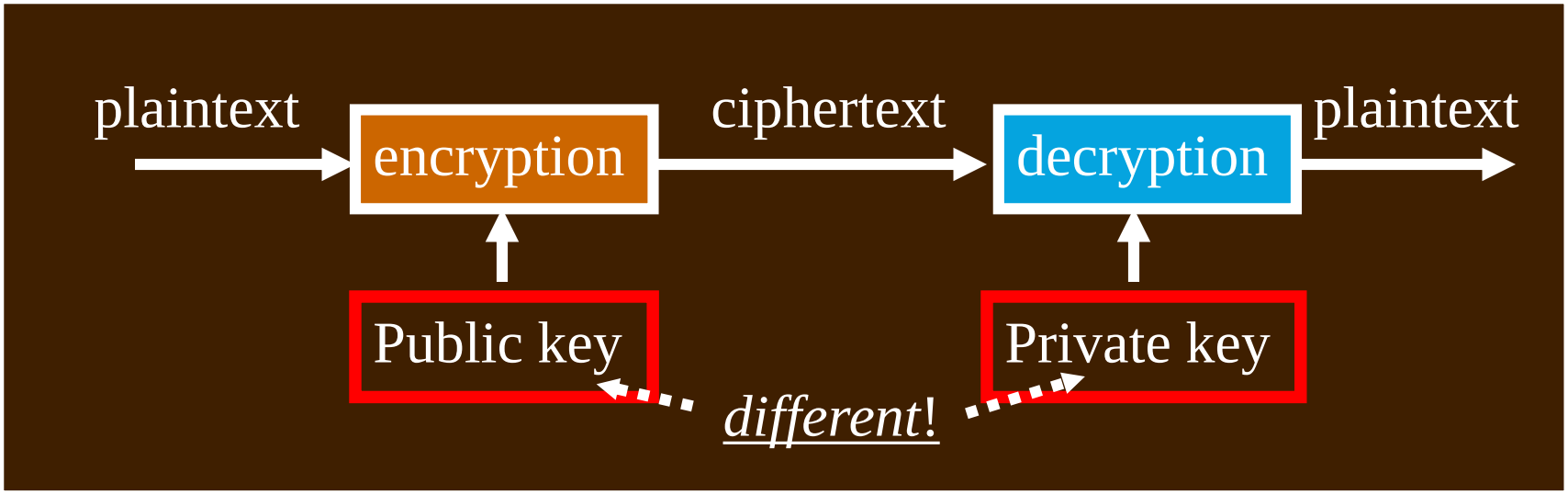
CIS 6930/4930 Computer and Network Security

Topic 5.2 Public Key Cryptography

Public Key Cryptography



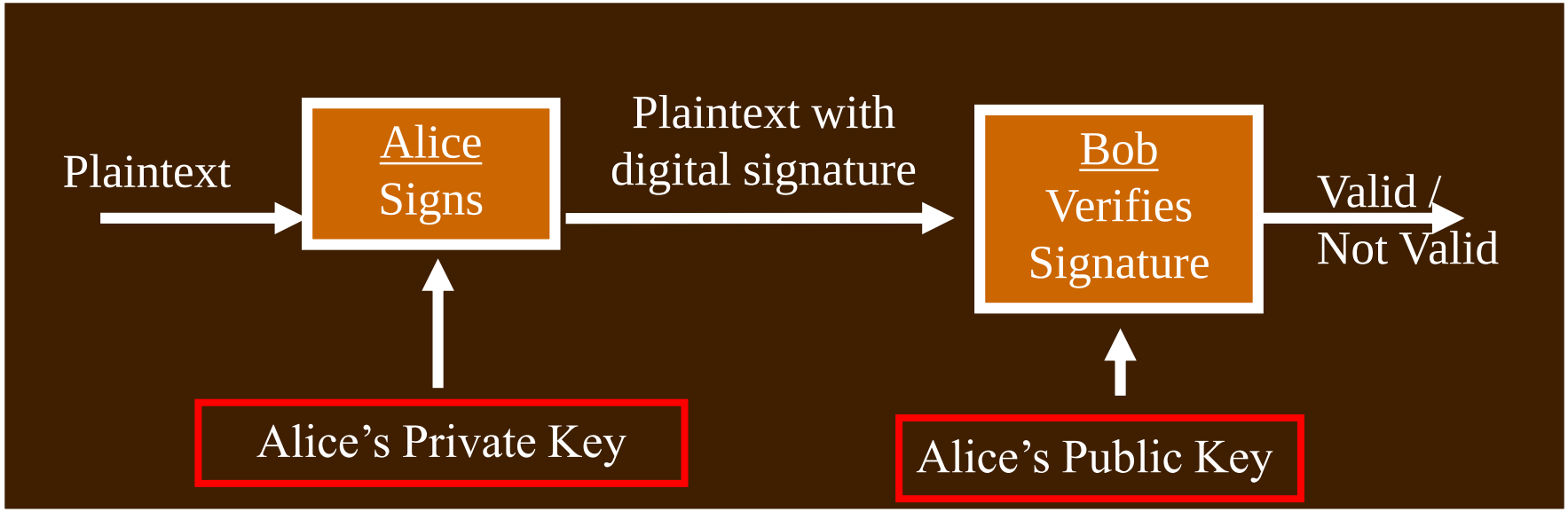
- Invented and published in 1975
- A *public / private key pair* is used
  - public key can be announced to everyone
  - private key is kept secret by the owner of the key
- Also known as *asymmetric* cryptography
- Much slower to compute than secret key cryptography

Applications of Public Key Crypto

1. Message integrity with *digital signatures*

   Alice computes hash, signs with her private key (no one else can do this without her key)

   Bob verifies hash on receipt using Alice's public key using the verification equation

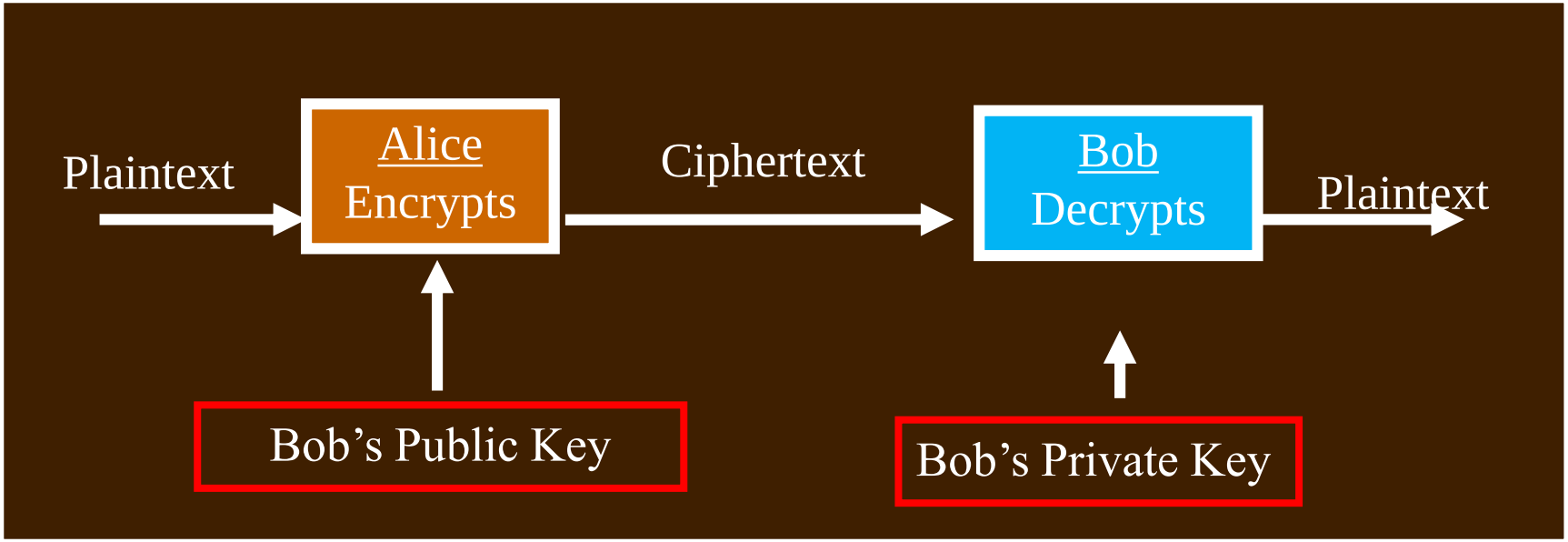


Applications (Cont'd)

- The digital signature is verifiable by anybody
- Only one person can sign the message: *non-repudiation*
  - Why is non-repudiation for a single party not possible with secret key cryptography?

Applications (Cont'd)

2. Communicating securely over an insecure channel
   - Alice encrypts plaintext using Bob's public key, and Bob decrypts ciphertext using his private key
   - No one else can decrypt the message (because they don't have Bob's private key)



Applications (Cont'd)

3. Secure storage
   - Users encrypt data using the storage provider's public key
4. *User Authentication*
   - Bob proves his identity to Alice by using his private key to perform an operation (without disclosing his private key)
   - Alice verifies result using Bob's public key

Public Key Algorithms

- Public key algorithms covered in this class, and their applications

System	Encryption / Decryption?	Digital Signatures?	Key Exchange?
RSA	Yes	Yes	Yes
Diffie-Hellman			Yes
DSA		Yes	

Public-Key Requirements

- It must be computationally
  - easy to generate a public / private key pair
  - hard to determine the private key, given the public key
- It must be computationally
  - easy to encrypt using the public key
  - easy to decrypt using the private key
  - hard to recover the plaintext message from just the ciphertext and the public key

Trapdoor One-Way Functions

- *Trapdoor* one-way function
  - $Y=f_k(X)$: easy to compute if k and X are known
  - $X=f^{-1}_k(Y)$: easy to compute if k and Y are known
  - $X=f^{-1}_k(Y)$: hard if Y is known but k is unknown
- Goal of designing public-key algorithm is to find appropriate trapdoor one-way function

The RSA Cipher

RSA (Rivest, Shamir, Adleman)

- The most popular public key method
  - provides both public key encryption and digital signatures
- Basis: factorization of large numbers is hard
- Variable key length (1024 bits or greater)
- Variable plaintext block size
  - plaintext block size must be smaller than key size
  - ciphertext block size is same as key size

Generating a Public/Private Key Pair

- Find large primes *p* and *q*
- Let $n = p*q$
  - do not disclose *p* and *q*!
  - $\phi(n)$ = *???*
- Choose an *e* that is relatively prime to $\phi(n)$
  - **public** key = <*e*,*n*>
- Find *d* = multiplicative inverse of *e* mod $\phi(n)$ (i.e., $e*d = 1 \bmod \phi(n)$)
  - **private** key = <*d*,*n*>

RSA Operations

- For plaintext message m and ciphertext c

Encryption: $c = m^e \bmod n$, $m < n$

Decryption: $m = c^d \bmod n$

Signing: $s = m^d \bmod n$, $m < n$

Verification: $m = s^e \bmod n$

Proof (D(E(m)) = m)

- Given
  - public key = <e, n> and private key = <d, n>
  - $n = p*q$, $\phi(n) = (p-1)(q-1)$
  - $e*d \equiv 1 \bmod \phi(n)$
- If encryption is $c = m^e \bmod n$, *d*ecryption…

  $= c^d \bmod n$

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

  $= m \bmod n$ (why?)

  $= m$ (why?)
- (digital signature proof is similar)

RSA Example: Encryption and Signing

- Choose p = 23, q = 11 (both primes)
  - $n = p*q = 253$
  - $\phi(n) = (p-1)(q-1) = 220$
- Choose e = **39** (relatively prime to 220)
  - public key = <**39**, 253>
- Find e^{-1} mod 220 = d = **79**
  (note: $39*79 \equiv 1 \bmod 220$)
  - private key = <**79**, 253>

Example (Cont'd)

- Suppose plaintext **m = 80**

Encryption

c = 80^{39} mod 253 = **37** ($c = m^e \bmod n$)

Decryption

m = 37^{79} mod 253 = **80** ($c^d \bmod n$)

Signing (in this case, for entire message **m**)

s = $\mathbf{80}^{79}$ mod 253 = 224 ($s = m^d \bmod n$)

Verification

m = 224^{39} mod 253 = **80** ($s^e \bmod n$)

Using RSA for Key Negotiation

- Procedure
  1. *A* sends random number *R*1 to *B*, encrypted with *B*'s public key
  2. *B* sends random number *R*2 to *A*, encrypted with *A*'s public key
  3. *A* and *B* both decrypt received messages using their respective private keys
  4. *A* and *B* both compute K = H(*R*1⊕*R*2), and use that as the shared key

Key Negotiation Example

- For Alice, e = 39, d = 79, n = 253
- For Bob, e = 23, d = 47, n = 589 (=19*31)
- Let **R1 = 15**, **R2 = 55**

1. Alice sends **306** = **15**$^{?}$ mod ? to Bob
2. Bob sends **187** = **55**$^{?}$ mod ? to Alice
3. Alice computes R2 = **55** = **187**$^{?}$ mod ?
4. Bob computes R1 = **15** = **306**$^{?}$ mod ?
5. A and B both compute K = H(R1⊕R2), and use that as the shared key

Is RSA Secure?

- <*e*,*n*> is public information
- If you could factor *n* into *p***q*, then
  - could compute $\phi(n) = (p-1)(q-1)$
  - could compute $d = e^{-1} \bmod \phi(n)$
  - would know the private key <*d*,*n*>!
- But: factoring large integers is hard!
  - classical problem worked on for centuries; no known reliable, fast method

Security (Cont'd)

- At present, key sizes of 1024 bits are considered to be secure, but 2048 bits is better
- Tips for making *n* difficult to factor
  1. *p* and *q* lengths should be similar (ex.: ~500 bits each if key is 1024 bits)
  2. both (*p*-1) and (*q*-1) should contain a "large" prime factor
  3. gcd(*p*-1, *q*-1) should be "small"
  4. *d* should be larger than $n^{1/4}$

Attacks Against RSA

- Brute force: try all possible private keys
  - can be defeated by using a large enough key space (e.g., 1024 bit keys or larger)
- Mathematical attacks
  - factor n into the product of two primes
    - possible for special cases of n

Attacks (Cont'd)

- Probable-message attack (using <*e*,*n*>)
  - encrypt all possible plaintext messages
  - try to find a match between the ciphertext and one of the encrypted messages
  - only works for small plaintext message sizes
- Solution: pad plaintext message with random text before encryption (why random?)
- PKCS #1 v1 specifies this padding format:

each 8 bits long

Timing Attacks Against RSA

- Recovers the private key from the running time of the decryption algorithm
- Computing $m = c^d \bmod n$ using repeated squaring algorithm:

```
d = 1;
for i = k downto 1 do
      d = (d * d) % n;              /* square */
      if (b_i == 1)
            d = (d * a) % n;        /* step 2 */
      endif
end do
return d;
```

Timing Attacks (Cont'd)

The attack proceeds bit by bit
Attacker assumed to know **c**, **m**
Attacker is able to determine bit *i* of *d* because, the highlighted step takes extra time if $d_i = 1$

Countermeasures to Timing Attacks

- Constant exponentiation time
  - Don’t return the result if the computation is too fast.
  - Hurt the performance.
- Random delay
  - Confuse the timing attack by adding a random delay
  - The attacker may be able to defeat random delay if the delay is not added carefully.