Lecture 3: Public Key Cryptography and Digital Signature

Network Security – 2025

Professor Anh T. Pham

Contents

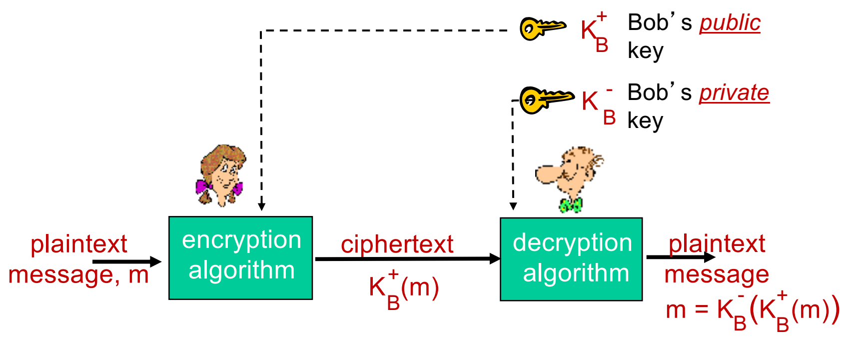
Basic Principles

- The concept of public-key cryptography evolved from an attempt to attack **two of the most difficult problems** associated with symmetric encryption:
  - **Key distribution:** How can Alice and Bob can agree on the secret key?
  - **Digital signatures**: Security spreads to commercial apps. How can parties verify each other, even met physically?
- Whitfield Diffie and Martin Hellman from Stanford University achieved a breakthrough in 1976 by coming up with a method that addressed both problems and was **radically different** from all previous approaches to cryptography

Misconceptions Concerning Public-Key Encryption

- Public-key encryption is **more secure** from cryptanalysis than symmetric encryption
  - **WRONG:** ***length of key*** and computational work!
- Public-key encryption is a general-purpose technique that has made **symmetric encryption obsolete**
  - **WRONG**: symmetric encryption is ***more efficient***, less overhead!
- There is a feeling that **key distribution is trivial** when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption
  - **WRONG:** some (not simpler) form of protocol is STILL needed!

Public-Key Cryptosystem



Six Components of PKC

- **Plaintext:** same, the readable message or data that is fed into the algorithm as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext
- **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact trans- formations performed by the encryption algorithm depend on the public or private key that is provided as input.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext

Requirements

1 need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2 given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: Modular Arithmetic

- x mod n = remainder of x when divide by n
- Facts:

  [(a mod n) + (b mod n)] mod n = (a+b) mod n

  [(a mod n) - (b mod n)] mod n = (a-b) mod n

  [(a mod n) * (b mod n)] mod n = (a*b) mod n
- Thus

  $(a \bmod n)^d \bmod n = a^d \bmod n$

- Example: x=14, n=10, d=2:
  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA: Getting Ready

- Message: just a bit pattern
- Bit pattern can be uniquely represented by an integer number
- Thus, encrypting a message is equivalent to **encrypting a number**.

example:

- m= 10010001 . This message is uniquely represented by the decimal number 145.
- To encrypt ***m***, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating public/private key pair

1. Choose two large prime numbers p, q. (e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose e (with $e<n$) that has no common factors with z (e, z are "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).
5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: Encryption, Decryption

6. given (n,e) and (n,d) as computed above
7. to encrypt message m $(<n)$, compute

   $c = m^e \bmod n$

8. to decrypt received bit pattern, c, compute

   $m = c^d \bmod n$

$$\boxed{\textit{magic happens!} \quad m = (\underbrace{m^e \bmod n}_{c})^d \bmod n}$$

RSA Example

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.

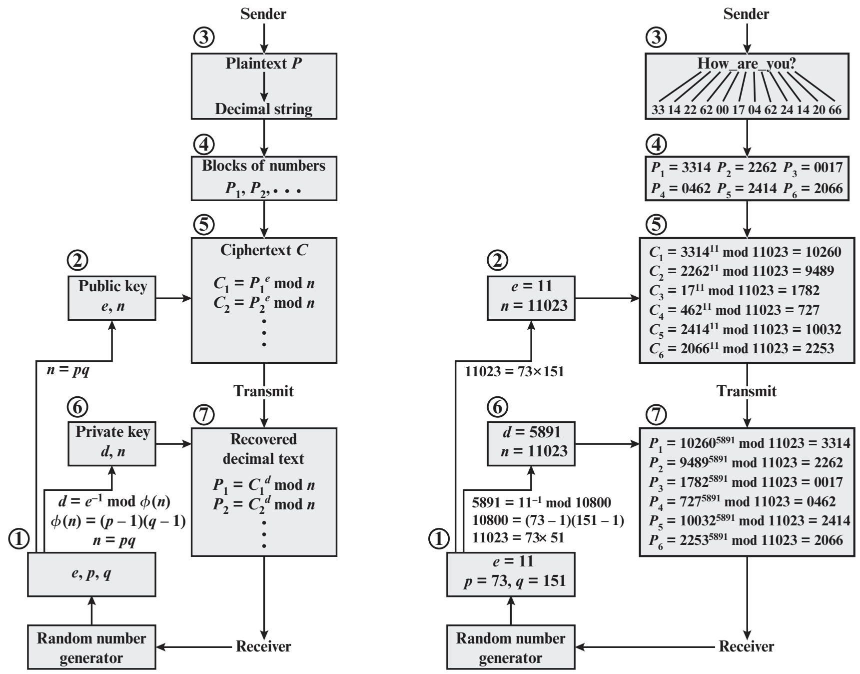
$e=5$ (so e, z relatively prime).

$d=29$ (so $ed-1$ exactly divisible by z).

encrypting 8-bit messages.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	00001000	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	12



Why does RSA work?

- Must show that $c^d \bmod n = m$
  where $c = m^e \bmod n$
- **Fact:** for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where n= pq and z = (p-1)(q-1)
- thus,
  $c^d \bmod n = (m^e \bmod n)^d \bmod n$
  $= m^{ed} \bmod n$
  $= m^{(ed \bmod z)} \bmod n$
  $= m^1 \bmod n$
  $= m$

RSA: Another Important Property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{} = m = \underbrace{K_B^+(K_B^-(m))}_{}$$

use public key first, followed by private key

use private key first, followed by public key

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$

$= m^{de} \bmod n$

$= (m^d \bmod n)^e \bmod n$

Why is RSA secure?

- Suppose you know Bob's public key (n,e). How hard is it **to determine d**?
- Essentially need to find factors of **n** without knowing the two factors **p** and **q**

fact: factoring a big number is hard

Why Factoring Large Number is Hard?

- We want to factor n =p x q, i.e. we need to find p and q, we need to try every prime number from 1 to n
- For example
  - n = 35
  - n = 77
  - n = 143
  - n = 3599

- Mathematicians tell us that the number of prime numbers between one and n is approximately n / ln(n)
- If we use 128-bit public key, the key would be a number between

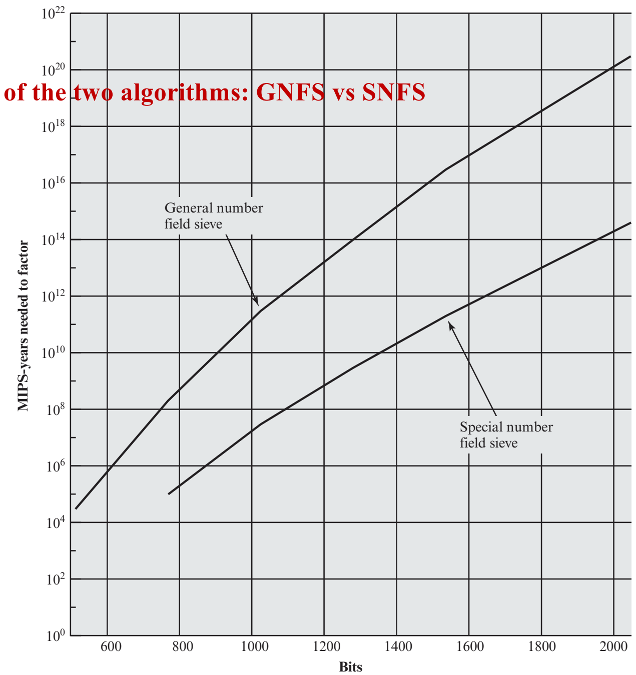
  1 and

  340,282,366,920,938,000,000,000,000,000,000,000,000
- The number of prime number is approximately

  3,835,341,275,459,350,000,000,000,000,000,000,000

Factoring Problem

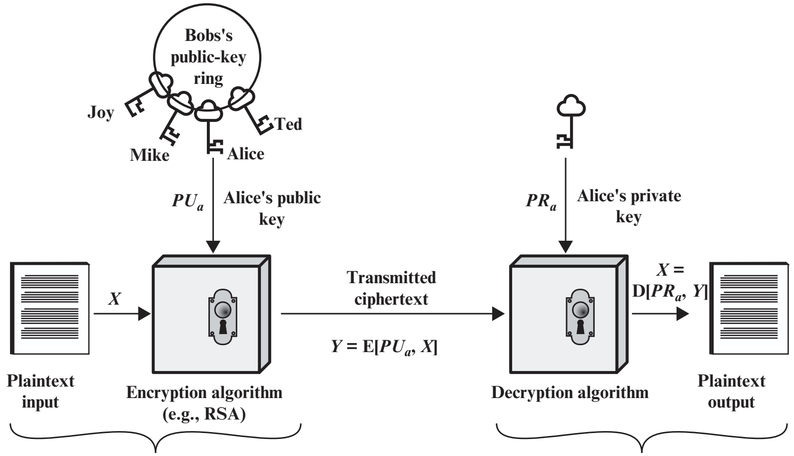
performance of the two algorithms: GNFS vs SNFS



Applications of Public-key

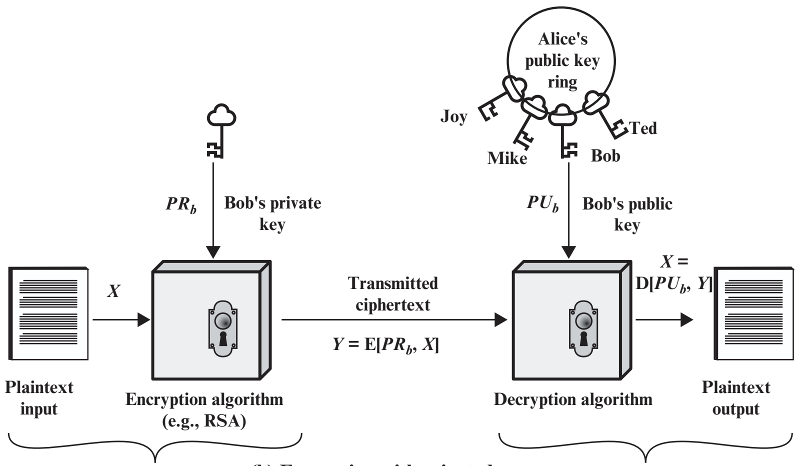
- **Encryption/decryption:** The sender encrypts a message with the recipient's public key.
- **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
  - More on this later
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties
  - More on this later

Encryption with Public Key



(a) Encryption with public key

Encryption with Private Key



(b) Encryption with private key

Public Key with Symmetric Key (DES)

- Exponentiation in RSA is **computationally intensive**
- DES is **at least 100 times faster** than RSA

→ Practically, we use public key cryptography to establish secure connection, then establish second key – symmetric session key – for encrypting data

session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- Once both have K_S, they use symmetric key cryptography

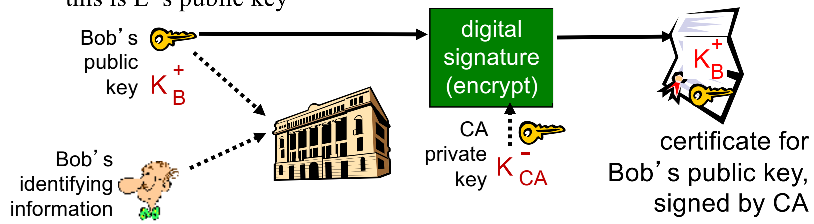
Public-key Certification

- Motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, **but says it's Bob's public key**
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

How can we verify that Bob's public key is only the one Bob has?

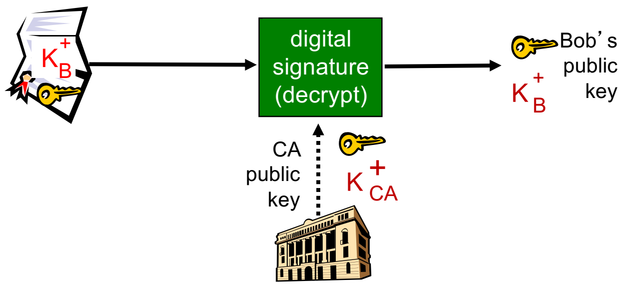
Certification Authorities

- *certification authority (CA):* binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



Certification Authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



Digital Signatures

Digital signatures

cryptographic technique analogous to hand-written signatures:

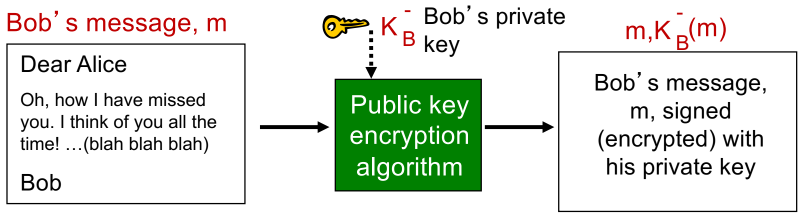
- sender (Bob) digitally signs document, establishing he is document owner/creator.

- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital Signatures

simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Digital Signatures

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

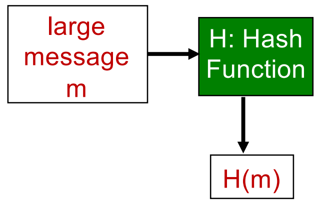
Alice thus verifies that:

- ✓ Bob signed m
- ✓ no one else signed m
- ✓ Bob signed m and not m'

non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Message digests



computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy- to-compute digital "fingerprint"

- apply hash function H to *m*, get fixed size message digest, *H(m)*.

Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x, computationally infeasible to find m such that x = H(m)

Digital Signature = Signed Message Digest

Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message: