MH4311 Cryptography

Lecture 14
Public Key Encryption
Part 2. ElGamal

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
- Public key encryption
  - RSA
  - **ElGamal**
    - **Specification**
    - **Implementation**
    - **Security**
- Digital signature
- Key establishment and management
- Introduction to other cryptographic topics

Recommended Reading

- CTP Chapter 6
- HAC Section 8.4
- Wikipedia
  - ElGamal Encryption
    http://en.wikipedia.org/wiki/ElGamal_encryption
  - Discrete Logarithm
    http://en.wikipedia.org/wiki/Discrete_logarithm

ElGamal Public Key Cryptosystem

- Based on the difficulty of discrete logarithm
- Invented by Taher ElGamal, 1985

ElGamal Cryptosystem

- **ElGamal encryption**
- ElGamal digital signature (learn it later)

Cyclic Group and Discrete Logarithm

Multiplicative group modulo prime *p*

- Multiplicative group modulo prime *p*
  - Denoted as Z_p^*
  - Elements: 1, 2, 3, 4, …, *p*-1
  - Operation: multiplication modulo *p*

Cyclic group

- A group G is called cyclic if there exists an element g in G such that

$$G = \{ g^i \mid i \text{ is an integer} \},$$

where g is a generator of G.

Cyclic group

- A multiplicative group Z_p^* is cyclic (*p* is prime)
- There are many types of cyclic groups:
  - The set of integers modulo *n*, with the operation of addition modulo *n*, forms a cyclic group. Every element which is coprime to *n* is a generator of this group.
    - The above additive group is denoted as Z_n^+
    - Example: In the group Z_6^+, there are 6 elements: 0, 1, 2, 3, 4, 5. 1 and 5 are the generators of this group. In this group, 1 = 1, 1+1 = 2, 1+1+1 = 3, 1+1+1+1 = 4, 1+1+1+1+1 = 5, 1+1+1+1+1+1 = 0.
      5 = 5, 5+5 = 4, 5+5+5 = 3, 5+5+5+5 = 2, 5+5+5+5=5 = 1, 5+5+5+5+5+5 = 0.

Example of generator

- Example of generator of cyclic group
  - 2 is a generator of Z_5^* since in this group,

    $$2^1 = 2,\ 2^2 = 4,\ 2^3 = 3,\ 2^4 = 1.$$

    3 is a generator of Z_5^* since in this group,

    $$3^1 = 3,\ 3^2 = 4,\ 3^3 = 2,\ 3^4 = 1.$$

    4 is NOT a generator of Z_5^* since in this group,

    $$4^1 = 4,\ 4^2 = 1,\ 4^3 = 4,\ 4^4 = 1.$$

Example of generator

- Example of generator of cyclic group
  - 2 is NOT a generator of Z_7^* since in this group,

    $2^1 = 2,\ 2^2 = 4,\ 2^3 = 1,\ 2^4 = 2,\ 2^5 = 4,\ 2^6 = 1.$

    3 is a generator of Z_7^* since in this group,

    $3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1.$

    4 is NOT a generator of Z_7^* since in this group,

    $4^1 = 4,\ 4^2 = 2,\ 4^3 = 1,\ 4^4 = 4,\ 4^5 = 2,\ 4^6 = 1.$

    5 is a generator of Z_7^* since in this group,

    $5^1 = 5,\ 5^2 = 4,\ 5^3 = 6,\ 5^4 = 2,\ 5^5 = 3,\ 5^6 = 1.$

    6 is NOT a generator of Z_7^* since in this group,

    $6^1 = 6,\ 6^2 = 1,\ 6^3 = 6,\ 6^4 = 1,\ 6^5 = 6,\ 6^6 = 1.$

Discrete logarithm

- Discrete logarithm (here for Z_p^*)

  1. p is a prime

  2. Let g be a generator of the multiplicative cyclic group Z_p^*

  3. Given $y \in Z_p^*$, to find x satisfying $g^x \bmod p = y$

- The above discrete logarithm problem is hard to solve for large primes (for example, 2048-bit primes)

ElGamal Encryption

ElGamal Encryption: Specification

- Key generation

1. Generate a large random prime p
2. Find a generator g of the multiplicative group Z_p^* .
3. Select a random secret integer x $(0 < a < p)$, and compute $y = g^x \bmod p$

Public key: (p, g, y)

Private key: x

ElGamal Encryption : Specification

- Encryption (Plaintext m: $0 < m < p$)

  1. Select a random per-message (one-time) secret integer k
  2. Compute

$$c_1 = g^k \bmod p$$

$$c_2 = m \cdot y^k \bmod p$$

Ciphertext: $C = (c_1, c_2)$

- Decryption

$$m = c_1^{-x} \cdot c_2 \bmod p$$

ElGamal Encryption: Specification

- Decryption process recovers the message:

$$
\begin{aligned}
& c_1^{-x} \cdot c_2 \bmod p \\
&= (g^k)^{-x} \cdot (m \cdot y^k) \bmod p \\
&= (g^x)^{-k} \cdot (m \cdot y^k) \bmod p \\
&= y^{-k} \cdot (m \cdot y^k) \bmod p \\
&= m
\end{aligned}
$$

ElGamal Encryption: Example

- Example (Toy ElGamal Encryption)

Key generation :

1. Select $p = 2357$, and a generator $g = 2$ of Z_{2357}^{*}
2. Private key $x = 1751$
3. $y = g^{x} \bmod p = 2^{1751} \bmod 2357 = 1185$

Public Key: $(p, g, y) = (2357, 2, 1185)$

Private Key: $x = 1751$

Encryption: $m = 2035$

1. Select a random integer $k = 1520$
2. $c_1 = g^{k} \bmod p = 2^{1520} \bmod 2357 = 1430$

   $c_2 = m \cdot y^{k} \bmod p = 2035 \times 1185^{1520} \bmod 2357 = 697$

Decryption :

$$m = c_1^{-x} \cdot c_2 = 1430^{-1751} \times 697 \bmod 2357 = 2035$$

ElGamal Encryption: Implementation

- To find a generator of the cyclic group Z_p^*

1. Factorize $p-1 = p_1^{e_1} p_2^{e_2} p_3^{e_3} \cdots p_t^{e_t}$

2. Choose a random integer β, if $\beta^{\frac{p-1}{p_i}} \bmod p \neq 1$ for $1 \leq i \leq t$, then β is a generator

   Remarks: if $\beta^{\frac{p-1}{p_i}} \bmod p = 1$, then $\beta^j \bmod p$ generates only $\dfrac{p-1}{p_i}$ elements since $\beta^j \equiv \beta^{j+\frac{p-1}{p_i}} \pmod{p}$

ElGamal Encryption: Implementation

- In order to find a generator of the cyclic group Z_p^*, p-1 should be factorized.
- It is computationally impossible to generate a random 2048-bit prime p, then factorize p-1
  - Solution: In practice, we generate a 2047-bit prime number p', let $p = 2p' + 1$, then test whether p is prime. Repeat this process until a 2048-bit prime p is found. The factors of p-1 are known.
  - A prime number p is called **safe prime** if $p = 2p' + 1$, where p' is a prime. Safe primes are widely used in cryptographic algorithms, such as RSA and ElGamal.

ElGamal Encryption: Security

- In ElGamal encryption, the prime p and the generator g can be shared by many users
  - In RSA, the modulus n should not be shared
- The one-time secret k used in ElGamal encryption should be a secret random value
- The security of ElGamal highly depends on the difficulty of solving the discrete log problem
  - Many discrete logarithm algorithms were developed to solve the discrete log problem.

Discrete Logarithm Algorithms

Discrete Logarithm Algorithms

- Shank's baby-step giant-step algorithm
- Pollard's rho algorithm for discrete logarithm
- Pohlig-Hellman algorithm
- Index calculus algorithm (only for Z_p^*)
- ……

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ generic

Shank's baby-step giant-step algorithm

$g^x \equiv b \pmod p$

$n=p-1$

- Basic idea:
  - Let $t = \lceil \sqrt{n} \rceil$, then x can be written as: $x = i \times t + j$, where $i < t,\ j < t$.

    It means that $g^x \bmod p = g^{i\times t + j} \bmod p = b$

    $$\Rightarrow g^{i\times t} \bmod p = b \times g^{-j} \bmod p$$

- Algorithm
  - Compute a table T1 with elements $(i, g^{i\times t} \bmod p)$ for all the $i < t$;
  - Compute a table T2 with elements $(j, b \times g^{-j} \bmod p)$ for all the $j < t$;
  - Compare T1 and T2, if $g^{i\times t} \bmod p = b \times g^{-j} \bmod p$,

    we know that $x = i \times t + j$

Shank's baby-step giant-step algorithm

- Complexity:

$$O(n^{0.5})\ \text{computations},\ O(n^{0.5})\ \text{memory}$$

Shank's baby-step giant-step algorithm

- Example: To find x satisfying $2^x \bmod 19 = 15$
  (x is a positive integer less than 19)

$G = Z^*_{19} = \{1, 2, \ldots, 18\}$
$g = 2$, $g^{-1} = 10$, $n = p\text{-}1 = 18$,
$t = 5$, $g^t \bmod 19 = 13$, $b = 15$

T1: $(i, g^{i\times t} \bmod 19)$	T2: $(j, b\times g^{-j} \bmod 19)$
(0, 1)	(0, 15)
(1, 13)	(**1**, **17**)
(**2**, **17**)	(2, 18)
(3, 12)	(3, 9)
(4, 4)	(4, 14)

$j = 1$
$i = 2$
$\boldsymbol{x = i\times t + j = 11}$

Pollard's rho algorithm for discrete logarithm

$g^x \equiv b \pmod{p}$
$n{=}p$-1

- Basic Idea: Birthday attack
  - Compute (g^u mod p) for $n^{0.5}$ random values of u
  - Compute (b^v mod p) for $n^{0.5}$ random values of v
  - Due to birthday paradox, we may find one pair (u,v) satisfying (g^u mod p) = (b^v mod p)
    - It means that $g^u \equiv g^{xv} \pmod{p}$, i.e., $u \equiv xv \pmod{p\text{-}1}$

      => find x successfully
  - Complexity
    - $O(n^{0.5})$ computation, $O(n^{0.5})$ memory

Pollard's rho algorithm for discrete logarithm

$g^x \equiv b \pmod{p}$
$n = p$-1

- Pollard's rho algorithm*
  - Try to reduce the memory in birthday attack
  - Idea:
    - Define $x_{i+1} = f(x_i) = g^{f_1(x_i)} b^{f_2(x_i)}$
      ("random" function f is chosen so that it is non-injective)
    - Use turtoise and hare algorithm to find $x_{i+1} = x_{2(i+1)}$,
      then find the period u, and the starting point of the cycle (denoted as x_a).
    - Then we know that $x_a = x_{u+a}$, i.e., $g^{f_1(x_{a-1})} b^{f_2(x_{a-1})} \equiv g^{f_1(x_{u+a-1})} b^{f_2(x_{u+a-1})} \bmod p$
      $\Rightarrow f_1(x_{a-1}) - f_1(x_{u+a-1}) \equiv x(-f_2(x_{a-1}) + f_2(x_{u+a-1})) \bmod p-1$

Pollard's rho algorithm for discrete logarithm*

$g^x \equiv b \pmod{p}$
$n = p - 1$

- Pollard's rho algorithm*
  - The function f is defined as follows:

Let G be a cyclic group of order n,

partition $G = G_0 \cup G_1 \cup G_2$, where G_i are almost the same size

$$f(x_i) = \begin{cases} bx_i & x_i \in G_0 \\ x_i^2 & x_i \in G_1 \\ gx_i & x_i \in G_2 \end{cases}$$

Pohlig-Hellman Algorithm

$g^x \equiv b \pmod{p}$
$n = p - 1$

Basic idea:

Suppose that $n = p - 1 = p_1 p_2 p_3 \cdots p_t$,

try to find $x \bmod p_i$ first,

then find x using the Chinese Remainder Theorem

Pohlig-Hellman Algorithm

$g^x \equiv b \pmod{p}$
$n=p-1$

Let $x = u_i \cdot p_i + v_i$, where $v_i = x \bmod p_i$

$$g^x \equiv b \pmod{p}$$

$$(g^x)^{n/p_i} \equiv b^{n/p_i} \pmod{p}$$

$$(g^{u_i \cdot p_i + v_i})^{n/p_i} \equiv b^{n/p_i} \pmod{p}$$

$$(g^{v_i})^{n/p_i} \equiv b^{n/p_i} \pmod{p}$$

$$(g^{n/p_i})^{v_i} \equiv b^{n/p_i} \pmod{p} \qquad (1)$$

Let $g' = g^{n/p_i}, b' = b^{n/p_i}$, (1) becomes

$$(g')^{v_i} \equiv b' \pmod{p} \qquad (2)$$

In (2), v_i can be found using baby-step giant-step algorithm (complexity $O(\sqrt{p_i})$)

=> For high security, *p*-1 should have a large factor

Pohlig-Hellman Algorithm

- In the previous slide, we assumed that each factor of p-1 appears only once
- If p_i^{k+1} (k is a positive integer) is a factor of p-1, and if p_i^{k+1} is too large, a minor modification is needed.

1) $x = ui \times p_i + vi$ mod pi

   $\Rightarrow$ find v_i using the method in the previous slide

2) $x = w_i \times p_i^2 + u_i \times pi + vi\ mod\ pi$

   $\Rightarrow$ find u_i from $(g^x)^{n/p_i^2} \equiv b^{n/p_i^2} \pmod{p}$

3) ….

4) Eventually, we find the value of ($x\ mod\ p_i^{k+1}$)

Pohlig-Hellman Algorithm

- Example: To find x satisfying $2^x \bmod 29 = 18$

  (x is a positive integer less than 29)

  Solution: $n = p - 1 = 28 = 4 \times 7$

  1) Let $x = u_1 \times 4 + v_1$;

$$(2^{u_1 \times 4 + v_1})^{28/4} \equiv 18^{28/4} (\text{mod } 29)$$

$$2^{7 \times v_1} \equiv 18^7 (\text{mod } 29)$$

$$12^{\text{v1}} \equiv 17 (\text{mod } 29)$$

$$v_1 = 3$$

(the value of v_1 is small here, we try 1, 2, 3)

Pohlig-Hellman Algorithm

2) Let $x = u_2 \times 7 + v_2$;

$$(2^{u_2 \times 7 + v_2})^{28/7} \equiv 18^{28/7} \pmod{29}$$

$$2^{4 \times v_2} \equiv 18^4 \pmod{29}$$

$$16^{v_2} \equiv 25 \pmod{29}$$

$$v_2 = 4$$

(the value of v_2 is small here, we try 1, 2, ..., 6;

you may also try the baby-step giant-step algorithm)

Pohlig-Hellman Algorithm

3) $x \equiv 3 \pmod{4}$;
$x \equiv 4 \pmod{7}$;

Apply the Chinese Remainder Theorem,
$x \equiv 11 \pmod{28}$,
so we have $x = 11$.

Index calculus algorithm

$g^x \equiv b \bmod p$
$n = p$-1

- A powerful algorithm for solving **integer** discrete logarithm
  - Exploit the property of "smooth integers"

Precomputation :

Step 1. Determine a value B (details not given here)

Denote those prime numbers less than B as $\{p_1, p_2, p_3, \cdots, p_t\}$

Step 2. Find t different x_i so that each $g^{x_i} \bmod p$ is B-smooth :

$$g^{x_i} \bmod p = p_1^{e_{i,1}} p_2^{e_{i,2}} p_3^{e_{i,3}} \cdots p_t^{e_{i,t}}$$

i.e., $x_i \equiv e_{i,1} \log_g p_1 + e_{i,2} \log_g p_2 + \cdots + e_{i,t} \log_g p_t \pmod{p-1}$

Step 3. Solve those t linear equations to determine the values of $\log_g p_i$

To find the value of $x = \log_g b$:

Try different values of s, find an s so that $g^s \cdot b \bmod p$ is B-smooth :

$$g^s \cdot b \bmod p = p_1^{f_1} p_2^{f_2} p_3^{f_3} \cdots p_t^{f_t},$$

then $s + x \equiv f_1 \log_g p_1 + f_2 \log_g p_2 + \cdots + f_t \log_g p_t \pmod{p-1}$

Index calculus algorithm

- Example:

  $p = 10007$, $g = 5$, $5^x \bmod p = 9451$

Precomputation:

Choose B=7, then the prime numbers not larger than 7 are {2, 3, 5, 7}.
We know that $\log_5 5=1$.

$5^{4063} \bmod 10007 = 42 = 2\times3\times7$
$5^{5136} \bmod 10007 = 54 = 2\times3^3$
$5^{9865} \bmod 10007 = 189 = 3^3\times7$

We obtain three equations:
$\log_5 2 + \log_5 3 + \log_5 7 \equiv 4063 \pmod{10006}$
$\log_5 2 + 3 \log_5 3 \equiv 5136 \pmod{10006}$
$3 \log_5 3 + \log_5 7 \equiv 9865 \pmod{10006}$

From these three equations, we obtain:

$\log_5 2 = 6578$, $\log_5 3 = 6190$, $\log_5 7=1301$

Index calculus algorithm

- Example (cont.)

  To find the value of $x = \log_5 9451$

  For an $s = 7736$,

  $5^{7736} \times 9451 \bmod 10007 = 8400 = 2^4 \times 3^1 \times 5^2 \times 7^1$

  Thus

  $7736 + x \equiv (4\log_5 2 + \log_5 3 + 2\log_5 5 + \log_5 7) \pmod{10006}$

  $x = 6057$

Discrete Logarithm Algorithms

- Complexity:

  Shank's baby-step giant-step alg.: $O(e^{0.5 \ln p})$

  Pollard's Rho discrete logarithm alg.: $O(e^{0.5 \ln p})$

  Pohlig-Hellman alg.: $< O(e^{0.5 \ln p})$

  (depending on the factors of *p*-1)

  Index calculus method: $O(e^{(1+O(1))\sqrt{\ln p \ln \ln p}})$

- Because of the Index calculus method, the size of *p* is at least 2048-bit when ElGamal is used in applications

Other Cyclic Groups

- Integer addition modulo p (Z_p^+)
  - Discrete log problem is easy: $g \cdot x \equiv b \pmod{p}$
  - Cannot be used for ElGamal encryption
- Elliptic curve over finite field with the operation of addition
  - Discrete log problem is hard
  - The index calculus method cannot be applied
    - Small public key size is possible
  - Can be used to replace Z_p^* in ElGamal encryption

Application of ElGamal Encryption

- Not as widely used as RSA encryption
  - The size of ciphertext of ElGamal encryption is large
    - twice that of *p*
  - ElGamal encryption is used in the latest version of PGP
    - PGP (Pretty Good Privacy) is a software typically used for encrypting and signing emails
    - RSA is also recommended in PGP

Summary

- ElGamal Encryption
  - Specification
  - Implementation
  - Security
    - Discrete logarithm algorithms
      - Shank's baby-step giant-step algorithm
      - Pollard's Rho algorithm
      - Pohlig-Hellig algorithm
        - » p-1 should have a large prime factor
      - Index calculus algorithm
        - » Large p: 3072-bit p for 128-bit security
    - Do not re-use the per-message secret k