GROUP CRYPTOGRAPHY AND DISCRETE LOGARITHMS

Enes KALECİ

Istinye University

June 18, 2025

Table of Contents

Introduction:What is Group Cryptography and discrete logarithms?

Cryptography is the science of keeping information safe using mathematics. One important part of this is called **group cryptography**, which uses special sets of numbers and operations, known as **groups**, to build secure systems.
A key idea in group cryptography is the **Discrete Logarithm Problem (DLP)**. This problem asks: if you know a number g and another number h, can you find the number x such that $g^x = h$? This is very hard to solve, and that's why it helps keep data secure.
In this presentation, we will explain how group cryptography works and how to solve the discrete logarithm problem.

The Diffie-Hellman key exchange

Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to generate a common secret key over an insecure channel without sharing it. 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 Diffie-Hellman Key Exchange

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 $k_B = A^b$.

Groups for cryptography

The security depends on the shared secret key. To break the system someone must find the secret key from the public key. This is called DLG. As we said before this problem is very hard, but there are some know algorthims that try to solve it.

Groups for cryptography

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.

The ElGamal encryption scheme

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

Protocol 1

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 $\mathrm{pk} = B \leftarrow g^b \in G$.

The ElGamal encryption scheme

Protocol 1

Encryption:

- **Input:** Plaintext $x \in G$.
- **Output:** Ciphertext $\mathrm{enc}_{\mathrm{pk}}(x) \in G^2$.
- 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 $\mathrm{enc}_{\mathrm{pk}}(x) = (y, A)$.

Decryption:

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

The ElGamal encryption scheme

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.

Baby-Step Giant-Step Algorithm

In this section, we will examine an algorithm used for DLG. 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

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 $\mathbb{Z}_d$.
```

Baby-Step Giant-Step Algorithm

Theorem 1

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 1

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.

Baby-Step Giant-Step Algorithm

continued

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.

Baby-Step and Giant-Step algorithms, the calculations are performed using the repeated squaring algorithm. So, each computation is carried out in $O(n^3)$ bit operations.

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 $x = g^{j-i}$.

Algorithm 2

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 $\mathbb{Z}_d$.
```

The Birthday attack

Theorem 2

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 $O(\sqrt{d}\log d)$.

Chinese Distribution Of Discrete Logarithms

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.

$$\begin{tikzcd}
 & \mathbb{Z}_d \arrow[dl, "\mu_{e_1}"'] \arrow[dr, "\mu_{e_2}"] \arrow[dd, shift right, "\mathrm{dexp}_g"'] & \\
\mathbb{Z}_{q_1} \arrow[dd, shift right, "\mathrm{dexp}_{g_1}"'] & & \mathbb{Z}_{q_2} \arrow[dd, shift right, "\mathrm{dexp}_{g_2}"'] \\
 & G \arrow[uu, shift right, "\mathrm{dlog}_g"'] \arrow[dl, "\pi_{e_1}"] \arrow[dr, "\pi_{e_2}"'] & \\
\pi_{e_1}(G) \arrow[uu, shift right, "\mathrm{dlog}_{g_1}"'] & & \pi_{e_2}(G) \arrow[uu, shift right, "\mathrm{dlog}_{g_2}"']
\end{tikzcd}$$

Chinese Distribution Of Discrete Logarithms

Suppose that $d = q_1 q_2$ with coprime q_1 and q_2, and that $a_i = \mathrm{dlog}_{g_i} x_i$ 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 $i = 1, 2$.

Chinese Distribution Of Discrete Logarithms

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 1 and 2:
    1. Compute $g_i = g^{d/q_i}$ and $x_i = x^{d/q_i}$, using the Repeated Squaring Algorithm
    2. 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 $i \leq r$.

Chinese Distribution Of Discrete Logarithms

Theorem

Let G be a cyclic group of n-bit order d. Then Algorithm 3 computes a discrete logarithm in G at the following cost:

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

(ii) $O(n^3)$ 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.

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 $H = \langle g^{p^{e-1}} \rangle$. we solve the discrete logarithm problem using the elements of the subgroup. The discrete logarithms we solve allow us to write the desired element $a = \mathrm{dlog}(x)$ in base p as

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

The Pohlig-Hellman algorithm

Algorithm: 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: $a = \mathrm{dlog}_G(x)$, 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:
    1. $x_i \leftarrow (x \cdot y_{i-1}^{-1})^{p^{e-i-1}}$
    2. Compute $a_i \leftarrow \mathrm{dlog}_h(x_i)$.
    3. Update $y_i \leftarrow y_{i-1} \cdot g^{-a_i p^i}$.
3. Return $a = a_{e-1}p^{e-1} + \cdots + a_0$.
```

The Pohlig-Hellman algorithm

Theorem

The algorithm correctly computes $\mathrm{dlog}_g x$. 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 $\mathrm{dlog}_g x$ 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.

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

$$= g^{b_ip^{e-i-1}+b_{i+1}p^{e-i}+b_{i+2}p^{e-i+1}+\cdots+b_{e-1}p^{e-1}}$$

For $x_i = 0$, we have:

The Pohlig-Hellman algorithm

continued

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

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, 2, and 3 is $O(e \log p)$.

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:

$$g^e = p_1^{\alpha_1} \cdots p_h^{\alpha_h} \, \text{in} \mathbb{Z}_p^*,$$

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

Index Calculus

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

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

We can solve the equation by substituting the expressions $\mathrm{dlog}_g p_1, \ldots, \mathrm{dlog}_g p_h$ that we calculated earlier. Thus we have solved the discrete logarithm.

Index Calculus

Remark

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 $p-1$.