Discrete Logarithm (DL)

Cryptography - CS 411 / CS 507

Erkay Savaş

Department of Computer Science and Engineering
Sabancı University

November 7, 2019

Cryptosystems Based on DL

- DL is the underlying hard problem for
  - Diffie-Hellman key exchange
  - DSA (Digital signature algorithm)
  - ElGamal encryption/digital signature algorithm
  - Elliptic curve cryptosystems
- DL is defined over finite groups

Discrete Logarithm Problem

- Let p be a prime and α and β be nonzero integers in $\mathbb{Z}_p$ and suppose
$$\beta = \alpha^x \bmod p.$$
- The problem of finding x is called the discrete logarithm problem.
- We can denote it as
$$x = \log_\alpha \beta$$
  - Often, α is a primitive root $\bmod p$
- Reminder: $\mathbb{Z}_p$ is a finite field $0, 1, \ldots, p-1$
- Reminder 2: $\mathbb{Z}_p^*$ is a cyclic finite group $1, \ldots, p-1$

Example: Discrete log

- Example:
  - Let $p = 11$, $\alpha = 2$, and $\beta = 9$.
  - By exhaustive search,

$\boldsymbol{i}$	0	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{\alpha}^i$	1	2	4	8	5	10	9	7	3	6	1

- $\log_2 9 \bmod 11 = 6$.
- The discrete log behaves in many ways like the usual logarithm.
- For instance, if α is primitive root of $\pmod{p}$, then $\log_\alpha(\beta_1\beta_2) \equiv \log_\alpha(\beta_1) + \log_\alpha(\beta_2) \pmod{p-1}$

Computing Discrete log

- When p is small, it is easy to compute discrete logarithms by exhaustive search.
- However, it is a hard problem to solve for primes p with more than 200 digits.
- It is as hard as the integer factorization problem.
- One-way function.
  - It is easy to compute modular exponentiation
  - But, it is hard to compute the inverse operation of the modular exponentiation, i.e. discrete log.

Computing Discrete Log

- α is usually a primitive root of $\bmod p$.
- $\alpha^{p-1} \equiv 1 \bmod p$. This implies that
$$\alpha^{m_1} \equiv \alpha^{m_2} \bmod p \Leftrightarrow {\color{red}?}$$
- Assume that
$$\beta = \alpha^x \bmod p, \qquad 0 \leq x \leq p-1$$
- It is difficult to find x.
- However, it is easy to find out if x is even or odd.
$$\alpha^{p-1} \equiv 1 \bmod p \rightarrow (\alpha^{(p-1)/2})^2 \equiv 1 \bmod p$$
$$\alpha^{(p-1)/2} \equiv \pm 1 \bmod p.$$

Computing Discrete Log

- But, we know $p-1$ is the smallest integer which yields $+1$, thus
  $$\alpha^{(p-1)/2} \equiv -1 \bmod p.$$
  recall α is primitive
- Starting with $\beta = \alpha^x \bmod p$, raise both sides to the $(p-1)/2$ power to obtain
  $$\beta^{(p-1)/2} \equiv \alpha^{x(p-1)/2} \bmod p \equiv (-1)^x \bmod p.$$
- Therefore, if $\beta^{(p-1)/2} \equiv 1 \bmod p$, then x is even; otherwise x is odd.

Discrete Log Algorithms

- Shanks's algorithm (baby-step giant-step) :
  - DL in $\mathbb{Z}_p^*$: $(p)^{1/2}$ steps.
  - Minimum security requirement: $(p-1) > 2^{224}$
- Pohlig-Hellman algorithm:
  - $|\mathbb{Z}_p^*| = p_1 p_2 p_3 \dots p_j$
  - complexity: either $O((p-1)^{1/2})$ or $O(\sum_i e_i(\log_2(p-1) + p_i^{1/2})$
  - Minimum security requirement: $(p-1) > 2^{224}$
- Index-calculus method:
  - Applies only to $\mathbb{Z}_p$ and $GF(p^k)$
  - complexity: $O(e^{(1+O(1)\sqrt{\ln(p)\ln(\ln(p))})})$
  - Minimum security requirement in $\mathbb{Z}_p^*$: $(p-1) > 2^{2048}$

Diffie-Hellman Key Exchange

- Proposed in 1976 by Diffie-Hellman
- Used in many protocols
- Can use DL problem on any finite group

- Protocol:
  - Setup phase:
    1. Find a large prime p
    2. Find a primitive element α in $\mathbb{Z}_p^*$ or in a subgroup of $\mathbb{Z}_p^*$.

Diffie-Hellman Key Exchange

Alice

1. Picks a random s_A
   $2 \leq s_A < p - 1$
2. Computes $p_A = \alpha^{s_A} \bmod p$
3. Sends p_A to Bob
4. Computes k_{BA}
   $k_{BA} = (p_B)^{s_A} \bmod p$
   $k_{BA} = (\alpha^{s_B})^{s_A} \bmod p$

Bob

1. Picks a random s_B
   $2 \leq s_B < p - 1$
2. Computes $p_B = \alpha^{s_B} \bmod p$
3. Sends p_B to Bob
4. Computes k_{AB}
   $k_{AB} = (p_A)^{s_B} \bmod p$
   $k_{AB} = (\alpha^{s_A})^{s_B} \bmod p$

Session key : $k = k_{BA} = k_{AB} = \alpha^{s_A s_B} \bmod p$

Security of Diffie-Hellman

- What an adversary observes are
  - p, α, p_A, p_B
  - he needs to know either s_A or s_B
- Problem 1: given p, α, p_A find s_A
  - $s_A = \log_\alpha p_A \pmod{p-1}$
  - discrete logarithm problem
- Problem 2: given p, α, p_B find s_B
  - $s_B = \log_\alpha p_B \pmod{p-1}$
  - discrete logarithm problem

Formalism

- "Computational Diffie-Hellman Problem"
  - p is prime and α is a generator in ${\mathbb{Z}_p}^*$
  - given $\alpha^x \bmod p$ and $\alpha^y \bmod p$
    - find $\alpha^{xy} \bmod p$
- Decision Diffie-Hellman Problem
  - p is prime and α is a generator in ${\mathbb{Z}_p}^*$
  - given $\alpha^x \bmod p$ and $\alpha^y \bmod p$, distinguishing between
    - $(\alpha, \alpha^x, \alpha^y, \alpha^{xy})$ and $(\alpha, \alpha^x, \alpha^y, \alpha^z)$

The ElGamal PKC

- Based on the difficulty of discrete logarithm, invented by Taher ElGamal in 1985.
- Alice wants to send a message m to Bob.
- Bob uses a large prime p and a primitive root α.
  - Assume m is an integer $0 < m < p$.
- Bob also picks a secret integer b and computes
  - $\beta = \alpha^b \bmod p$.
- $\{p, \alpha\}$ are public parameters
- $\{\beta\}$ is Bob's public key.
- $\{b\}$ is his private key

The ElGamal PKC: Protocol

<u>Alice</u>

Chooses a secret integer
$k < p - 1$ at random
Computes $r = \alpha^k \bmod p$
Computes $t = \beta^k \times m \bmod p$
Sends (r, t) to Bob.

<u>Bob</u>

Computes $t \times r^{-b} \bmod p = m$

This works since
$t \times r^{-b} \equiv \beta^k \times m \times (\alpha^k)^{-b} \equiv \alpha^{kb} \times m \times \alpha^{-kb}$

Security of ElGamal PKC

- b must be kept secret.
- k is a random integer,
  - β^k is also a random nonzero integer $\mathrm{mod} p$.
  - Therefore, $t = \beta^k \times m \bmod p$ is the message m multiplied by a random integer.
  - t is also a random integer
- If Eve knows k,
  - she can calculate $t \times \beta^{-k} \bmod p = m$.
  - k must be secret
- Knowing r does not help by itself.

Security of ElGamal PKC

- A different random k must be used for each message m.
  - Assume Alice uses the same k for two different messages m_1 and m_2,
  - the corresponding ciphertexts are (r, t_1) and (r, t_2).
  - If Eve finds out the plaintext m_1 (i.e., known plaintext attack), she can also determine m_2 as follows
  - $t_1/m_1 \equiv \beta^k \equiv t_2/m_2 \bmod p \rightarrow m_2 \equiv (t_2 m_1)/t1$

Efficient Implementation of ElGamal

- We have two primes
  - p: large; q: relatively smaller (e.g., 2048-bit and 224-bit primes, respectively)
  - $q|(p-1)$
- G_q: a subgroup of $\mathbb{Z}_p{}^*$
  - g is a generator of G_q.
- Example
  - $q = 5$, $p = 31$
  - $g = 2$
  - $2^0 \ (\mathrm{mod}\ 3)1 = 1$, $2^1 \ (\mathrm{mod}\ 3)1 = 2$, $2^2 \ (\mathrm{mod}\ 3)1 = 4$, $2^3 \ (\mathrm{mod}\ 3)1 = 8$, $2^4 \ (\mathrm{mod}\ 3)1 = 16$, $2^5 \ (\mathrm{mod}\ 3)1 = 1$
  - $G_5 = \{1, 2, 4, 8, 16\}$

Efficient Implementation of ElGamal

- Key generation
  - s: private key $1 < s < q - 1$
  - h: public key $h = g^s \pmod{p}$
- Encryption
  - k random key $1 < k < q - 1$
  - $r = g^k \pmod{p}$
  - $t = h^k m \pmod{p}$
  - (r, t): ciphertext
- Decryption
  - $tr^{-s} \pmod{p}$

Key Generation Algorithm (2048, 224)

1. Generate a random q such that $2^{223} < q < 2^{224}$
2. Choose a random integer k such that $2^{1823} \leq k < 2^{1824}$
3. $p \leftarrow kq + 1$
4. If p is not prime then go to Step 2
5. Choose a random element $\alpha \in \mathbb{Z}_p^*$
6. $g \leftarrow \alpha^{(p-1)/q} \bmod p$
7. If $g = 1$ then go to Step 5