Lecture 3

A point to remember

- Symmetric-Key Cryptography: Sender and Receiver both know the secret key. The encryption and decryption algorithms **need not be identical**.

Public Key Cryptography

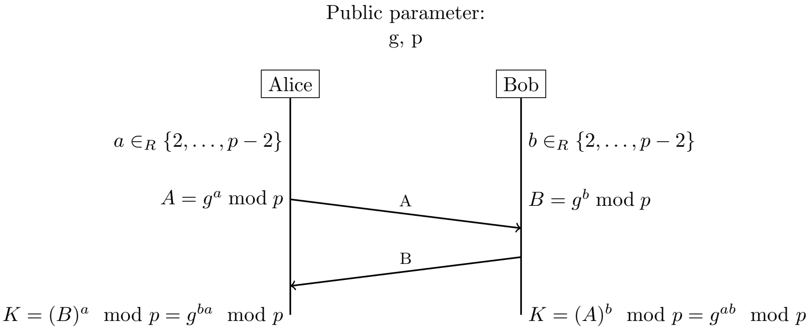
University of Birmingham

Outline of This Lecture

- Diffie-Hellman Key Exchange
- The Setup of Public Key Cryptography
- RSA Encryption and Signatures
- Public Key Certificates

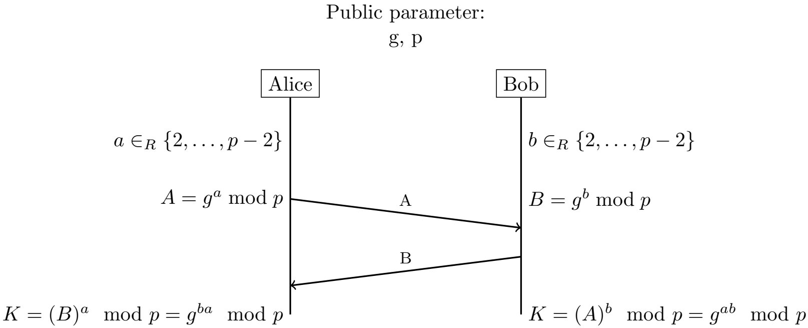
Secure Key Exchange

Recall p is a large prime, $g < p$



Example: Suppose $p = 13$ and $g = 7$.

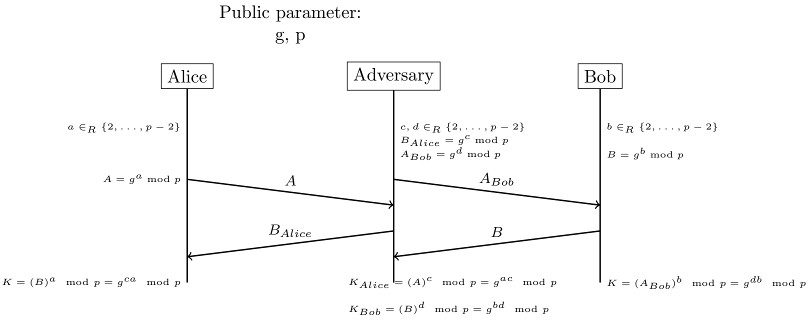
Secure Key Exchange: Idea of Public-Key Cryptography



Public Keys

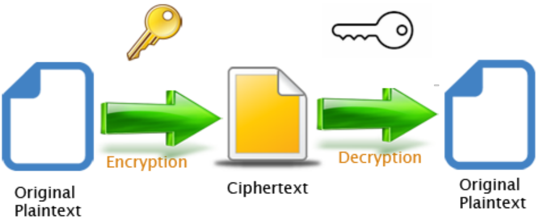
$g^a \mod p$ and $g^b \mod p$ can be called public keys. The secrets a and b are private keys.

Public Keys needs to be authenticated



Encryption and Authentication using Public Keys

Encryption using Public-Key



- Encryption uses receiver's Public-Key
- Decryption uses receiver's Private-Key

Encryption using RSA

Textbook RSA scheme

- Three Algorithms (Gen, Enc, Dec)
  - Gen: on input a security parameter λ.
    - Generate two distinct primes p and q of same bit-size λ
    - Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$
    - Choose at random an integer e $(1 < e < \phi(N))$ such that $\gcd(e, \phi(N)) = 1$
    - Let $\mathbb{Z}_N^*$={x | 0 <x<N and gcd(x,N)=1}
    - Compute d such that $e \cdot d \equiv 1 \ (mod\ \phi(N))$
    - Public key $PK = (e, N)$. The private key $SK = e, d, N$

gcd:greatest common divisor.

Encryption using RSA

Textbook RSA scheme

- $Enc(PK, m)$: On input an element $m \in \mathbb{Z}_N^*$ and the public key $PK = (e, N)$ compute
  - $c = m^e \ (mod\ N)$
- $Dec(SK, c)$: On input an element $c \in \mathbb{Z}_N^*$ and the private key $SK = (e, d, N)$ compute
  - $m = c^d \ (mod\ N)$

RSA Keygen:Example

- Let $p = 5, q = 11$
- $N = 55, \phi(N) = 4 \times 10 = 40$
- Suppose $e = 7$. Then $d = 23$ as $7 \times 23 = 161 \equiv 1 \bmod 40$
- PK=$(7, 55)$, SK=$(23, 55)$

Notes on RSA

- RSA security depends on hardness of finding d from e, N; Related to hardness of factoring of N.
- The textbook algorithms are deterministic. In practice, some random padding is used.
- Shor's quantum algorithm can solve factoring in polynomial time. However, a quantum computer of required capacity is still quite far away in the future.

Notes on RSA

- RSA security depends on hardness of finding d from e, N; Related to hardness of factoring of N.
- The textbook algorithms are deterministic. In practice, some random padding is used.
- Shor's quantum algorithm can solve factoring in polynomial time. However, a quantum computer of required capacity is still quite far away in the future.
- Wikipedia says any $m < N$ would work. Strictly speaking, it is required that gcd$(m, N) = 1$. On the other hand, finding a $m < N$ such that gcd$(m, N) > 1$ will lead to finding p or q, and breaking the system.

Digital Signatures

Signature Scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$

$$1^k \rightarrow \mathsf{Gen} \rightarrow (pk, sk)$$

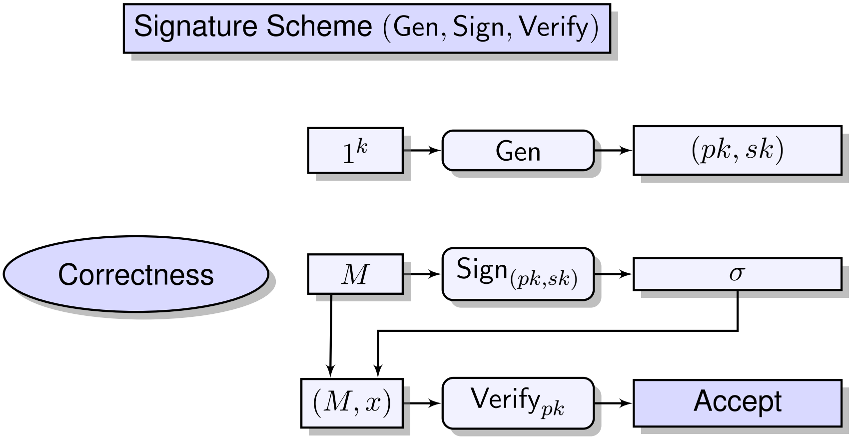
$$M \rightarrow \mathsf{Sign}_{(pk,sk)} \rightarrow \sigma$$

$$(M, x) \rightarrow \mathsf{Verify}_{pk} \rightarrow \text{Accept / Reject}$$

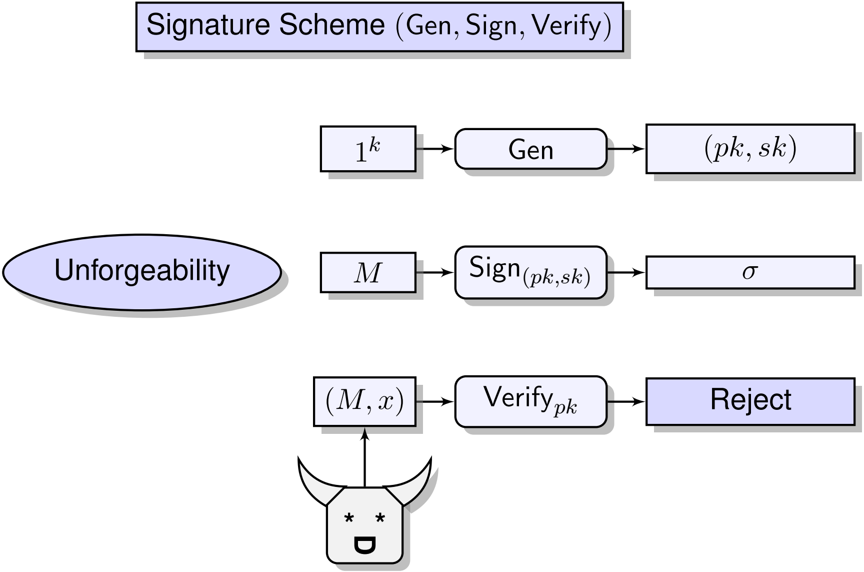
Required Properties:

- Correctness
- Unforgeability

Digital Signatures



Digital Signatures



Signature using RSA

Procedure $\mathsf{Keygen}(1^\lambda)$

Choose two random λ-bit primes p and q
$n = p \cdot q$
$\phi = (p-1)(q-1)$
Select e such that
$1 < e < \phi$ and $\gcd(e, \phi) = 1$
Compute d such that
$1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$
Set $PK = (e, n)$
Set $SK = (d, n)$
return (PK, SK)

Procedure $\mathsf{Sign}(SK, m)$

// We assume $m \in \mathbb{Z}_n^*$
$\sigma = H(m)^d \bmod n$
return σ

Procedure $\mathsf{Verify}(PK, m, \sigma)$

if $H(m) = \sigma^e \bmod n$
  return Accept
else return Reject

Signature using RSA

Procedure $\mathsf{Keygen}(1^\lambda)$

- Choose two random λ-bit primes p and q
- $n = p \cdot q$
- $\phi = (p-1)(q-1)$
- Select e such that $1 < e < \phi$ and $\gcd(e, \phi) = 1$
- Compute d such that $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$
- Set $PK = (e, n)$
- Set $SK = (d, n)$
- **return** (PK, SK)

Procedure $\mathsf{Sign}(SK, m)$

// We assume $m \in \mathbb{Z}_n^*$

- $\sigma = H(m)^d \bmod n$
- **return** σ

Procedure $\mathsf{Verify}(PK, m, \sigma)$

- **if** $H(m) = \sigma^e \bmod n$
  - **return** Accept
- **else return** Reject

Why use H

Possible attack without H.

Authenticating Public Keys

Certificates of Public-Key. Demo in class.