Discrete Log Based Cryptosystems

密碼學與應用

海洋大學資訊工程系
丁培毅

Discrete Log Problem

- Given a prime number p, $\alpha \in Z_p{}^*$, $\beta \equiv \alpha^x \pmod{p}$ '*finding x*' is called the discrete logarithm problem
- Not every discrete log problem has solution and not every discrete log problem is hard
- if n is the smallest positive integer such that $\alpha^n \equiv 1 \pmod{p}$ (i.e. $n=\mathrm{ord}_p(\alpha)$) we may assume $0 \le x < n$, and then denote

$$x = L_\alpha(\beta)$$

x is the discrete log of β with respect to α

- *ex.* $p = 11$, $\alpha = 2$, $2^6 \equiv 9 \pmod{11}$, $L_2(9) = 6$

Discrete Log Problem

- Often α is a primitive root modulo p, which means that every β in Z_p^* is a power of α (mod p).
- If α is not a primitive root, then the discrete log will not be defined (i.e. no solution) for certain values of β in Z_p^*.
- If α is a primitive root modulo *p*, then

$$L_\alpha(\beta_1\beta_2) \equiv L_\alpha(\beta_1) + L_\alpha(\beta_2) \pmod{\mathbf{p-1}}$$

- When p is small, it is easy to compute discrete logs by exhaustive search through all possible exponents
- When p is large and satisfying a certain properties, solving a discrete logarithm problem is "believed to be hard"
- The bit length of the largest prime number for which discrete logarithm can be computed is approximately the same size of the largest integer that can be factored. (2001: 110-digit (370-bit) prime numbers for discrete logs, 155-digit (512-bit) integers for factoring)

One-Way Function

- $f(x)$ is a one-way function if
  - given x, $f(x)$ is easy to compute
  - given y, it is "computationally infeasible" to find x s.t. $f(x) = y$
- $f(x)$ is a trapdoor one-way function if
  - it is a one-way function
  - given the trapdoor t and y, it is easy to find x s.t. $f(x) = y$
- candidates:
  - modular exponentiation (one-way)
  - multiplication of large primes (one-way)
  - RSA function (trapdoor one-way)
  - modular square (trapdoor one-way)

Discrete Log Based Systems

- Diffie-Hellman Key Exchange
- Pohlig-Hellman Secret Key System
- ElGamal Cryptosystem / Signature Scheme
- Cramer-Shoup Cryptosystem
- Digital Signature Standard (DSS, DSA)
- Schnorr Signature Scheme
- Paillier Cryptosystem (both Factoring & DL)
- Boneh-Franklin Identity-based Encryption

Compute Discrete Log

- Pohlig-Hellman, Birthday Attack, Index-Calculus, Baby-step Giant-step
- Preliminary:
  - let α be a primitive root modulo p so p-1 is the smallest positive exponent such that $\alpha^{p-1} \equiv 1 \pmod p$

    $$\alpha^{m_1} \equiv \alpha^{m_2} \pmod p \Leftrightarrow m_1 \equiv m_2 \pmod{p-1}$$

  - consider the discrete log problem $\beta \equiv \alpha^x \pmod p$, it is difficult to find out the value of x, but it is easy to find out whether x is even or odd i.e. $x \pmod 2$ or the LSB of x

    $$\boxed{\text{if } \beta^{(p-1)/2} \text{ is -1 then x is odd; else if } \beta^{(p-1)/2} \text{ is 1 then x is even}}$$

    $(p$-1)/2 is an integer

    $(\alpha^{(p-1)/2})^2 \equiv \alpha^{(p-1)} \equiv 1 \pmod p \Rightarrow \alpha^{(p-1)/2} \equiv \pm 1 \pmod p$

    because α is a primitive root, $\alpha^{(p-1)/2} \equiv -1 \pmod p$

    therefore, $\beta^{(p-1)/2} \equiv \alpha^{x\,(p-1)/2} \equiv (-1)^x \pmod p$

  - using the same method, if $2^k \mid p-1$, it is easy to calculate the k-LSB bits of x

Baby-step Giant-step

- *Meet-in-the-middle* algorithm for computing discrete logarithm
- D. Shanks, 1971

To solve $\alpha^x \equiv \beta \pmod{n}$,

① write $x = i\,m + j$, $0 \le i,j < m = \lceil \sqrt{n} \rceil$

② test all i,j, for $\beta\, (\alpha^{-m})^i \equiv \alpha^j \pmod{n}$

- Running time and space complexity is $O(\sqrt{n}\,)$ ($\ll O(n)$ brute-force)
- A generic algorithm, works for every finite cyclic group.
- not necessary to know the order of the group G in advance. It still works if n is merely an upper bound on the group order.
- Usually is used for groups whose order is prime. Pohlig-Hellman algorithm is more efficient for composite order group.

Pohlig-Hellman Algorithm

- compute the discrete logs when p-1 has only small prime factors
- let $p\text{-}1=\prod_i q_i^{r_i}$ be the factorization of p-1 into prime numbers
- Plans: compute $L_\alpha(\beta)$ (mod $q_i^{r_i}$) then use CRT to find $L_\alpha(\beta)$ (mod p-1)

$$\text{let } x = x_0 + x_1 q + x_2 q^2 + \ldots + x_{r-1} q^{r-1} + \ldots$$

where $x_i \in Z_q$ i.e. express x in q-ary representation

$$x\left(\frac{p\text{-}1}{q}\right) = x_0\left(\frac{p\text{-}1}{q}\right) + (p\text{-}1)\,(x_1 + x_2 q + x_3 q^2 + \ldots) = x_0\left(\frac{p\text{-}1}{q}\right) + (p\text{-}1)\,n$$

$$\beta^{(p\text{-}1)/q} \equiv \alpha^{x(p\text{-}1)/q} \equiv \alpha^{x_0(p\text{-}1)/q}\left(\alpha^{(p\text{-}1)}\right)^n \equiv \alpha^{x_0(p\text{-}1)/q} \pmod{p}$$

Pohlig-Hellman Algorithm

To find x_0, we enumerate $\alpha^{k(p-1)/q}$ (mod p), k=0,1,2,…q-1, and match against with $\beta^{(p-1)/q}$, there is a unique solution since $k(p-1)/q$ (mod p-1) are all different for k=0,1,2,…q-1

- extension of the above procedure yields the remaining coefficients

  assume $q^2 \mid p-1$ $\quad \beta_1 \equiv \beta\, \alpha^{-x_0} \equiv \alpha^{q(x_1+x_2 q+\ldots)} \pmod{p}$

  $$\beta_1^{(p-1)/q^2} \equiv \alpha^{(p-1)(x_1+x_2 q+\ldots)/q} \equiv \alpha^{x_1 (p-1)/q} \left(\alpha^{(p-1)}\right)^{x_2+x_3 q+\ldots}$$
  $$\equiv \alpha^{x_1 (p-1)/q} \pmod{p}$$

  to find x_1, we enumerate $\alpha^{k(p-1)q}$ (mod p), k=0,1,2,…q-1, and match against with $\beta_1^{(p-1)/q^2}$

- Why should q be small for Pohlig-Hellman algorithm to work??
  - ★ The algorithm needs to enumerate $\alpha^{k(p-1)/q}$ (mod p), k=0,1,…q-1

Pohlig-Hellman Algorithm

- Note: the above enumerations are the same in computing each x_i (i.e. can be stored and used several times)
- In a Discrete Log based cryptosystem, we should make sure that p-1 has at least a large prime factor.
- If p-1 = $t \cdot q$ (i.e. p-1 has a large prime factor q), the algorithm can still determine $L_\alpha(\beta) \pmod{t}$ if t is composed of small prime factors. (still leaks much information, if $t = 2^{10}$, 10-LSB bits of $L_\alpha(\beta)$ will be known)
  - Usually β is chosen to be a power of α^t such that $L_\alpha(\beta) \pmod{t}$ is zero. $\beta = (\alpha^t)^m \equiv \alpha^x \pmod{p} \Rightarrow x \equiv t\,m \pmod{p-1} \Rightarrow x \equiv 0 \pmod{t}$
  - However, the difficulty of this discrete log problem is reduced no matter what β you choose. It only guarantees that $L_\alpha(\beta) \pmod{q}$ is difficult, you should not hide any information in $L_\alpha(\beta) \pmod{t}$

Index Calculus

- ✧ Idea is similar to the quadratic sieve method of factoring.
- ✧ Factor base: prime numbers less than a bound B, $\{p_1, p_2, \ldots p_m\}$
- ✧ Example: $p=131$, $\alpha=2$. Let $B=10$, consider the prime numbers $\{2, 3, 5, 7\}$

$$\left\{\begin{array}{lll} 2^1 \equiv 2 & & \pmod{131} \\ 2^8 \equiv 5^3 & & \pmod{131} \\ 2^{12} \equiv 5 \cdot 7 & & \pmod{131} \\ 2^{14} \equiv 3^2 & & \pmod{131} \\ 2^{34} \equiv 3 \cdot 5^2 & & \pmod{131} \end{array}\right. \Longrightarrow \left\{\begin{array}{lll} 1 \equiv L_2(2) & & \pmod{130} \\ 8 \equiv 3\,L_2(5) & & \pmod{130} \\ 12 \equiv L_2(5) + L_2(7) & & \pmod{130} \\ 14 \equiv 2L_2(3) & & \pmod{130} \\ 34 \equiv L_2(3) + 2L_2(5) & & \pmod{130} \end{array}\right.$$

$$\Longrightarrow \left\{\begin{array}{ll} L_2(2) \equiv 1 & \pmod{130} \\ L_2(3) \equiv 72 & \pmod{130} \\ L_2(5) \equiv 46 & \pmod{130} \\ L_2(7) \equiv 96 & \pmod{130} \end{array}\right.$$

If we want to compute $L_2(37)$
try a few random exponents and found
$37 \cdot 2^{43} \equiv 3 \cdot 5 \cdot 7 \pmod{131}$, therefore,
$L_2(37) \equiv -43 + L_2(3) + L_2(5) + L_2(7)$
$\equiv 41 \pmod{130}$

Index Calculus

- Precomputation:
  - Compute α^k (mod p) for several values of k
  - Try to write it as a product of the primes less than B. i.e. $\alpha^k = \Pi\, p_i^{a_i}$ (mod p) If this is not the case, try another k. Then

    $$k \equiv \sum a_i\, L_\alpha(p_i) \pmod{p-1}$$

    when we have enough such relations, we can solve for $L_\alpha(p_i)$ for each i
- For some random r, compute $\beta\, \alpha^r$ and try to write it as a product of $\{p_1, p_2, \ldots p_m\}$ i.e. $\beta\, \alpha^r = \Pi\, p_i^{b_i}$ (mod p)

  $$L_\alpha(\beta) \equiv -r + \sum b_i\, L_\alpha(p_i) \pmod{p-1}$$
- This algorithm is effective if p is of moderate size.
- This means that p should be chosen to have at least 200 digits (~665 bits), if the discrete log problem is to be hard.

Computing Discrete Log Mod 4

- Discrete Log Problem: Given α, β, p solving $x = L_\alpha(\beta)$ such that $\beta \equiv \alpha^x \pmod{p}$
- Using Pohlig-Hellman Algorithm, if $p \equiv 1 \pmod{4}$, then it is easy to compute $L_\alpha(\beta) \pmod{4}$
- For $p \equiv 3 \pmod{4}$, Pohlig-Hellman Algorithm does not show us a way to calculate $L_\alpha(\beta) \pmod{4}$ since it is easy to raise an integer to the $(p\text{-}1)/2$ power but it is not easy to raise an integer to the $(p\text{-}1)/4$ power.
- Idea: we can take square root of a QR when $p \equiv 3 \pmod{4}$
  i.e. Given y, find x, s.t. $x^2 \equiv y \pmod{p}$

$$x \equiv \pm y^{\frac{p+1}{4}} \pmod{p}$$

Computing Discrete Log Mod 4

- To find $\gamma^{(p\text{-}1)/4}$: Can we find $\gamma^{(p\text{-}1)/2}$ first and then take square root of it? In this way, it seems that we can calculate $L_\alpha(\beta)$ (mod 4) and even $L_\alpha(\beta)$ (mod 8) …and the Discrete Log Problem can be easily solved???
- What's wrong with the above arguments?
  - From the formula on the previous slide, given $\gamma^{(p\text{-}1)/2}$ you won't be able to get one single $\gamma^{(p\text{-}1)/4}$, instead you get two possible values. Since $L_\alpha(\beta)$ (mod 4) has one bit more information than $L_\alpha(\beta)$ (mod 2), you actually do not get any more information through the procedure just described.

    equally possible
  - On the next slide, we prove this with a 'reduction argument'. "if we have an algorithm that can calculate $L_\alpha(\beta)$ (mod 4) efficiently, we can use it to compute discrete log quickly"

Computing Discrete Log Mod 4

- Lemma. Let $p \equiv 3 \pmod 4$ be prime, let $r \geq 2$, and let y be an integer. Suppose α and γ are two elements in Z_p^* such that $\gamma \equiv \alpha^{2^r y} \pmod p$. Then

$$\gamma^{(p+1)/4} \equiv \alpha^{2^{r-1} y} \pmod p$$

Proof:

$$\gamma^{(p+1)/4} \equiv \alpha^{(p+1)2^{r-2} y} \equiv \alpha^{(p-1+2)2^{r-2} y} \equiv \alpha^{2^{r-1} y} \underbrace{\alpha^{(p-1)2^{r-2} y}}_{1}$$

$$\equiv \alpha^{2^{r-1} y} \pmod p$$

Note: this is similar to the method of taking square root

the key difference is that $\gamma^{(p+1)/4}$ is equal to a single value instead of two, since $\alpha^{2^{r-1} y}$ is a quadratic residue (QR) which is always positive

Computing Discrete Log Mod 4

✧ "if we have an algorithm that can calculate $L_\alpha(\beta) \pmod 4$ efficiently, we can use it to compute discrete log quickly"

Proof:

- ✧ assume we have a machine that, given an input β, outputs $L_\alpha(\beta) \pmod 4$
- ✧ assume $\beta \equiv \alpha^x \pmod p$, let $x = x_0 + 2x_1 + 4x_2 + \ldots + 2^n x_n$ be the binary representation of x, using the $L_\alpha(\beta) \pmod 4$ machine, we determine x_0 and x_1
- ✧ let $\beta_2 \equiv \beta\, \alpha^{-(x_0+2x_1)} \equiv \alpha^{2^2(x_2+2x_3+2^2x_4+\ldots)} \pmod p$, using the previous lemma, $(\beta_2)^{(p+1)/4} \equiv \alpha^{2(x_2+2x_3+2^2x_4+\ldots)} \pmod p$, using the $L_\alpha(\beta) \pmod 4$ machine, we determine x_2
- ✧ repeat the above n-3 times, we can obtain $x_3, x_4, x_5, \ldots x_n$ and the discrete log $L_\alpha(\beta) \pmod{p-1}$ is easily solved!!!

✧ Because we believe that discrete log is hard to compute in general, we are comfortable to accept that $L_\alpha(\beta) \pmod 4$ is difficult to calculate.

Bit Commitment

- The story
  - Alice claims that she has a method to predict the outcome of football games
  - Alice wants to sell her method to Bob
  - Bob asks her to prove her method works by predicting the result of the game that will be played this weekend.
  - "No way!!" says Alice. "Then you will simply make your bets and not pay me. If you want me to prove my method works, why don't I show you my prediction for last weeks game?"
- Alice wants to send a bit *b* to Bob. The requirements:
  - Bob cannot determine the value of the bit without Alice's help
  - Alice cannot change the bit once she sends it to Bob.
- Analogy: Sealed Envelop, Locked Safety Box

Bit Commitment with DL

- Alice and Bob agree on a large prime $p \equiv 3 \pmod 4$ and a primitive root α
- Commit
  - Alice chooses a random number $x < p-1$ whose second bit x_1 is b
  - Alice sends $\beta \equiv \alpha^x \pmod p$ to Bob
- Reveal
  - Alice sends Bob the full value of x
  - Bob checks $\beta \equiv \alpha^x \pmod p$ and finds $b \equiv x \pmod 4$.
- We assume that Bob cannot compute discrete logs for p. Therefore, he can not compute discrete logs modulo 4 (i.e. x_1 or b).

Bit Commitment with DL

- To avoid Alice denying that she knows x at the revealing stage, Bob could ask Alice to make a ZKP of knowing x at the commitment stage.
- To avoid Alice denying that she had sent β, Bob could ask Alice to digitally sign β.

General Bit Commitment Schemes

- Two stages:
  - Commit
  - Reveal (Disclosure)
- Formal Requirements:
  - Secrecy (hiding)
  - Unambiguity (binding)
- Various Schemes
  - Using Symmetric Cryptography
  - Using One Way Functions (eg. RSA, Discrete logs)
  - Using Pseudo Random Number Generator (PRNG)
  - Using Oblivious Transfer

Pohlig-Hellman Secret Key System

- Secret Key system, Alice and Bob trust each other.
- Alice and Bob share a pair of secret key (x, x^{-1}) where $x \cdot x^{-1} \equiv 1 \pmod{p-1}$, gcd(x, p-1)=1 (i.e. x is odd), p is a large prime number and (p-1)/2 is also a large prime number
- Encryption

$$c \equiv m^x \pmod{p}$$

- Decryption

$$m \equiv c^{x^{-1}} \pmod{p}$$

Note: 1. x^{-1} can be easily derived from x and p

2. $ord_p(m)$ should be large (since $ord_p(m) | p-1$, it has better be p-1 or (p-1)/2)

Diffie-Hellman Key Exchange

- ✧ Diffie and Hellman, 1976, first Public Key System
- ✧ Used now in IPSec and SSL for jointly generating encryption keys and exchanging symmetric data encryption keys (DES, 3DES…)

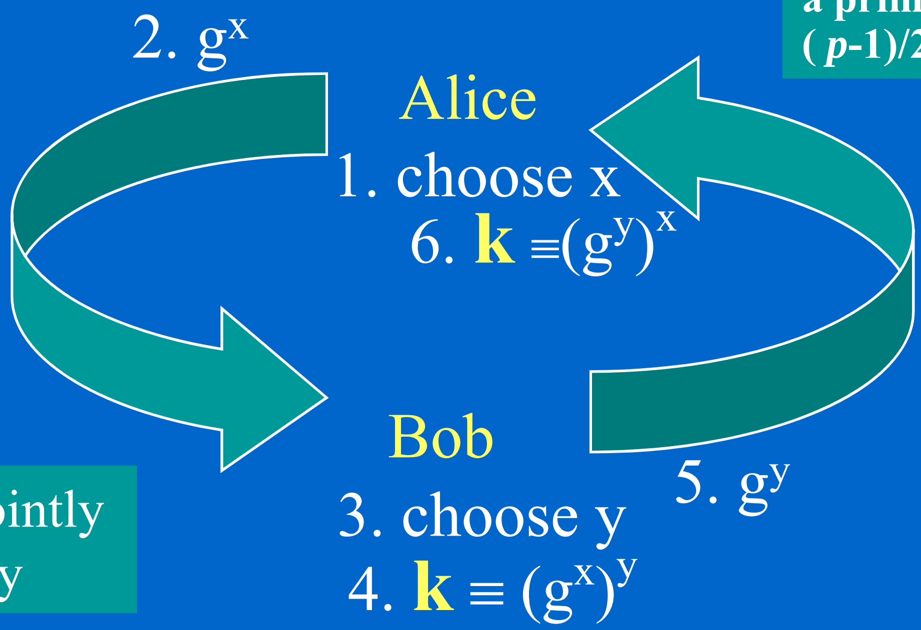
the length of p is usually 1024 bits, often the order of α can be constrained to a 160-bit (or 256-bit) q, therefore, x_a and x_b can be reduced to 160 bit

- ✧ Protocol:
  - ✶ Alice and Bob use a public modulus p and a primitive α.
  - ✶ Alice chooses a private exponent x_a in Z_p^*, computes the public value $y_a \equiv \alpha^{x_a} \pmod{p}$, and sends y_a to Bob.
  - ✶ Bob chooses a private exponent x_b in Z_p^*, computes the public value $y_b \equiv \alpha^{x_b} \pmod{p}$, and sends y_b to Alice.
  - ✶ Alice calculates the shared key as $y_b^{x_a} \equiv \alpha^{x_a x_b} \pmod{p}$ and Bob calculates the shared key as $y_a^{x_b} \equiv \alpha^{x_a x_b} \pmod{p}$

Diffie-Hellman Key Exchange

- Any commutative one-way function can be used to design this type of public key distribution system. Other than the modulo exponential function, Lucas Function and Elliptic Curve Function are also candidates.

all operations are modulo *p*, *p* is a prime number and is chosen s.t. (*p*-1)/2 also a large prime number



generate key k jointly and exchange key

Optional CA

Alice	g^x
Bob	g^y

DDH problem

- Computational Diffie-Hellman Assumption
  - given g^x and g^y, there is no efficient algorithm that can compute g^{xy}
  - do not guarantee that partial bits of g^{xy} are hidden, the Legendre symbol of g^{xy} is leaked
- Decision Diffie-Hellman Assumption
  - Boneh, 1998, "The decision Diffie-Hellman Problem"
  - given g^x and g^y, there is no efficient algorithm that can distinguish the distribution of $< g^x, g^y, g^{xy}>$ and $< g^x, g^y, g^z>$
  - far stronger than the DH assumption
  - can be used to construct efficient cryptographic systems with strong security properties
  - In a group where DDH does not hold, ElGamal Cryptosystem is not semantically secure (the Legendre symbol of m is leaked)

DDH problem (cont'd)

- Legendre symbol of z in Z_p^*: $z^{(p-1)/2}$ (mod p)
  if z is a QR_p then its Legendre symbol is 1, otherwise –1
- g^y is a quadratic residue modulo p iff LSB of y is 0 (i.e. y is even)
- If one of x or y is even, then xy is even and g^{xy} is a quadratic residue

---

- The DDH assumption is stronger than the DL assumption:
  Assuming that adversary cannot solve discrete log cannot guarantee that DH key exchange is safe. DH key exchange is only safe under the DDH assumption.

- break DDH $\Leftarrow$ break CDH $\Leftarrow$ break DL
  DDH is secure $\Rightarrow$ CDH is secure $\Rightarrow$ DL is secure
  (intractable) (intractable) (intractable)
- break RSA $\Leftarrow$ break FACT
  RSA is secure $\Rightarrow$ Fact is secure

DDH in Z_p^*

- Given g^x, g^y, g^z one can easily test if x is odd, y is odd, and z is odd.
- Ex. If x is odd, y is odd and z is even, then z can not be xy

x	y	z	result
odd	odd	odd	nothing
odd	odd	even	$z \neq xy$
odd	even	odd	$z \neq xy$
odd	even	even	nothing
even	odd	odd	$z \neq xy$
even	odd	even	nothing
even	even	odd	$z \neq xy$
even	even	even	nothing

  in Z_p^*, there are at least 1/2 probability that DDH does not hold

- Modification: consider the DDH problem in an order-q subgroup generated by $h \equiv g^2 \pmod{p}$ in Z_p^* where $p=2q+1$, p and q are prime numbers, g is a primitive in Z_p^*

Goals of Modern Cryptography

- Make the intractability assumption more adequate, specific, and clear
- Design cryptosystem that depends on less strict assumptions
- Proven security

Security of Diffie-Hellman Algorithm

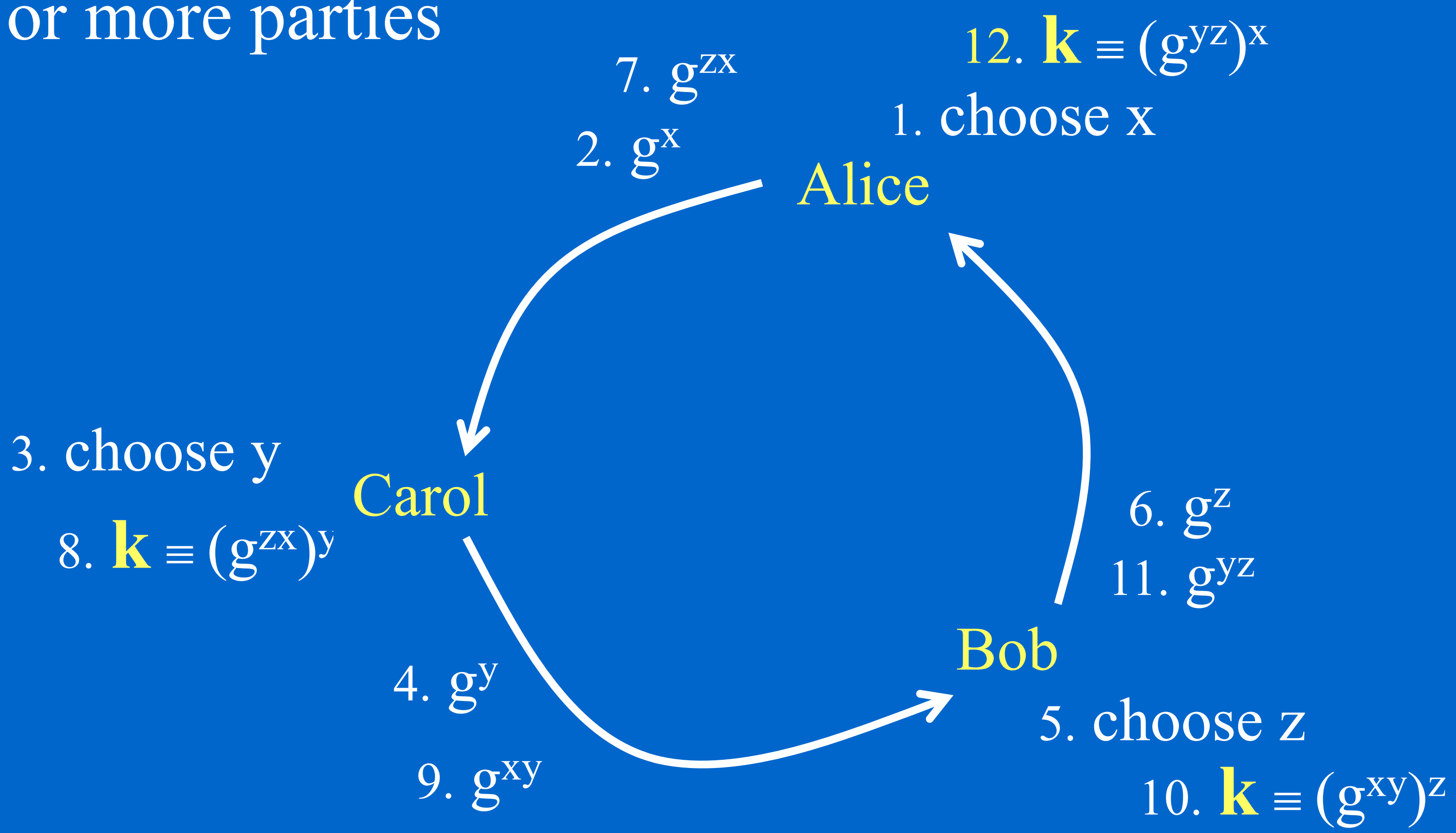
- still an assumption … the 'DH assumption'
- DH is secure $\Rightarrow$ DL is secure (break DH $\Leftarrow$ break DL)

  if DL is not secure, i.e. given g^x we can solve for x and given g^y we can solve for y, then DH is not secure. Eve can intercept g^x and g^y and easily derives x or y and computes the shared key $(g^x)^y$ or $(g^y)^x$
- DL is secure $\not\Rightarrow$ DH is secure

  if DH can be broken, i.e. given g^x and g^y, shared key k= g^{xy} can be derived. Since k = $(g^x)^y = (g^y)^x$, not too much information about x or y can be derived from the above equation.
- In general, it is believed that DL is secure, but it does not provide any assurance about whether DH is secure (Eve might be able to predict some of the bits of g^{xy})

Diffie-Hellman Key Exchange

- ✧ Three or more parties



- ✧ Conference Key Distribution System (CKDS)

Diffie-Hellman Key Exchange

- Variants: Hughes Crypto'94
  - Allow Alice to generate a key and send it to Bob

Alice

1. choose x
2. $\mathbf{k} \equiv g^x$

5. $(g^y)^x$

4. g^y

Bob

3. choose y

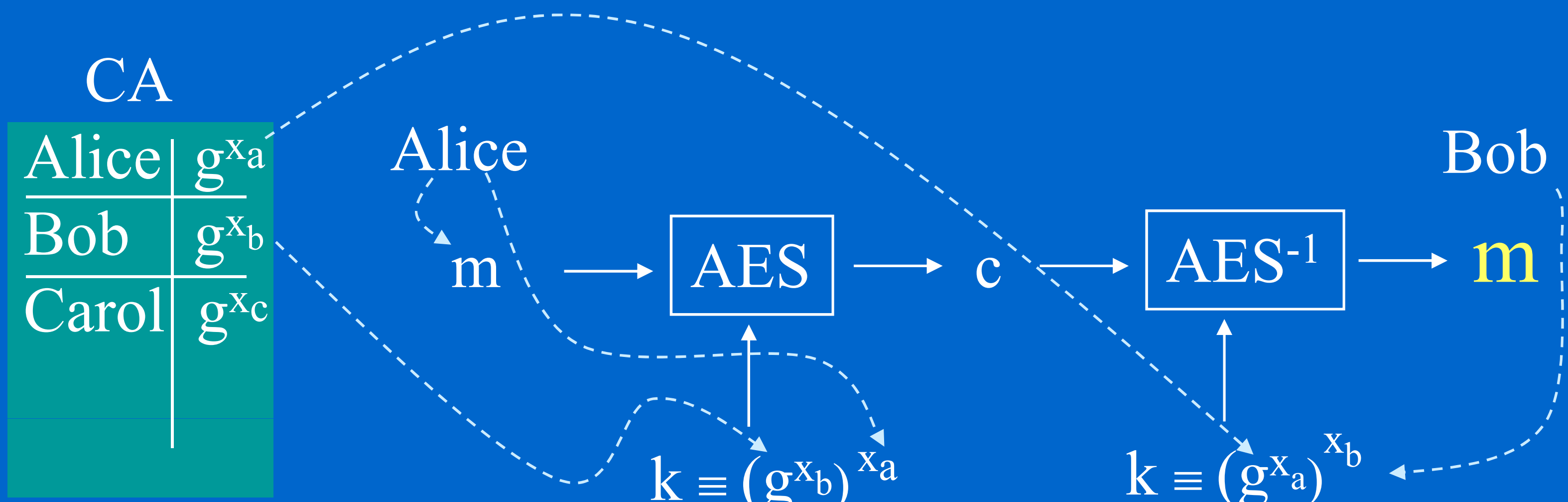
6. $\mathbf{k} \equiv ((g^y)^x)^{y^{-1}} \equiv g^x$

$y \cdot y^{-1} \equiv 1 \pmod{p-1}$

all other operations are (mod p)

DH sharing secret keys in a group

- If each pairs in a group (ex. {A, B, C, D, E, F}) want to use symmetric encryption system (like AES) to communicate frequently. They need to share, in this example, 30 keys. Everyone need to share five keys with others.
- Alternative: Each one in the group chooses a secret number $\{x_a, x_b, x_c, x_d, x_e, x_f\}$. We can have a central database to keep and certify all public values $\{g^{x_a}, g^{x_b}, g^{x_c}, g^{x_d}, g^{x_e}, g^{x_f}\}$, and use DH as follows:



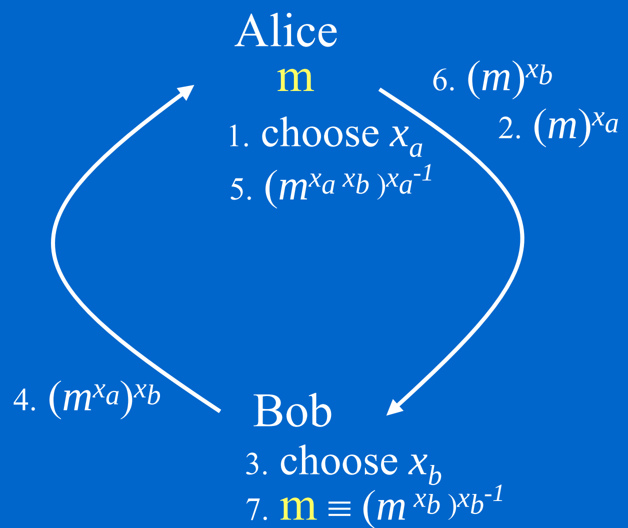
Diffie-Hellman Protocol and Attack

- RFC 2631, Diffie-Hellman Key Agreement Method, E. Rescorla, June 1999

- small subgroup attack
  - L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", Technical report CORR 98-05, University of Waterloo, 1998.
  - C.H. Lim and P.J. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup", Crypto'97, pp. 249-263.

3-Pass Communication Protocol

- Shamir
- Alice wants to send a secret message *m* to Bob. They use a common large prime number *p*
- Protocol:
  - Alice chooses a secret number x_a and Bob chooses a secret number x_b such that x_a^{-1} and x_b^{-1} (mod p-1) exist
  - Alice sends $y_1 \equiv m^{x_a}$ (mod p) to Bob
  - Bob sends $y_2 \equiv y_1^{x_b}$ (mod p) to Alice
  - Alice sends $y_3 \equiv y_2^{x_a^{-1}}$ (mod p) to Bob
  - Bob computes $m \equiv y_3^{x_b^{-1}}$ (mod p)



- Key idea: modulo exponentiation is commutative
- Analogy: a safety box with two locks
- Any commutative trapdoor oneway function can be used

ElGamal PKC

- ElGamal 1985 (9 years after Diffie-Hellman)
- Probabilistic Encryption System: For the same public key, the same plaintext could give different ciphertexts in distinct encryption sessions. This can resist low-entropy attack.

Low entropy attack:

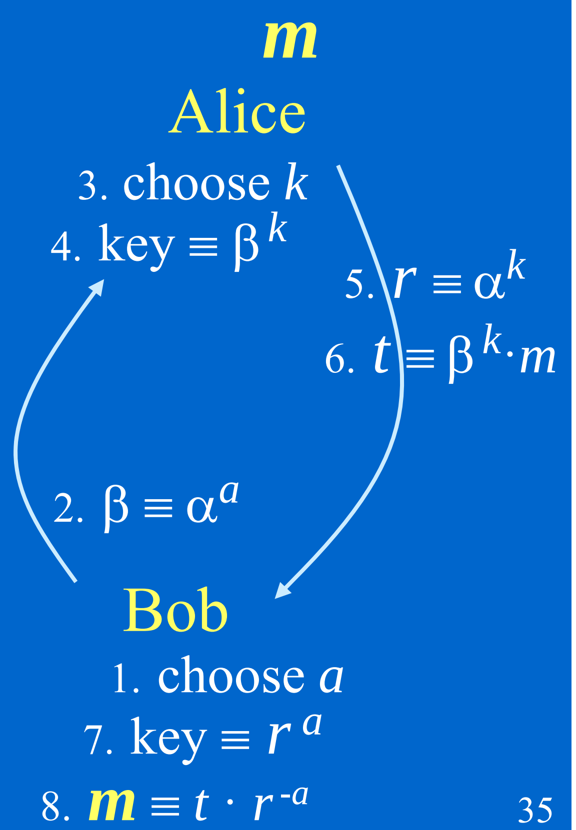
Number of messages is small.
Some messages occur much more often.
$\Rightarrow$ low entropy in the source

For a deterministic encryption scheme, attacker can record the ciphertext frequency pattern and learn something or use chosen plaintext attack to compile a codebook to decipher the following ciphertext.

- Application of Diffie-Hellman Algorithm

ElGamal PKC

- Alice wants to send a message to Bob
- Bob first chooses a large prime number p, $p = 2q + 1$, q is also prime, a primitive root α', calculate $\alpha = \alpha'^2$, a secret integer a in Z^*_p, and compute $\beta \equiv \alpha^a \pmod{p}$
  - Bob's Private Key: a
  - Bob's Public Key: (p, α, β)
- Encryption:
  - Alice downloads Bob's public key (p, α, β)
  - Alice chooses a secret random integer $k \in Z^*_p$ and compute $r \equiv \alpha^k \pmod{p}$
  - Alice computes $t \equiv \beta^k \cdot m \pmod{p}$
  - Alice sends the ciphertext (r, t) to Bob
- Decryption
  - Bob computes $m \equiv t \cdot r^{-a} \pmod{p}$



ElGamal PKC

- Security
  - If Eve knows a, she can calculate the key $r^a \equiv (\alpha^k)^a$ and decrypt (r, t) like Bob. Therefore, Bob has to **keep *a* secret**. By looking at the public key $\beta \equiv \alpha^a$ and $r \equiv \alpha^k$, Eve can either solve the DH problem to recover the key α^{ka} or solve the DLP to recover a directly, and therefore, the key $(\alpha^k)^a$.
  - If Eve knows the random value k, she can calculate the key by calculating $\beta^k \equiv (\alpha^a)^k$, and decrypt (r, t) by calculating $m \equiv t \cdot \beta^{-k} \pmod{p}$. Therefore, Alice has to **keep *k* secret**. By looking at the public value $r \equiv \alpha^k$ and $\beta \equiv \alpha^a$, Eve can either solve the DH problem to recover the key α^{ka} or solve the DLP to recover k directly, and therefore, the key $(\alpha^a)^k$.

(ElGamal PKC is secure $\Leftrightarrow$ DDH is secure) $\underset{\not\Leftarrow}{\Rightarrow}$ DL is secure

ElGamal PKC

- ✧ Security:
  - ✶ If k is a random integer in $Z_p{}^*$, and if β is a primitive in $Z_p{}^*$, then β^k is a random integer in $Z_p{}^*$ and $t \equiv \beta^k \cdot m \pmod{p}$ is a random integer in $Z_p{}^*$. (recall the $\psi(x)$ in proving the Fermat's Little Theorem). Knowing t and r without knowing a or k does not give Eve any information about m.
  - ✶ **Different *k*** should be used for each m

    If one k is used for two messages m_1 and m_2 sent to Bob, i.e. (r, t_1) and (r, t_2), then Eve can determine m_1 from m_2 or m_2 from m_1 since

    $$t_1/m_1 \equiv t_2/m_2 \equiv \beta^k \pmod{p}$$

    Therefore, it Eve knows m_1

    $$m_2 \equiv t_2\, m_1 / t_1 \pmod{p}$$

ElGamal PKC

✧ Is ElGamel Encryption commutative?

i.e. $E_2(E_1(m) \overset{?}{=} E_1(E_2(m))$ or

$D_1(E_2(E_1(m)) \overset{?}{=} E_2(m)$

- let's say E_1 is for Alice to encrypt messages for Bob and E_2 is for Bob to encrypt messages for Carol
- if both encryption use the same modulus p, then
  $D_1(E_2(E_1(m)) = (\beta_2{}^{k_2} \cdot (\beta_1{}^{k_1} \cdot m\ )) \cdot r_1{}^{-a_1} \equiv \beta_2{}^{k_2} \cdot m = E_2(m)$
- answer is yes if using the same modulus

Semantic Security of ElGamal PKC

- Is ElGamal encryption semantically secure?
  - NOT in arbitrary group: ex. In Z_p^* with a primitive α

Public key: α is a primitive root, $\beta \equiv \alpha^a$ (mod p)
Ciphertext: $(r,t)=(\alpha^k, \beta^k \cdot m)$
Since α be a primitive root in Z_p^*,
Let $m \equiv \alpha^x$ (mod p) and $t \equiv \alpha^y$(mod p)
then $y \equiv a \cdot k+x$ (mod p-1)

known

a	k	y	deduction
odd	odd	odd	x is even
odd	odd	even	x is odd
odd	even	odd	x is odd
odd	even	even	x is even

a	k	y	deduction
even	odd	odd	x is odd
even	odd	even	x is even
even	even	odd	x is odd
even	even	even	x is even

  - Only in an order-q subgroup generated by $\alpha \equiv g^2$ (mod p) in Z_p^* where p=2q+1, p and q are prime numbers, g is a primitive in Z_p^*, under the assumption of DDH

Rogue Key Attack

- A group insider registers public keys as a function of other's public key without demonstrating the possession of the corresponding private keys. e.g.

Alice — Bob registers two related public keys

pk_A: g^x — pk_{B_1}: g^{2x} — pk_{B_2}: g^{3x}

sk_A: x

Assume that sender S wants to broadcast to A, B_1, B_2 keys K_A, K, K with the following ElGamal ciphertext $(g^r, (g^x)^r K_A, (g^{2x})^r K, (g^{3x})^r K)$

Bob can obtain K_A by calculating $(g^x)^r K_A * (g^{2x})^r K * ((g^{3x})^r K)^{-1}$

The problems are: shared randomness, CA does not verify the ownership of the private key.

Discrete Logarithm Timeline