Fundamentals of Information & Network Security ECE 471/571

Lecture #17: PKC and RSA

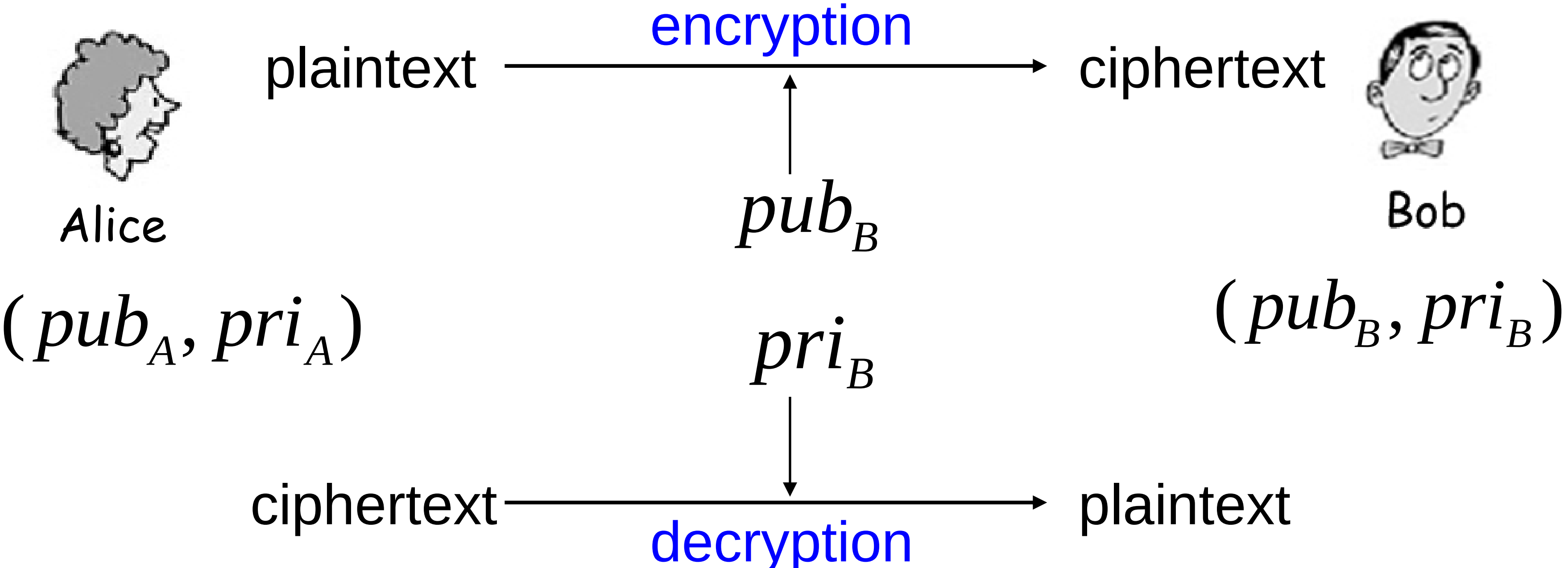
Instructor: Ming Li

Dept of Electrical and Computer Engineering

University of Arizona

Public Key Cryptography (PKC)

- Invented in 1975 (A.k.a. Asymmetric cryptography)
- Involve two keys: *Private key* (*d*): must keep secret; *Public key* (*e*): made public
- Encryption/decryption: encryption can be done by everyone using the recipient’s public key, decryption can be done only by the recipient with his/her private key



plaintext $\xrightarrow{\text{encryption}}$ ciphertext

pub_B

pri_B

ciphertext $\xrightarrow[\text{decryption}]{}$ plaintext

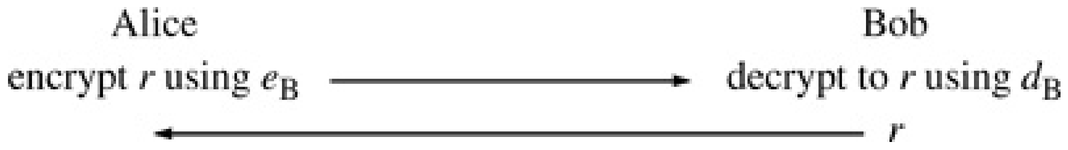
Alice (pub_A, pri_A)

Bob (pub_B, pri_B)

PKC Applications

Everything that SKC does can be done by PKC!

- Transmitting over an insecure channel
- Secure storage over insecure media
- Authentication



Alice

encrypt r using e_B

Bob

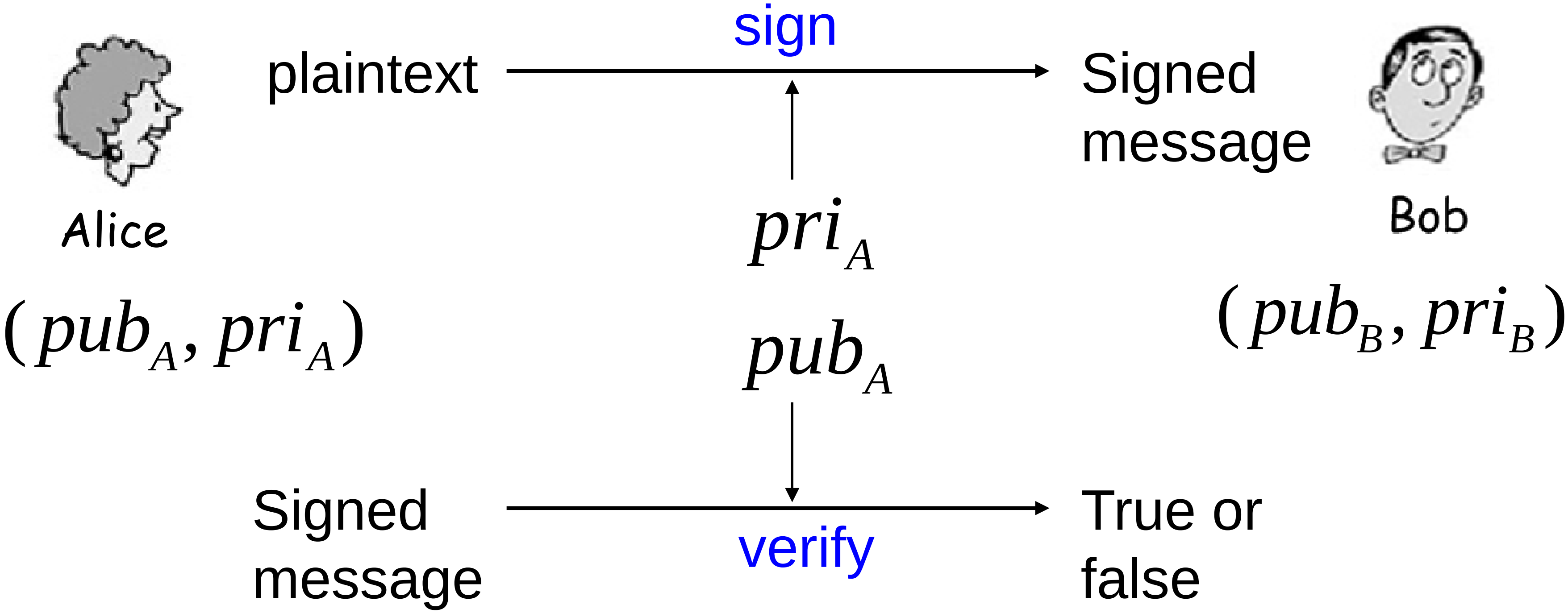
decrypt to r using d_B

r

- Key exchange: establish a shared session key with PKC

PKC Applications (Cont’d)

- Digital signature: non-repudiation



Alice

(pub_A, pri_A)

plaintext

sign

Signed message

Bob

(pub_B, pri_B)

pri_A

pub_A

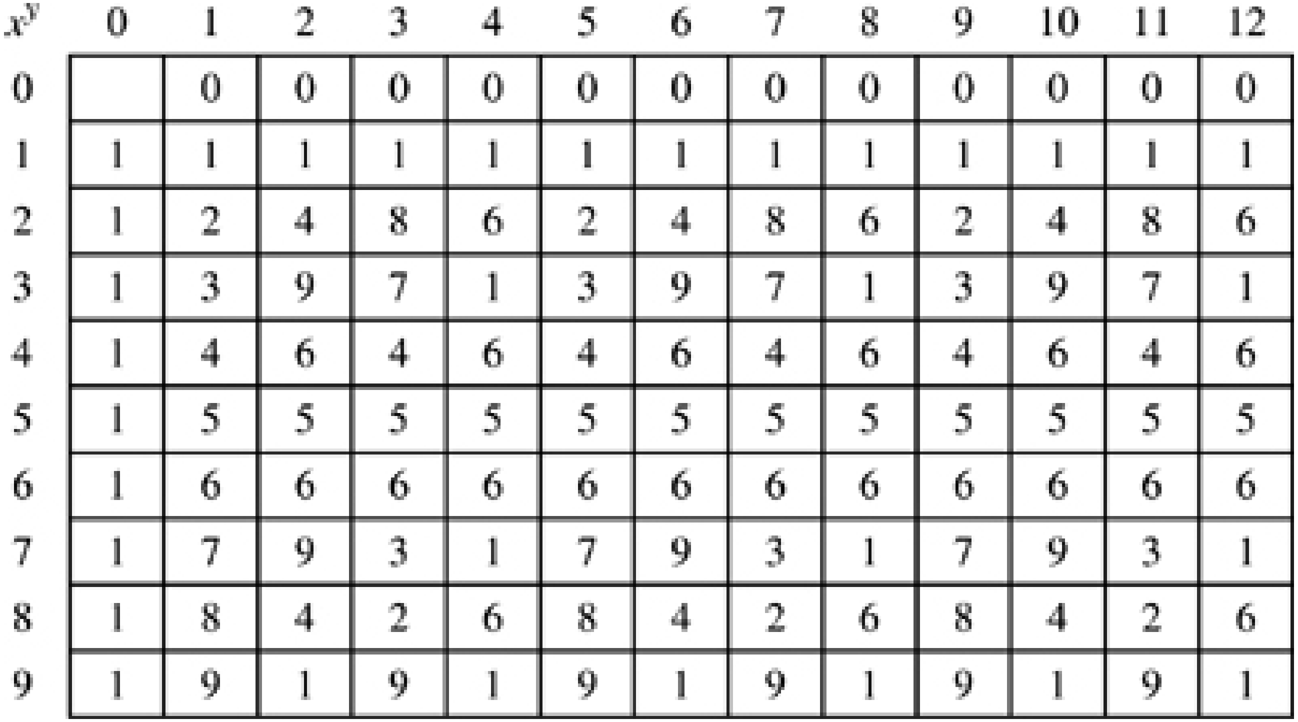
Signed message

verify

True or false

Modular Exponentiation (1)

- Example:
  - $4^6 = 4096$, $4^6 = 6 \bmod 10$
  - $x^y \bmod n \neq x^{y+n} \bmod n$
  - Group: Z^*_n: closed under mult. Mod n



x^y	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

Modular Exponentiation (2)

- $x^y \bmod n \equiv x^{(y \bmod \phi(n))} \bmod n$
  - Example: n=10, $\phi(n)=4$, {1,3,7,9}
- If $y \equiv 1 \bmod \phi(n)$,

  then for any number x,

  $x^y \bmod n \equiv x \bmod n$
- Examples

- Euler's theorem
- Fermat's theorem

RSA

- Named after Rivest, Shamir, and Adleman
- private key / public key, use one to encrypt and the other to decrypt
- Key length: variable (512 bits)

  Input: variable (< key length)

  Output: the length of the key
- Advantage: Easy key management
- Disadvantage: Much slower than secret key algorithms

- Mostly used for short message encryption, digital signature, and secret key exchange

RSA Algorithm

- Choose two large primes, p and q,

  $n = p \times q$ and $\phi(n) = (p - 1)(q - 1)$

- Choose e that is relatively prime to $\phi(n)$
- By Euclid's algorithm, find d that is the multiplicative inverse of e mod $\phi(n)$, i.e.,

  $$e \times d \equiv 1 \bmod \phi(n)$$

- Let <e, n> be the public key and <d, n> the private key

Encryption/Decryption

- Encryption w/public key <e, n> :

$$c = m^{e} \bmod n$$

- Decryption w/private key <d, n> :

$$m = c^{d} \bmod n$$

$$\begin{aligned} c^{d} \bmod n &\equiv (m^{e} \bmod n)^{d} \bmod n \\ &\equiv (m^{e})^{d} \bmod n \\ &\equiv m \bmod n \\ &\equiv m \end{aligned}$$

Let’s walk through an example!

Example

- $p = 11$, $q = 7$, $n = 77$, $\Phi(n) = 60$
- $d = 13$, $e = 37$ ($ed = 481$; $ed \bmod 60 = 1$)

- Let $M = 15$. Then $C \equiv M^e \bmod n$
  - $C \equiv 15^{37} \pmod{77} = 71$

- $M \equiv C^d \bmod n$
  - $M \equiv 71^{13} \pmod{77} = 15$

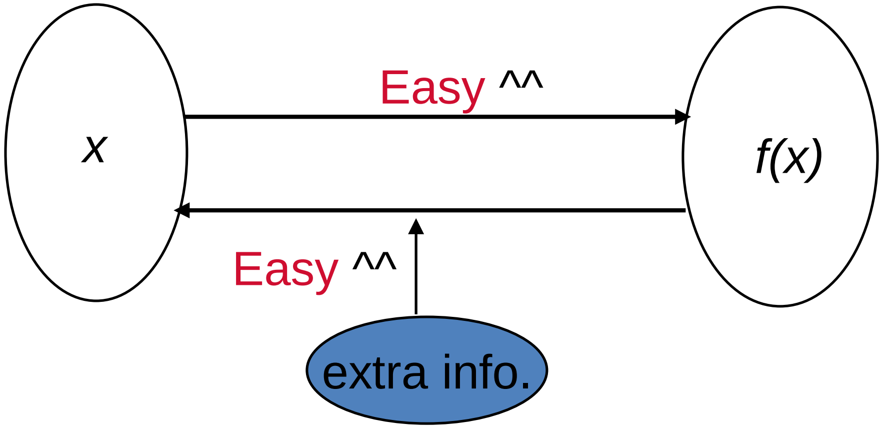
RSA Signature

- Sign with private key d: $S = m^d \bmod n$
- Verify with public key <e, n>: $m = S^e \bmod n$

$$\begin{aligned} S^e \bmod n &= (m^d \bmod n)^e \bmod n \\ &= (m^d)^e \bmod n \\ &= m \bmod n \\ &= m \end{aligned}$$

Concepts of PKC

- Trapdoor one-way function
  - ✓ Given *x*, easy to compute *f(x)*
  - ✓ Given *y*, difficult to compute $f^{-1}(y)$ in general
  - ✓ Easy to compute $f^{-1}(y)$ for given *y* to only who knows certain information(which we call trapdoor information)



Why is RSA secure?

- Given n, it is hard to factor it to get p and q.
  - If you can factor quickly, you can break RSA.
- RSA misuse
  - Alice uses Bob's public key to encrypt a message to Bob. If Frank knows the message is one of many possible messages, he can use the same public key to compute and compare the ciphertexts to find the message
  - (Solution?)