MAS 433: Cryptography

Lecture 15
Public Key Encryption
Part 2. ElGamal

Wu Hongjun

Lecture Outline

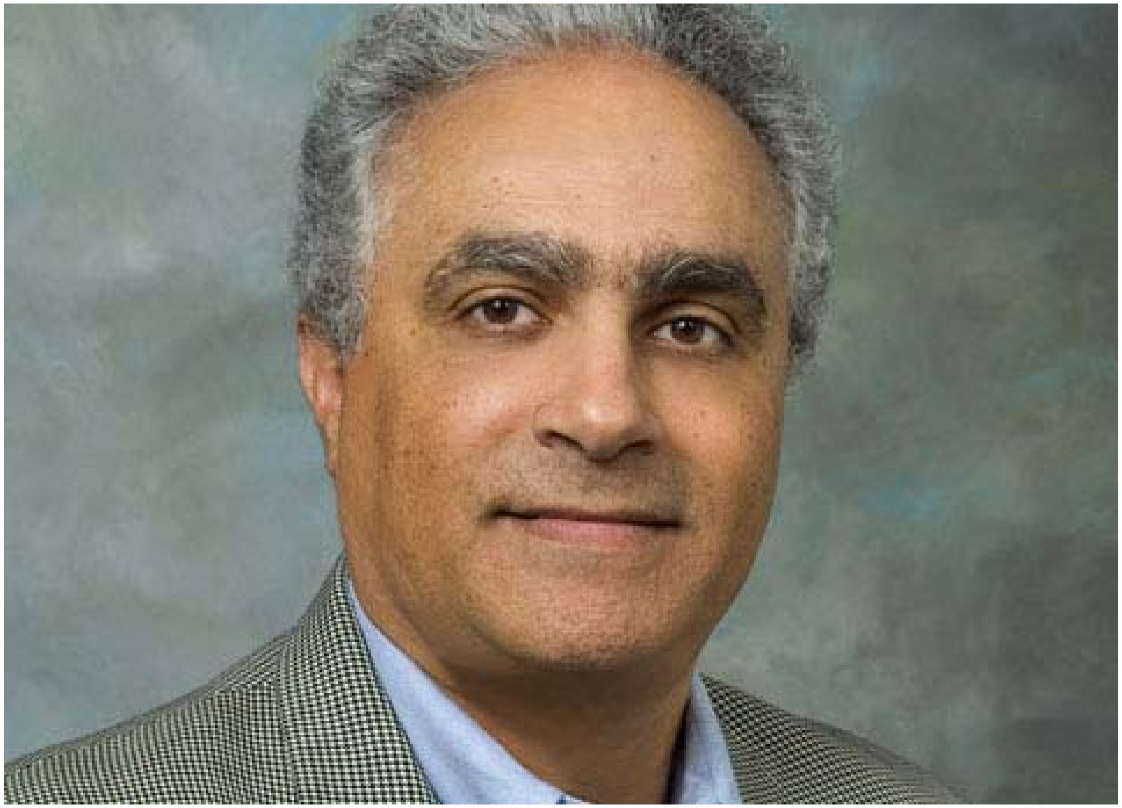
- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
- Public key encryption
  - RSA
  - **ElGamal**
    - **Specification**
    - **Implementation**
    - **Security**
  - Message padding (OAEP)
- Digital signature
- Key establishment and management
- Introduction 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 Cryptosystem

- Based on discrete logarithm problem
- Invented by Taher Elgamal, 1985



ElGamal Cryptosystem

- Discrete logarithm problem (for Z_p^*):

1. Let p be a large prime
2. Let g be a generator of the multiplicative cyclic group Z_p^*
3. Given $y \in Z_p^*$, difficult to find x satisfying $g^x \bmod p = y$

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.

ElGamal Cryptosystem

- **ElGamal encryption**
- ElGamal digital signature (to be learned later)

ElGamal Encryption: Specification

- Key generation

1. Generate a large random prime p
2. Find a generator g of the multiplicative group Z_p^* of the integers modulo p.
3. Select a random 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: Implementation

- How 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

Question: How to factorize p-1 if p is large?
(for security reason, the size of p is normally as large as 2048-bit, but the factorization of large integer is hard.)

ElGamal Encryption: Implementation

Question: How to factorize *p*-1 if *p* is large?
(for security reason, the size of *p* is normally as large as 2048-bit)

Answer: In practice, we select the factors of *p*-1 first, then test whether *p* is prime or not. One of the factor of *p*-1 must be large

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. $c_1 = g^k \bmod p = 2^{1751} \bmod 2357 = 1430$

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

Decryption:

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

ElGamal: Security

- The security of ElGamal depends on the difficulty of discrete logarithm
- Discrete logarithm
  - Given a cyclic group with order n, and
  - Simply, how difficult it is to find x satisfying

$$g^x = b \bmod p \quad ?$$

Discrete Logarithm Algorithms

- Shank’s baby-step giant-step algorithm
- Pollard's rho algorithm for discrete logarithm
- Pohlig-Hellman algorithm
- Index calculus algorithm
- ……

generic

Shank's baby-step giant-step algorithm

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

- 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: $\log_2 15 \bmod 19 = ?$

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

T1: $(i, g^{i\times t} \bmod 19)$	T2: $(j, b\times g^{-j} \bmod 19)$	
(0, 1)	(0, 15)	
(1, 13)	(**1**, **17**)	$j = 1$
(**2**, **17**)	(2, 18)	$i = 2$
(3, 12)	(3, 9)	$x = i\times t + j = 11$
(4, 4)	(4, 14)	

$\log_2 15 \bmod 19 = 11$

Pollard's rho algorithm for discrete logarithm

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

- Basic Idea: Birthday attack
  - Compute ($g^u \bmod p$) for $n^{0.5}$ random values of u
  - Compute ($b^v \bmod p$) for $n^{0.5}$ random values of v
  - Due to birthday paradox, we can find that there is one pair (u,v) satisfying ($g^u \bmod p$) = ($b^v \bmod p$)
    - It means that $g^u \equiv g^{xv} \bmod p$, i.e., $u \equiv xv \bmod p$-1
      => find x successfully
  - Complexity
    - $O(n^{0.5})$ computation, $O(n^{0.5})$ memory

Pollard's rho algorithm for discrete logarithm

$g^x = b \bmod p$

$n = p\text{-}1$

- Pollard' 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

$g^x = b \bmod p$
$n = p\text{-}1$

- Pollard' 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

$$f(x_{i+1}) = \begin{cases} bx_i & x_i \in G_0 \\ x_i^2 & x_i \in G_1 \\ gx_i & x_i \in G_2 \end{cases}$$

Pohlig-Hellman Algorithm

$g^x = b \bmod p$

$n = p\text{-}1$

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

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

$$g^x \equiv b (\bmod p)$$
$$(g^x)^{n/p_i} \equiv b^{n/p_i} (\bmod p)$$
$$(g^{u_i \cdot p_i + v_i})^{n/p_i} \equiv b^{n/p_i} (\bmod p)$$
$$(g^{v_i})^{n/p_i} \equiv b^{n/p_i} (\bmod p)$$
$$(g^{n/p_i})^{v_i} \equiv b^{n/p_i} (\bmod p) \qquad (1)$$

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

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

In (2), v_i can be found using Pollard's Rho method (complexity $O(\sqrt{p_i})$)

(g' can be considered as a generator with order p_i)

(Modification is needed if p_i appears more than once in n)

Index calculus algorithm

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

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

Precomputation :

Step 1. Determine a value B

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={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:

$\log_5 2 + \log_5 3 + \log_5 7 \equiv 4063 \bmod 10006$
$\log_5 2 + 3 \log_5 3 \equiv 5136 \bmod 10006$
$3 \log_5 3 + \log_5 7 \equiv 9865 \bmod 10006$

From these three equations, we obtain:

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

Index calculus algorithm

- Example (contd.)

  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

  $$7736 + x \equiv (4\log_5 2 + \log_5 3 + 2\log_5 5 + \log_5 7) \bmod 10006$$

  $$x = 6057$$

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.: $< O(e^{0.5\ln p})$

  (depdending on the factors of p-1)

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

Application of ElGamal Encryption

- Not widely used
  - The size ciphertext is large
    - twice that of p
  - Used in the latest version of PGP

Other Cyclic Group

- Integer addition over Z_p
  - Too weak: $g \cdot x = b \bmod p$
- Addition over elliptic curve
  - Strong
  - The index calculus method cannot be applied
    - Small public key size is possible
  - Public key cryptosystems based on elliptic curve may become popular in applications in the future

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: 2048-bit p for 128-bit security
    - Do not re-use the per-message secret k