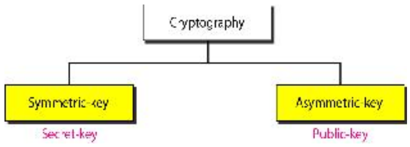
Cryptography

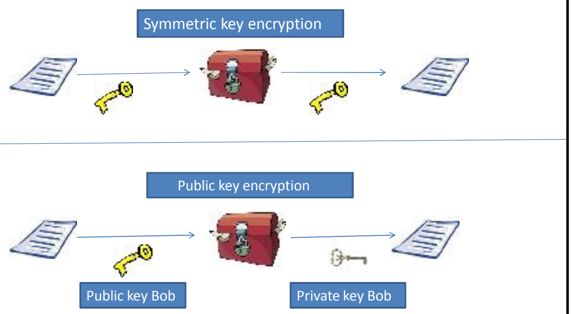
Asymmetric key Cryptography

Anand Ballabh Joshi
Department of Mathematics
University of Lucknow, Lucknow, India

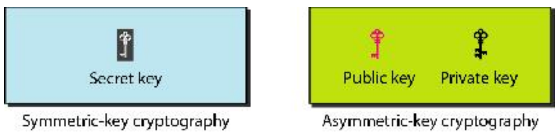
Category of cryptography



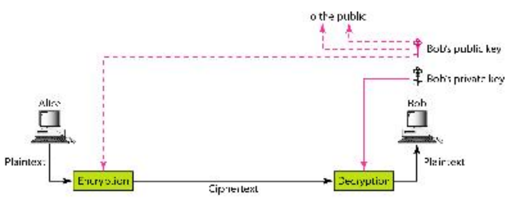
Symmetric and public key encryption



Keys in cryptography



Public (Asymmetric) key cryptography



Prime Numbers

- A number is said to be prime if it is dividible by 1 and itself.
- There is infinitely many prime numbers.
- $® (n) = \lim n / \log n$
- No function to generate all primes
- Mersenne Prime:
  $M_p = 2^p - 1$
  $M_2 = 2^2 - 1 = 3, M_3 = 2^3 - 1 = 7, M_5 = 2^5 - 1 = 31, M_7 = 2^7 - 1 = 127$
  $M_{11} = 2^{11} - 1 = 2047 = 23.89$
- Fermat Prime: $F_n = 2^{2^n} + 1, F_5 = 4294967297 = 641 \times 6700417$

Primality Test

- The scheme for generating large primes like Mersenne and Fermat failed
- How to generate large prime for cryptography
- Choose a large number and test it is prime
- Two categories of testing prime:

Deterministic algorithm: always gives a correct answer

Probabilistic algorithm: gives an answer that is correct most of the time, but not all the time

Primality test

- Deterministic algorithm:

1. Divisibility algorithm—use as divisors all numbers smaller than $\sqrt{n}$
2. AKS algorithm– 2002, Agarwal, Kayal, Saxena polynomial bit operation time complexity

- Probabilistic algorithm:

1. Fermat test
2. Square root test: $\sqrt{1} = \pm 1 (\bmod\ p), \sqrt{1} = \pm 1 (\bmod\ n)$

And other values

3.Miller –Rabbin test: combination of 1 and 2

Factorization

- Factorization plays a very important role in the security of several public key cryptography
- Factorization method

1. Trial division (sieve of Eratosthenes) $p \leq \sqrt{n}$

Method is good if $n \leq 2^{10}$ inefficient and infeasible for factoring large integers, complexity exponential

2.Fermat factorization method: $n = x^2 - y^2 = ab$, $a = x + y, b = x - y$

Factorization

- Pollard p-1 method
- Pollard rho method
- Quadratic sieve : sieve procedure to find value $x^2 (\bmod\ n)$ used to factor integer more than 100 digits almost 300 bits .

Complexity subexponential, $O(e^c), c = (\ln n \ln \ln)^{1/2}$

- Number field sieve: base on find $x^2 \equiv y^2 (\bmod\ n)$

Complexity is $O(e^c), c = 2(\ln n)^{1/3}(\ln \ln n)^{2/3}$

Factorization

- Assume that there is a computer can perform 2^{30} (almost 1 billion) bit operations per second. What is the approximate time required for this computer to factor an integer of 100 digits using

(i) Quadratic sieve method (ii) number sieve

- A number with 100 digits has almost 300 bits

$n = 2^{300}, \ln 2^{300} = 207, \ln\ln 2^{300} = 5$. For quadratic sieve method

We have $(207)^{1/2}(5)^{1/2} = 14 \times 2.23 \quad 32$ this means we need e^{32} bit operations that can be done in $e^{32}/2^{30} \quad 20$ hours

- For N.F.S. : $(207)^{1/3} \times (5)^{2/3} = 6 \times 3 = 18$ this meand we need e^{18}

Bit operations that can be done in $e^{18}/2^{30} \quad 6$ seconds

Discrete logarithm

- Exponential and logarithm are inverse process
- Exponential $y = a^x$ Logarithm: $x = \log_a y$
- In cryptography a common modular operation is exponential $y = a^x (\bmod\ n)$, n has primitive roots.
- Discrete log $x = \mathrm{dlog}_a\ y\ (\bmod\ n)$

- Fast exponential is possible using square and multiply method.
- The main idea behind this method to treat the exponent as a binary number of bits.
- In cryptography if we use exponentiation to encrypt or decrypt, the adversary can use logarithm to attack .
- Exhaust search: write an algorithm that continuously calculate $y = a^x (\bmod\ n)$ until it find value of given y.
- This algorithm is very inefficient for large integers. The complexity is this algorithm is exponential.

Discrete log problem

- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find `i` such that `b = a^i (mod p)`
- this is written as $i = \mathrm{dlog}_a\ b\ (\mathrm{mod}\ p)$
- if a is a primitive root then it always exists, otherwise it may not, e.g.,
  - $x = \log_3 4 \bmod 13$ has no answer
  - $x = \log_2 3 \bmod 13 = 4$ by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem

Discrete log cryptography

- The following questions arises in this cryptosystem

1. Given an element a and a group G= $\langle Z_n^*, \times\rangle$ how to find the a is primitive root of G?

(i)We need to find $\phi(n)$, which is as difficult as factorization of n.

(ii)We need to check o(a)= $\phi(n)$,

2. Given a group G, how to check all primitive roots of G? this is more difficult than first task because we need to repeat part (ii) for all elements of G

3. Given G how to select a primitive root of G?

In cryptography the user choose the value of n so he/she knows the value of $\phi(n)$. To find primitive root user tries several elements until he finds the first one.

Discrete log problem

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

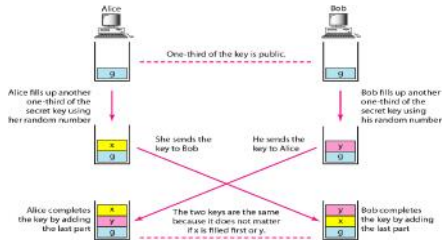
a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9

Diffie Hellman Key exchange

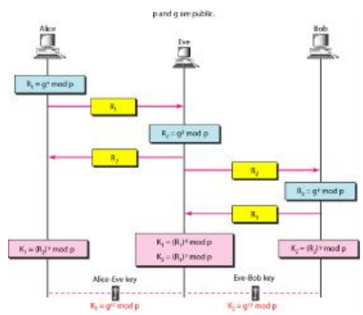
- The symmetric key in the Diffie Hillman protocol is K = g^{xy} mod p.



Diffie Hellman key exchange example

- *Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume g = 7 and p = 23. The steps are as follows:*
- *1. Alice chooses x = 3 and calculates $R_1 = 7^3$ mod 23 = 21.*
- *2. Bob chooses y = 6 and calculates $R_2 = 7^6$ mod 23 = 4.*
- *3. Alice sends the number 21 to Bob.*
- *4. Bob sends the number 4 to Alice.*
- *5. Alice calculates the symmetric key K = 4^3 mod 23 = 18.*
- *6. Bob calculates the symmetric key K = 21^6 mod 23 = 18.*
- *The value of K is the same for both Alice and Bob; g^{xy} mod p = 7^{18} mod 23 = 18.*

Man in the middle attack



Public key encryption scheme

- RSA public key cryptosystem
- Elgamal public key cryptosystem
- Digital signature based on public key cryptosystem

Public key cryptography requirements

- need a trapdoor one-way function
- one-way function has
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- a trap-door one-way function has
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- a practical public-key scheme depends on a suitable trap-door one-way function

Security of Public Key Schemes

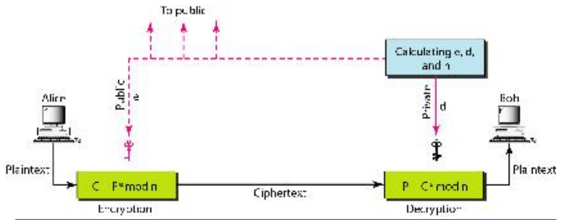
- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

Public key cryptography: RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - factorization takes $O(e^{\log n \log \log n})$ operations (hard)
  - Finding $\phi(n)$ is as difficult as factoring the number n

Public key cryptography: RSA

Select large primes randomly p and q. Find product n=p.q, calculate ø(n=(p-1)(q-1)



In RSA, *e* and *n* are announced to the public; *d* and Φ are kept secret.

1<e<ø(n), gcd(e,ø(n))=1,   e.d ≡ 1 mod ø(n) and 0≤d≤n

Security due to cost of factoring large numbers, Finding $\phi(n)$ is as difficult as factoring the number n

RSA: Example

- *Bob chooses 7 and 11 as p and q and calculates n = 7 · 11 = 77. The value of F = (7 − 1) (11 − 1) or 60. Now he chooses two keys, e and d. If he chooses e to be 13, then d is 37. Now imagine Alice sends the plaintext 5 to Bob. She uses the public key 13 to encrypt 5.*

Plaintext: 5
$C = 5^{13} = 26 \bmod 77$
Ciphertext: 26

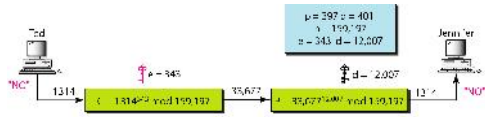
RSA: Example

Jennifer creates a pair of keys for herself. She chooses p = 397 and q = 401. She calculates n = 159,197 and Φ *= 396 · 400 = 158,400. She then chooses e = 343 and d = 12,007. Show how Ted can send a message to Jennifer if he knows e and n.*

- *Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25) with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is 1314^{343} = 33,677 mod 159,197. Jennifer receives the message 33,677 and uses the decryption key d to decipher it as $33{,}677^{12{,}007}$ = 1314 mod 159,197. Jennifer then decodes 1314 as the message "NO". Figure 30.25 shows the process.*

RSA: Example

- currently assume 1024-2048 bit RSA is secure



RSA: Realistic example

- *Let us give a realistic example. We randomly chose an integer of 512 bits. The integer p is a 159-digit number.*

p = 96130345313583504574191581280615427909309845594996215822583150879647940 45505647063849125716018034750312098666606492420191808780667421096063354 219926661209

The integer q is 160-digit number

q = 12060191957231446918276794204450896001555925054637033936061798321731482 14848376465921538945320917522527322683010712069560460251388714552496900 0359660045617

RSA: Realistic example

We calculate n=pq. It has 309 digits:

n = 11593504173967614968892509864615887523771457375454144775485526137614788 54083263508172768788159683251684688493006254857641112501624145523391829 27162507656772727460097082714127730434960500556347274566628060099924037 10299142447229221577279853172703383938133469268413732762200096667667183 1831088373420823444370953

We calculate Φ *. It has 309 digits:*

ϕ = 11593504173967614968892509864615887523771457375454144775485526137614788 54083263508172768788159683251684688493006254857641112501624145523391829 27162507656751054233608492916752034482627988117554787657013923444405716 98958172819609822636107546721186461217135910735864061400888517026537727 7264467341066243857664128

RSA: Realistic example

We choose e = 35,535. We then find d.

e = 35535

d = 58008302860037763936093661289677917594669062089650962180422866111380593852 82235873170628691003002171085904433840217072986908760061153062025249598844 48047568240966247081485817130463240644077704833134010850947385295645071936 77406119732655742423721761767462077637164207600337085333288532144708859551 36670294831

Alice wants to send the message "THIS IS A TEST" which can be changed to a numeric value by using the 00–26 encoding scheme (26 is the space character).

RSA: Realistic example

The ciphertext calculated by Alice is $C = P^e$, which is.

C = 47530912364622682720636555061054518094237179607049171652323924305445296 06131993285666178434183591141511974112520056829797945717360361012782188 47892741566090480023507190715277185914975188465888632101148354103361657 89846796838676373376577746562507928052114814184404814184430812773059004 692874248559166462108656

Bob can recover the plaintext from the ciphertext by using $P = C^d$, which is

P = 1907081826081826002619041819

RSA: Realistic example

The ciphertext calculated by Alice is C = P^e, which is.

C = 4753091236462268272063655506105451809423717960704917165232392430544529
6061319932856661784341835911415119741125200568297979457173603610127821
8847892741566090480023507190715277185914975188465888632101148354103361
6578984679683867637337657774656250792805211481418440481418443081277305
9004692874248559166462108656

Bob can recover the plaintext from the ciphertext by using P = C^d, which is

P = 1907081826081826002619041819

The recovered plaintext is THIS IS A TEST after decoding.

Elgamal Cryptosystem

Presented in 1984 by Tather Elgamal

Key aspects:

- Based on the Discrete Logarithm problem
- Randomized encryption

Application:

- Establishing a secure channel for key sharing
- Encrypting messages

Elgamal Cryptosystem: key generation

Participant A generates the public/private key pair

1. Generate large prime p and generator g of the multiplicative Group $\mathbb{Z}_p^*$ pf of the integers modulo p.
2. Select a random integer a, $1 \leq a \leq p-2$, and compute $g^a \bmod p$.
3. A's Public key is (p, g, g^a); A's Private key is a.

Elgamal Cryptosystem: Encryption and decryption process

Participant B encrypts a message m to A

1. Obtain A's authentic public key (p, g, g^a).
2. Represent the message as integers m in the range $\{0, 1, \ldots, p-1\}$.
3. Select a random integer k, $1 \leq k \leq p-2$
4. Compute $\gamma = g^k \bmod p$ and $\delta = m * (g^a)^k$
5. Send cipher text $c = (\gamma, \delta)$ to A

Participant A receives encrypted message m from B

1. Use private key a to compute $(\gamma^{p-1-a}) \bmod p$.
   Note: $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$
2. Recover m by computing $(\gamma^{-a}) * \delta \bmod p$.