z}_d$.

Theorem 4. *The algorithm works correctly as specified. Its expected time is $O(\sqrt{d} \log d)$ multiplications in G, with expected space for $O(\sqrt{d})$ elements of G.*

Proof. The elements of the sets X and Y are independent and randomly distributed in the group G. According to the Birthday Paradox, a collision in G is expected after $O(\sqrt{d})$ steps. If a collision $xg^i = g^j$ occurs, the equation $\mathrm{dlog}_g x = j - i$ holds.

The elements of X and Y are calculated using RSA, which costs $O(\log d)$ group operations. Since the algorithm performs $O(\sqrt{d})$ operations, the total cost is $O(\sqrt{d} \log d)$.

□

4.5 Chinese distribution of discrete logarithms

In this algorithm, we have a finite cyclic group $G = \langle g \rangle$ with $d = \#G$ elements. Take a factorization $d = q_1 \cdots q_r$, where $q_1, \ldots, q_r$ are pairwise coprime. The map $\pi_{d/q} : G \to G$, which raises the group elements to the power d/q, is considered. With this map, we transfer our group elements to the subgroup and solve the discrete logarithm in that subgroup.

Lemma 5. *Suppose that $d = q_1 q_2$ with coprime q_1 and q_2, and that $a_i = dlog_{g_i} x_i$ in $\mathbb{Z}_{q_i}$, where $g_i = g^{d/q_i}$ and $x_i = x^{d/q_i} \in \pi_{d/q_i}(G)$, for $i = 1, 2$. Then $dlog_g x = a_i$ in $\mathbb{Z}_{q_i}$ for $i = 1, 2$.*

Proof. Let $d = q_1 q_2$ with q_1 and q_2 coprime. Define $g_i = g^{d/q_i}$ and $x_i = x^{d/q_i}$ for $i = 1, 2$. By the definition of the discrete logarithm, a_i satisfies $x_i = g_i^{a_i}$. Substituting g_i and x_i, we get $x^{d/q_i} = g^{(d/q_i)a_i}$. Since q_1 and q_2 are coprime, the Chinese Remainder Theorem ensures a unique $a \in \mathbb{Z}_d$ such that $a \equiv a_i \pmod{q_i}$ for $i = 1, 2$. Thus, we can write $x = g^a$, and $\mathrm{dlog}_g x = a_i$ in $\mathbb{Z}_{q_i}$ for $i = 1, 2$.

□

Algorithm 3. Chinese remaindering for discrete logarithms.
Input: A cyclic group $G = \langle g \rangle$ of order $d = \#G$, and $x \in G$.
Output: $a = \mathrm{dlog}_g x$.

1. Compute the prime power factorization of d.
2. For each $i \leq r$, do steps 3 and 4:
   (a) Compute $g_i = g^{d/q_i}$ and $x_i = x^{d/q_i}$, using the Repeated Squaring Algorithm
   (b) Compute the discrete logarithm $a_i = \mathrm{dlog}_{g_i} x_i \in \mathbb{Z}_{q_i}$ in $S_i = \langle g_i \rangle$.
3. Combine these "small" discrete logarithms via the Chinese Remainder Algorithm to find the unique $a \in \mathbb{Z}_d$ such that $a = a_i$ in $\mathbb{Z}_{q_i}$ for all $i \leq r$.

Theorem 6. *Let G be a cyclic group of n-bit order d. Then Algorithm 4.23 computes a discrete logarithm in G at the following cost:*

(i) factoring the integer d,

(ii) one discrete logarithm in each of the groups $S_1, \ldots, S_r$,

(iii) $O(n^2)$ operations in G,

(iv) $O(n^3)$ bit operations.

Proof. To compute all g_i and x_i, it requires at most $2r \cdot 2n \leq 4n^2$ operations, which is equivalent to $O(n^2)$ group operations. Moreover, computing g_i and x_i involves using Repeated Squaring Algorithm (RSA), and this algorithm performs $O(n^3)$ bit operations.

□

4.6 The Pohlig-Hellman algorithm

This algorithm works in the cyclic group $G = \langle g \rangle$ of order p^e, where p is a prime number and $e \geq 2$ is an integer. Using the

$$\pi_{p^{e-1}} : x \mapsto x^{p^{e-1}}$$

map that raises to the p^{e-1}-th power, we transfer to the subgroup $H = \langle g^{p^{e-1}} \rangle$. The order of this subgroup is p. After transferring the elements to the subgroup, we solve the discrete logarithm problem using the elements of the subgroup. The discrete logarithms we solve allow us to write the desired element $a = \text{dlog}(x)$ in base p as

$$a = a_{e-1} \cdot p^{e-1} + \ldots + a_0.$$

Algorithm 4. Pohlig-Hellman Algorithm
Input: A cyclic group $G = \langle g \rangle$ of order p^e, where p is a prime, $e \geq 2$ is an integer, and $x \in G$.
Output: $a = \text{dlog}_G(x)$, the discrete logarithm of x in G.

1. Compute $h = g^{p^{e-1}}$ and set $y_{-1} = 1 \in G$.
2. For $i = 0$ to $e - 1$, do:
   (a) $x_i \leftarrow (x \cdot y_{i-1}^{-1})^{p^{e-i-1}}$
   (b) Compute $a_i \leftarrow \text{dlog}_h(x_i)$.
   (c) Update $y_i \leftarrow y_{i-1} \cdot g^{-a_i p^i}$.
3. Return $a = a_{e-1}p^{e-1} + \cdots + a_0$.

Theorem 7. *The algorithm correctly computes $dlog_g x$. It uses $O(e^2 \log p)$ operations in G.*

Proof. Let $a = a_{e-1}p^{e-1} + \cdots + a_0$ be the representation of $\text{dlog}_g x$ in base p, and a_i and y_i for $0 \leq i < e$ as computed in the algorithm. We begin with $i \geq 0$ and proceed as follows.

$$\begin{aligned} x_i = (xy_{i-1})^{p^{e-i-1}} &= g^{a_0+a_1p+a_2p^2+\cdots+a_{e-1}p^{e-1}} \cdot g^{-(a_0+a_1p+a_2p^2+\cdots+a_{i-1}p^{i-1})} \\ &= g^{b_ip^{e-i-1}+b_{i+1}p^{e-i}+b_{i+2}p^{e-i+1}+\cdots+b_{e-1}p^{e-1}} \end{aligned}$$

For $x_i = 0$, we have:

$$x_0 = g^{b_0 p^{e-1}} + g^{b_1 p^e} + g^{b_2 p^{e+1}} + \cdots .$$

Thus, we can easily see that the first term $g^{b_0 p^{e-1}} \in H$. All other terms are 1 because $\left(g^{p^e}\right)^{b_1} = 1$. This explains that $x_i \in H$, which proves that the algorithm works correctly.

The algorithm works in e steps and uses the Repeated Squaring Algorithm (RSA) in each step. Therefore, the time complexity of steps 1, 3, and 5 is $O(e \log p)$. Since the algorithm finishes in e steps, the total time complexity is $O(e^2 \log p)$. □

4.7 Index calculus

This algorithm solves the discrete logarithm problem in $\mathbb{Z}_p^*$, where p is a prime number. The group $\mathbb{Z}_p^*$ is a cyclic group with generator g and order $d = p - 1$. The problem we want to solve is $a = \operatorname{dlog}_g x$, where $x \in \mathbb{Z}_p^*$.

The first step is to choose a factor base $B = \{p_1, \ldots, p_h\}$, consisting of prime numbers up to a certain bound. Initially, we do not work with x. Instead, we randomly select $e \in \mathbb{Z}_{p-1}$, compute g^e, and check if the resulting number is B-smooth. If it is B-smooth, we have the relation:

$$g^e = p_1^{\alpha_1} \cdots p_h^{\alpha_h} \text{in} \mathbb{Z}_p^*,$$
$$e = \alpha_1 \operatorname{dlog}_g p_1 + \cdots + \alpha_h \operatorname{dlog}_g p_h \text{ in } \mathbb{Z}_{p-1}.$$

We repeat this process until we gather enough linear equations. Typically, $h + 20$ such relations are sufficient. Using these relations, we compute $\operatorname{dlog}_g p_1, \ldots, \operatorname{dlog}_g p_h$ by solving a matrix equation. After solving, we now know the values of $\operatorname{dlog}_g p_1, \ldots, \operatorname{dlog}_g p_h$.

Next, we involve x in the computation. We randomly select $e \in \mathbb{Z}_{p-1}$, compute xg^e, and check if it is B-smooth. If it is B-smooth, we have:

$$xg^e = p_1^{\beta_1} \cdots p_h^{\beta_h} \text{ in } \mathbb{Z}_p.$$

Then:

$$\operatorname{dlog}_g x = -e + \beta_1 \operatorname{dlog}_g p_1 + \cdots + \beta_h \operatorname{dlog}_g p_h \text{ in } \mathbb{Z}_{p-1}.$$

Remark 2. The time complexity is $\exp((1 + o(1))\sqrt{n \log n})$ for an n-bit p, similar to Dixon's random squares method. This includes the factorization of $p - 1$.

5 The Extended Euclidean Algorithm

The Euclidean Algorithm finds the greatest common divisor of two numbers through a series of division operations between them. The Extended Euclidean Algorithm , however, not only computes the GCD but also determines the coefficients in the linear equation that represents the GCD. These coefficients are known as the Bzout coefficients and are expressed as:

$$a \cdot x + b \cdot y = \gcd(a, b)$$

According to this equation, the GCD of the numbers a and b can be written in the form $a \cdot x + b \cdot y$, where x and y are the coefficients that satisfy this representation of the GCD.

Algorithm 5. Extended Euclidean Algorithm
Input:
$a, b \in R$, where R is a Euclidean domain.
Output:
$\ell \in \mathbb{N}, r_i, s_i, t_i \in R$ for $0 \leq i \leq \ell + 1$, and $q_i \in R$ for $1 \leq i \leq \ell$, as computed below.

1. $r_0 \leftarrow a,\ s_0 \leftarrow 1,\ t_0 \leftarrow 0,$
   $r_1 \leftarrow b,\ s_1 \leftarrow 0,\ t_1 \leftarrow 1.$
2. $i \leftarrow 1.$
3. While $r_i \neq 0$ **do**
   - $q_i \leftarrow r_{i-1}$ quo r_i,
   - $r_{i+1} \leftarrow r_{i-1} - q_i r_i$,
   - $s_{i+1} \leftarrow s_{i-1} - q_i s_i$,
   - $t_{i+1} \leftarrow t_{i-1} - q_i t_i$,
   - $i \leftarrow i + 1.$
4. $\ell \leftarrow i - 1.$
5. If r_ℓ is a unit, replace (r_ℓ, s_ℓ, t_ℓ) by $(1, s_\ell / r_\ell, t_\ell / r_\ell)$.
6. Return ℓ, r_i, s_i, t_i for $0 \leq i \leq \ell + 1$, and q_i for $1 \leq i \leq \ell$.

The Extended Euclidean Algorithm, for n-bit inputs, does $O(n)$ work in each step and runs for $O(n)$ steps. So, the total time is $O(n^2)$.

5.1 Modular inverses

Using the Extended Euclidean Algorithm, we can compute Bzout coefficients, which allow us to find the modular inverse.

Algorithm 6. Modular inverse in a Euclidean domain R.
Input:
Elements $a, b \in R$, with $b \neq 0$.
Output:
Either "a is not invertible" or an inverse s of a in $R/(b)$.

1. Call Algorithm 5 to obtain a representation $c = sy + tb$ of $c = \gcd(a, b)$.
2. If $c = 1$, then return s.
3. Else, return "a is not invertible".

6 Efficient exponentiation

Since the classical method of exponentiation is inefficient, we use the more efficient repeated squaring algorithm in our computations.

Algorithm 7. Repeated Squaring Algorithm
Input:
A group G, a base $x \in G$, and an exponent $e \in \mathbb{Z}$ with $1 \leq e < \#G$.
Output:
$x^e \in G$.

1. Let $\sum_{0 \leq i < n} e_i 2^i$ be the binary representation of e with $e_0, \ldots, e_{n-1} \in \{0, 1\}$, $e_{n-1} = 1$, and n its bit length.
2. $y \leftarrow x$.
3. For i from $n - 2$ downto 0, do:
4. $y \leftarrow y^2$.
5. If $e_i = 1$, then $y \leftarrow y \cdot x$.
6. Return y.

The algorithm performs a total of $(n-1)$ group operations, and in the worst case, it requires $2n - 2$ operations, which corresponds to a cost of $O(n)$. Additionally, the exponentiation operation in the algorithm has a cost of $O(n^2)$, and the algorithm performs it $O(n)$ times, resulting in a total cost of $O(n^3)$ bit operations.

7 The Chinese Remainder Algorithm

The Chinese Remainder Theorem states that for any integers $n_1, n_2, n_3, \ldots, n_k$ that are pairwise coprime, and for any remainders $a_1, a_2, \ldots, a_k$, there exists an integer x modulo $N = n_1 n_2 n_3 \ldots n_k$ that satisfies:

$$x \equiv a_i \pmod{n_i}, \quad \text{for each } i.$$

Algorithm 8. Chinese Remainder Algorithm
Input:
Moduli $q_1, \ldots, q_r$ and residues $a_1, \ldots, a_r$ with all $a_i \in \mathbb{Z}$.
Output:
A simultaneous solution $x \in \mathbb{Z}$ to the congruences $x = a_i \pmod{q_i}$ or the error message "the q_i are not pairwise coprime".

1. $\ell_i \leftarrow \prod_{j \neq i} q_j$ for all i.
2. Try to compute $y_i = \ell_i^{-1}$ in $\mathbb{Z}_{q_i}$ for all i, using Algorithm 5. If it outputs "ℓ_i is not invertible" for some i, then return "the q_i are not pairwise coprime".
3. $m \leftarrow q_1 \cdot q_2 \cdots q_r$.
4. $x \leftarrow \sum_{i=0}^{r} a_i y_i \ell_i \pmod{m}$.
5. Return x.

Theorem 8. *The cost of the Algorithm 7 is $O(n^2)$ bit operations.*

Proof. To calculate $y_i = \ell_i^{-1}$, the Extended Euclidean Algorithm (EEA) is used, which requires $O(n^2)$ bit operations. Since all other costs are smaller, the total cost of the algorithm is $O(n^2)$ bit operations.

□

References

[1] Joachim von zur Gathen, *CryptoSchool*, Springer, 2015.