CS557: Cryptography

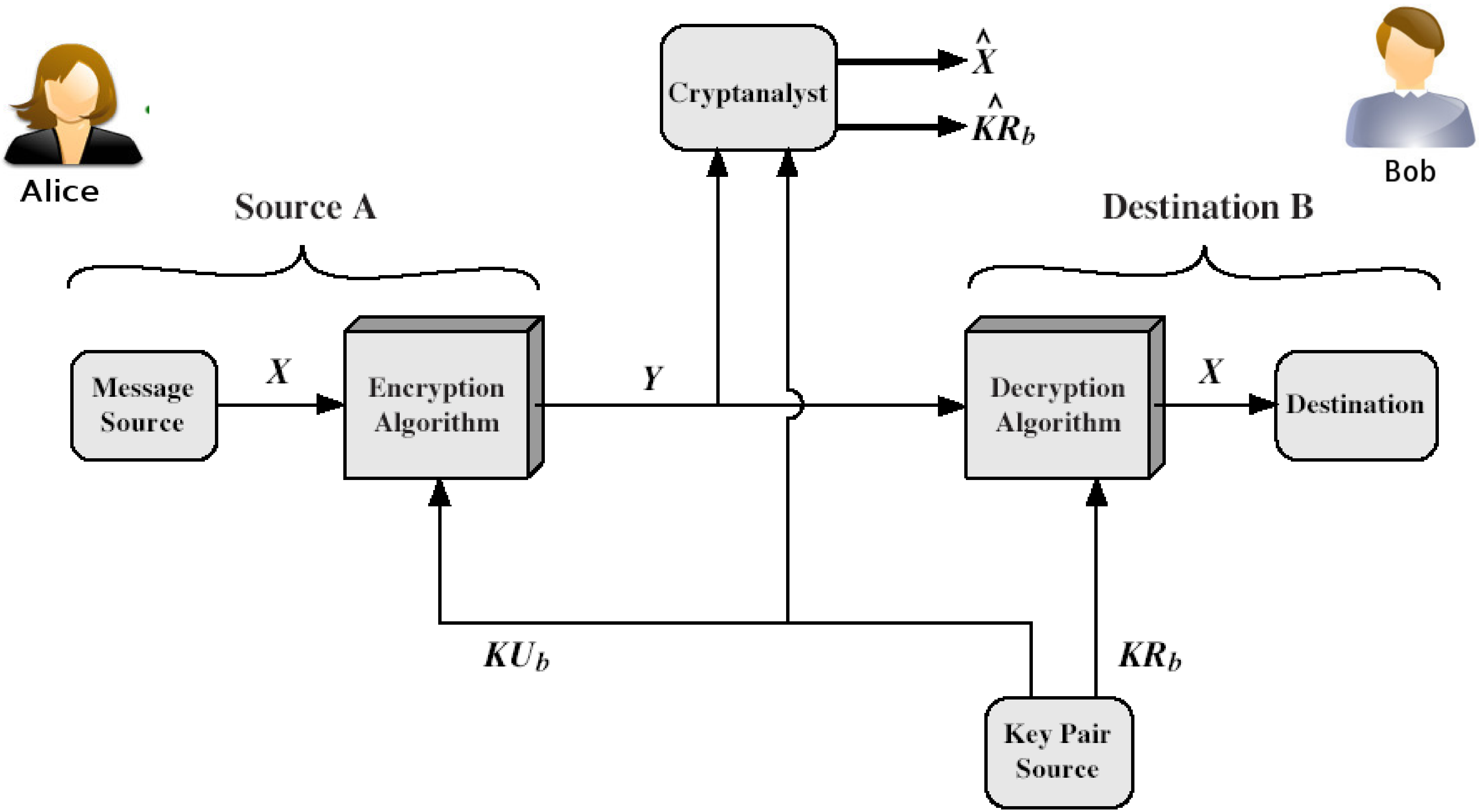
Public-key Cryptography-II

S. Tripathy
IIT Patna

Present Class

- Public Key Cryptography
  - Public Key Encryption
    - RSA

Public-Key Cryptosystems (confidentiality)



Public-Key Cryptosystem: Secrecy

RSA

- Key Setup
  - each user generates a public/private key pair by:
    - selecting two large primes at random : p, q
    - computing their system modulus $N=p.q$
    - note $\varnothing(N)=(p-1)(q-1)$
  - select at random the encryption key e
  - where $1<e<\varnothing(N)$, $\gcd(e,\varnothing(N))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod \varnothing(N)$ and $0\leq d\leq N$
- publish their public encryption key: KU={e,N}
- keep secret private decryption key: KR={d,p,q}

- to encrypt a message M the sender:
  - obtains **public key** of recipient $KU=\{e,N\}$
  - computes: $C=M^e \bmod N$, where $0\leq M<N$
- to decrypt the ciphertext C the owner:
  - uses their private key $KR=\{d,p,q\}$
  - computes: $M=C^d \bmod N$

RSA Encryption is one-way trapdoor

- Now $D_d [E_e[x]] = x$
  $E[x]$ and $D[y]$ can be computed efficiently if keys are known

- $E^{-1}[y]$cannot be computed efficiently without knowledge of the (private) decryption key d.

- *Also, it should be possible to select keys reasonably efficiently. Efficiency requirements are less stringent since it has not to be done too often.*

RSA Key Generation

- users of RSA must:
  1. determine two primes at random: p, q
     - primes p,q must not be easily derived from modulus $N=p.q$
       - means must be sufficiently large
     - Primes are dense so choose randomly.
     - Probabilistic primality testing methods known.

  2. select either e or d and compute the other
     - typically guess and use probabilistic test
     - exponents e, d are inverses, so use Inverse algorithm to compute the other (Extended Euclidean algorithm)

Prime numbers

- Definitions:
  - A Prime number is an integer that has no integer factors other than 1 and itself.
  - Otherwise, it is called composite number.

- A primality testing is a test to determine whether or not a given number is prime, as opposed to actually decomposing the number into its constituent prime factors (which is known as prime factorization)

The Largest Known Prime

- A Mersenne prime is a prime of the form

$$2^q - 1$$

- The largest known prime as of March 2007 is the 44th known Mersenne prime

$$p = 2^{32582657} - 1$$

- Which has 9,808,358 decimal digits
  - This would take over 2000 pages to print, assuming a page contains 60 lines with 80 characters per line.
- **Largest** known **prime number**:As of Dec 2018,
  - the **largest** known **prime number** is $2^{82589933} - 1$, a **number** with 24,862,048 digits. It was found by the Great Internet Mersenne **Prime** Search (GIMPS).

Primality Test

- The primality test provides the probability of whether or not a large number is prime.
- Several theorems including Fermat's theorem provide idea of primality test.
- Cryptography schemes such as RSA algorithm heavily based on primality test.

1. Square Root Compositeness Theorem

   +

2. Fermat's Theorem =

3. Miller-Rabin Compositeness Test

A Naïve Algorithm (Primality Test)

- A Naïve Algorithm
  - Pick any integer P that is greater than 2.
  - Try to divide P by all odd integers starting from 3 to square root of P.
  - If P is divisible by any one of these odd integers, we can conclude that P is composite.
  - The worst case is that we have to go through all odd number testing cases up to square root of P.
  - Time complexity is O(square root of N)

Square Root Compositeness (Primality Test)

The Square Root Compositeness theorem gives a way to factor certain composite numbers

Given integers n, x, and y:

$$If\ x^2 \equiv y^2 \pmod{n}, but\ x \neq \pm y \pmod{n}$$

Then n is composite

Ex.: n=21?

x=2, y=16

Fermat’s Theorem (Primality Test)

- Fermat’s Theorem
  - Given that P is an integer that we would like to test that it is either a PRIME or not.
  - And A is another integer that is greater than zero and less than P.
  - From Fermat’s Theorem, if P is a PRIME, it will satisfy this two equalities:
    - A^(p-1) = 1(mod P) or A^(p-1)mod P = 1
    - A^P = A(mod P) or A^P mod P = A

  - For instances, if P = 341, will P be PRIME?

    -> from previous equalities, we would be able to obtain that:

    2^(341-1)mod 341 = 1, if A = 2

  - However, if we choose A equal to 3:
  - 3^(341-1)mod 341 = 56 !!!!!!!!!
    - That means Fermat’s Theorem is not true in this case!

Review: Fermat can be used to test for compositeness, but doesn't give factors

- Fermat's little theorem:
  - If n is prime and doesn't divide a, then $a^{n-1} \equiv 1 \pmod{n}$

- Contrapositive:
  - If $a^{n-1} \neq 1 \pmod{n}$ then n is composite

- In practice,
  - If $a^{n-1} \equiv 1 \pmod{n}$ then n is probably prime
  - Rare counterexamples called *pseudoprimes*

A is… \ a^{n-1}	=1	≠1
Prime	Usually true	None
Composite	Rare pseudoprime	All

Rabin-Miller’s (Primality Test)

- Rabin-Miller’s Probabilistic Primality Algorithm
  - The Rabin-Miller’s Probabilistic Primality test was by Rabin, based on Miller’s idea. This algorithm provides a fast method of determining of primality of a number with a controllably small probability of error.
  - Given (b, n), where n is the number to be tested for primality, and b is randomly chosen in [1, n-1].
    - Let n-1 = (2^q)*m, where m is an odd integer.
      - b^m = 1(mod n)
      - $\exists i \in$[0, q-1] such that b^((2^i) m)= -1(mod n)
  - If the testing number satisfies either cases, it will be said as “inconclusive”or Probable prime number

Ex.:

- Consider the Carmichael number n = 561.
  - Then n -1 =560 = 2 ^4 *35.
  - m=35,
  - For a = 2, we get

  - 2 ^35 =263 ≠ 1 mod 561
  - 2^35 = 263 ≠ -1 mod 561

  - 2 ^(2*35=)70 =166 mod 561
  - 2 ^(4*35=)140 = 67 mod 561

  - 2 ^(8*35=)280 = 1 mod 561. so composite
-
- Thus, 2 is a Miller-Rabin witness for compositeness of n =561

RSA Security

- Three major approaches to attacking RSA:
  - brute force key search:
    - infeasible given size of numbers
  - mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
  - timing attacks (on running of decryption)

Factoring Problem

- mathematical approach to find d takes 3 forms:
    - factor N=p.q, hence find ø(N) and then d
    - determine ø(N) directly and find d
    - find d directly
- currently believe all equivalent to factoring
    - barring dramatic breakthrough 1024+ bit RSA secure
        - ensure p, q of similar size and matching other constraints

Security lies on Factoring

- have seen slow improvements over the years
- biggest improvement comes from improved algorithm
    - "Quadratic Sieve" to "Generalized Number Field Sieve"

- Rivest's estimation in 1977: to factor a 129-digit number requires 40000 trillion years
    - That was the first RSA challenge: RSA-129, award $100.
    - RSA-129 was factored in 1994 - Atkins, Graff, Lenstra, Leland + 600 volunteers using 1600 computers for about one year.
    - RSA-challenge that was factored: RSA-640 in Dec. 2005.
    - RSA 704 was factored in July 2, 2012
    - RSA 829 in 2020
- Current recommendations
    - Individual users: n should have 768 bits (231 digits)
    - Organizations (short term): 1024 bits (308 digits)
    - Organizations (long term): 2048 bits (616 digits)

Complexity of Factoring Problem

- Trial division
  - Complexity √n
- Pollard p-1 method

input : an integer *n* , and a prespecified "bound" B

output : factors of n

$$a \leftarrow 2$$

$$\text{for } \; j \leftarrow 2 \; \text{ to } \; B$$

$$\quad \text{do } \; a \leftarrow a^j \mod n$$

$$d \leftarrow \gcd(a-1, n)$$

$$\text{if } \; 1 < d < n$$

$$\quad \text{then } \; \text{return}(d)$$

$$\quad \text{else } \; \text{return}(\text{"}\mathit{failure}\text{"})$$

- Thanks