CYS401: Fundamentals of Cybersecurity

Lecture 6.1:

Encryption and Cryptography (2)

Asymmetric Cryptography/ Public Key Encryption

Agenda

- Symmetric Versus Asymmetric Cryptography
- Asymmetric Ciphers/ Public Key Encryption
- Asymmetric Cryptography Trapdoor One- Way Function
- RSA Cryptosystem: Encryption, Decryption, key Generation
- Extended Euclidean Algorithm
- RSA Encryption vs. RSA Signature
- RSA Security
- Hash Function vs Message Authentication Code (MAC)
- Hashing Algorithms: SHA & MD
- Digital Signatures

Objectives

- Distinguish between Symmetric and asymmetric key cryptography
- Understand Asymmetric key cryptography.
- Discuss the RSA cryptosystem
- Learn about the mathematical foundations of RSA, including prime factorization and modular arithmetic.
- Discuss the security of RSA
- Understand Data Integrity: Applications of Hash Functions

Symmetric Cryptography Versus Asymmetric Cryptography

Note-1

Symmetric-key cryptography is based on sharing secrecy; asymmetric-key cryptography is based on personal secrecy.

Note-2

In symmetric-key cryptography system, the number of keys needed for each user is 1.
In asymmetric-key cryptography system, the number of keys needed for each user is 2.

Symmetric Cryptography Versus Asymmetric Cryptography

Note-3

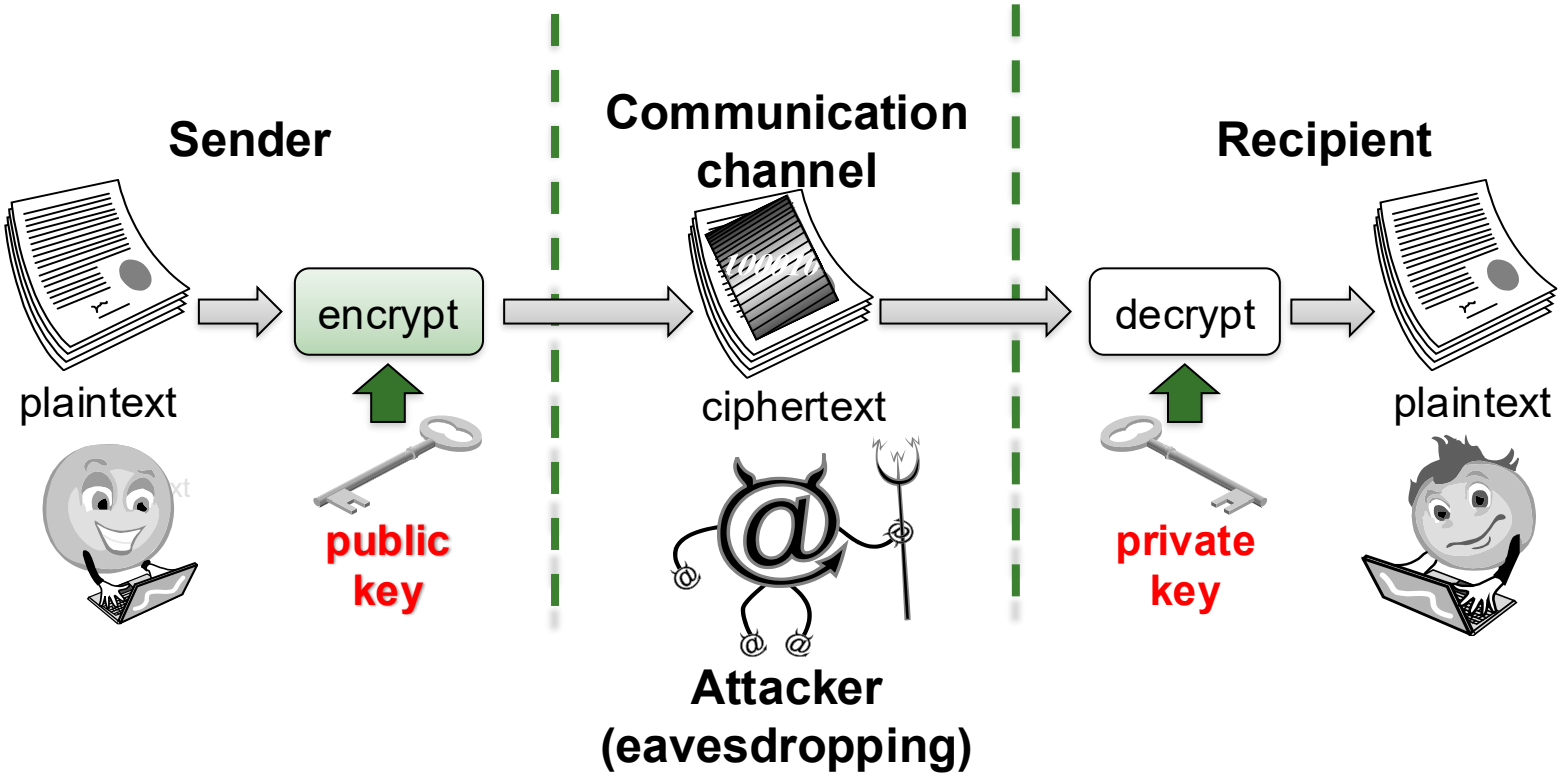
In symmetric-key cryptography, symbols in plaintext and ciphertext are permuted or substituted.
In asymmetric-key cryptography, plaintext and ciphertext are treated as integers.

Note-4

Symmetric-key cryptography is appropriate for long messages, and the speed of encryption/decryption is fast.
Asymmetric-key cryptography is appropriate for short messages, and the speed of encryption/decryption is slow.

Asymmetric Key Cryptography / Public Key Encryption

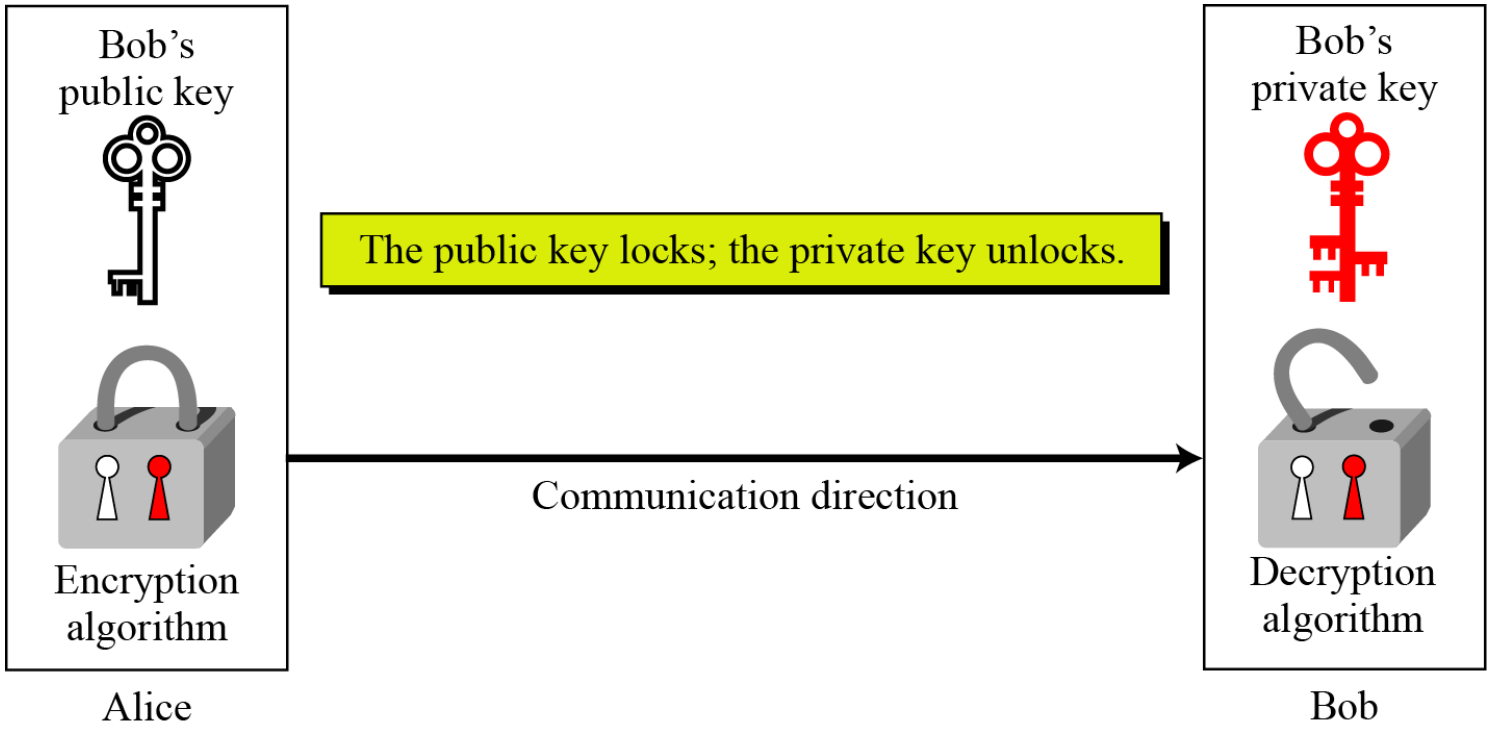
- Asymmetric-key cryptography can be used for confidentiality (privacy / secrecy), authentication, or both.
- Separate keys are used for encryption and decryption.



Keys

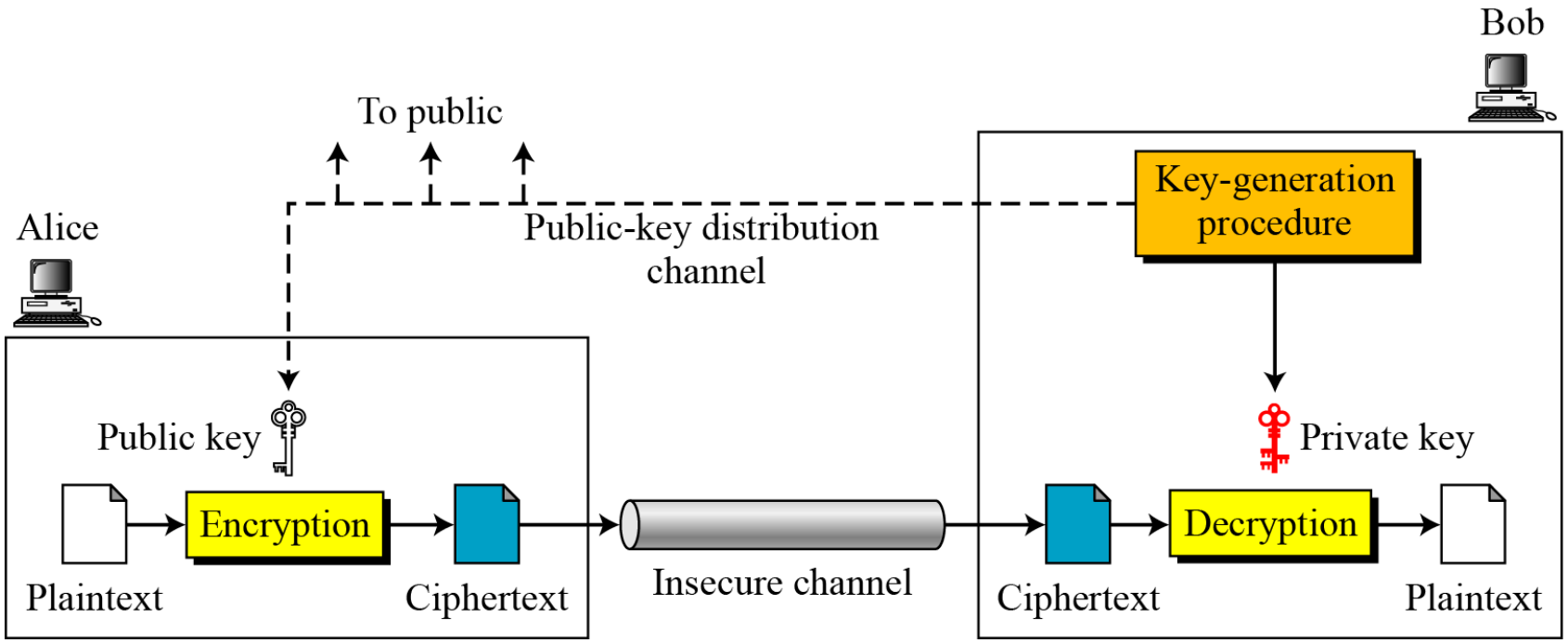
Asymmetric key cryptography uses two separate keys: one private and one public.

Figure: ***Locking and unlocking in asymmetric-key cryptosystem***



General Idea

General idea of asymmetric-key cryptosystem

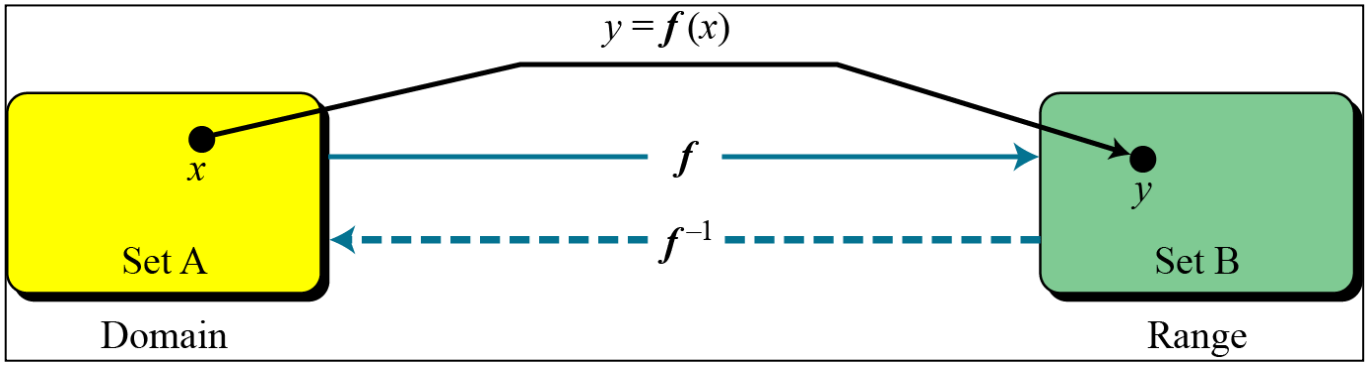


$$C = f(K_{public}, P) \qquad P = g(K_{private}, C)$$

Trapdoor One-Way Function

The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.

Functions ***A function as rule mapping a domain to a range***



One-Way Function (OWF)

1. *f is easy to compute → y=f(x)*
2. *f^{-1} is difficult to compute → $x=f^{-1}(y)$*

RSA CRYPTOSYSTEM

- The most common public-key algorithm is called the RSA method after its inventors (**R**ivest, **S**hamir and **A**dleman)
- Aims to compute three factors **N, D, and e** based on prime numbers.
- The **private key** here is a pair of numbers **(N,d)**
- The **public key** is also a pair of numbers **(N,e)**
- Note that N is common to the private and public keys

RSA *Encryption…*

- The sender uses the following algorithm to encrypt the message: (with Public key)

$$C = M^e \bmod N$$ where M is the plain text

  - M is the plain text
  - C is the ciphertext
  - (N, e) are the public key components
  - Plaintext M is raised to the power e and divided by mod N
  - The mode term indicates that the remainder is sent as the ciphertext

RSA *Decryption…*

- The receiver uses the following algorithm to decrypt the message: (with private key)

$$\mathbf{M = C^d \bmod N}$$

  - In this algorithm, M and C are plaintext and ciphertext
  - The two numbers d and N are components of the private key

RSA Continued

Encryption

Algorithm 10.3 *RSA encryption*

```
RSA_Encryption (P, e, n)                    // P is the plaintext in Z_n and P < n
{
    C  ←  Fast_Exponentiation (P, e, n)     // Calculation of (P^e mod n)
    return C
}
```

Decryption

Algorithm 10.4 *RSA decryption*

```
RSA_Decryption (C, d, n)                    //C is the ciphertext in Z_n
{
    P  ←  Fast_Exponentiation (C, d, n)     // Calculation of (C^d mod n)
    return P
}
```

RSA: Key Generation

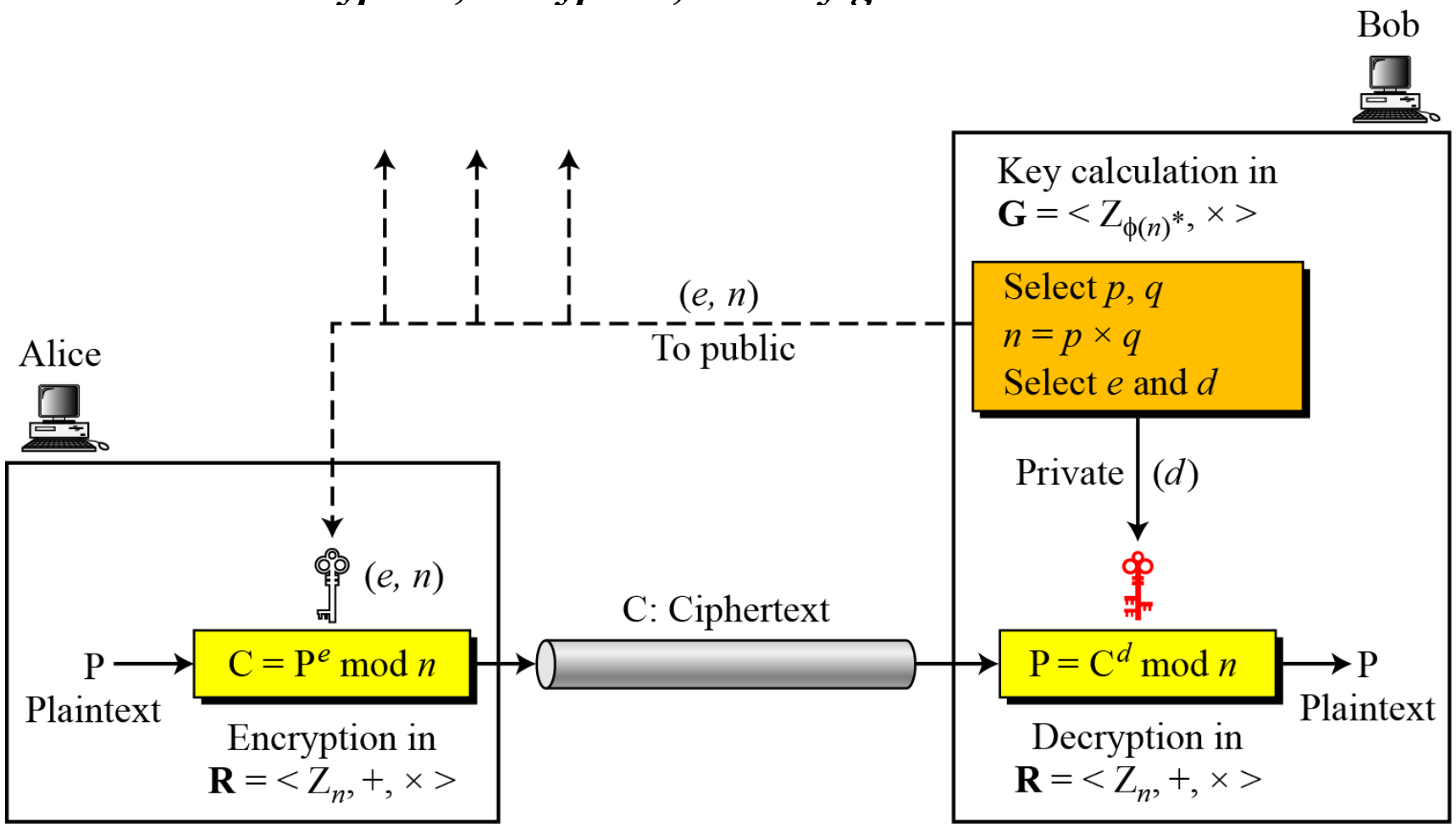
- Select two prime numbers p and q, where $p \neq q$
- Calculate $n = p \times q$
- Calculate $\phi(n) = (p-1)x(q-1)$
- Select prime integer e, where $1 < e < \phi(n)$
- Choose prime number d, such that $1 < d < \phi(n)$ and $d \times e \bmod \phi(n) = 1$ i.e $d = e^{-1} \bmod \phi(n)$. Calculated using extended Euclidean Algorithm

Thus

- Public Key PU = {e,n}
- Private Key PR = {d,n}

RSA Procedure

Encryption, decryption, and key generation in RSA

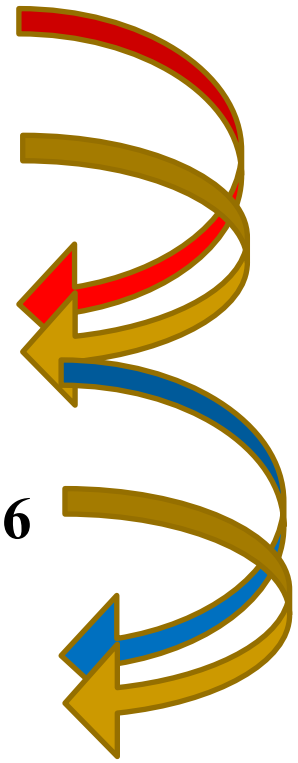


In RSA, p and q must be at least 512 bits; n must be at least 1024 bits.

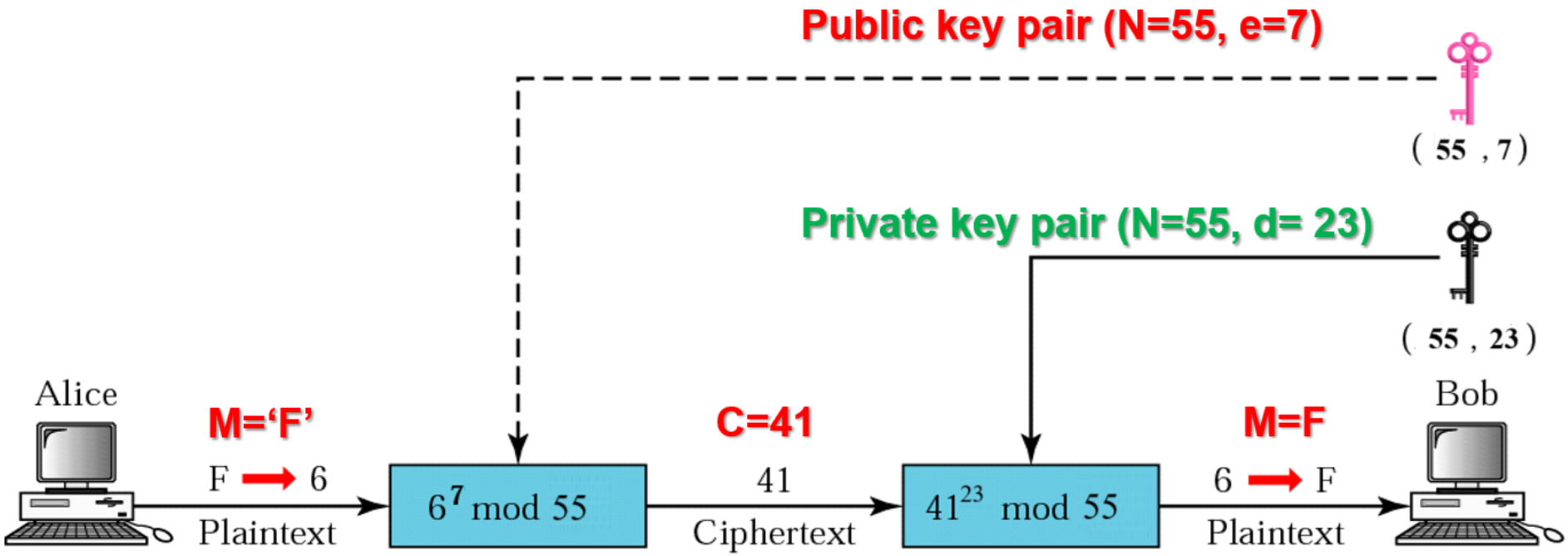
Basic Concepts: Extended Euclidean Algorithm

If p = 5, q=11, e=7, then n=55, **ø(n)** =40. Find d, whereby e*d mod **ø(n)** = 1

40	40	ø(n)
7	1	
7*5=35	1*5=5	
40-35=5	40-5=35	
7	1	
5	35	
5*1=5	35*1=35	
7-5=2	1-35=-34 = 6	
5	35	
2	6	
2*2=4	6*2=12	
5 - 4= 1	35 – 12 = 23	



RSA *Application*



RSA: Some Trivial Examples

RSA Example 1:

Bob chooses 7 and 11 as *p* and *q* and calculates *n* = 77. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, *e* and *d*, from $Z_{60}*$. If he chooses *e* to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

Plaintext: 5	$C = 5^{13} = 26 \bmod 77$	Ciphertext: 26

Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

Ciphertext: 26	$P = 26^{37} = 5 \bmod 77$	Plaintext: 5

RSA Example 2:

- Select primes: **p=3** & **q=11**
- Compute **n** = pq =3 × 11= 33
- Compute **ø(n)** = (p–1)(q-1)=2 × 10= 20
- Select **e** such that gcd(e,20) = 1; choose **e=7**
- Determine **d** such that de = 1 mod 20, **d= 3**
- Publish public key **PU={7, 33}**
- Keep secret private key **PR={3, 33}**

Given message **M = 2**

- Encryption: $C = 2^7 \bmod 33 = 29$
- Decryption: $M = 29^3 \bmod 33 = 2$

RSA Homework Practice:

Find the public key and private key of RSA for the following factors:

- $p = 7,\ q = 13$
- $n =$
- $\phi(n) =$
- $e =$
- $d =$

■Keys:

- public key: ( )
- private key: ( )

Encryption
vs.
Signature

Encryption

- Ensures the confidentiality
- Only users with key can decrypt

Signature

- Ensures the integrity
- The message should come from authenticated sender.
- The sender with private key sign the message.

RSA for Encryption

- Encrypt using public key
- Decrypt using private key

RSA for signature

- Sign using private key
- Verify using public key

	Sender has	Recipient has
Signing	Sender private key	Sender public key
Encrypting	Recipient public key	Recipient private key

RSA Signature

- Setup:
  - $n = pq$, with p and q primes
  - e relatively prime to $\phi(n) = (p - 1)(q - 1)$
  - d inverse of e in $Z_{\phi(n)}$
- Keys:
  - Public key: $K_E = (n, e)$
  - Private key: $K_D = (n, d)$
- Sign:
  - Plaintext M in Z_n
  - $S = M^d \bmod n$
  - $Sig = (S, M)$
- Verify:
  - $M = S^e \bmod n$

Example

- Setup:
  - $p = 7,\ q = 13$
  - $n = 7 \cdot 13 = 91$
  - $\phi(n) = 6 \cdot 12 = 72$
  - $e = 5$
  - $d = 29$
- Keys:
  - public key: (91, 5)
  - private key: 29
- Sign: using private
  - $M = 24$
  - $S = 24^{29} \bmod 91 = 33$
- Decryption: using public
  - $M = 33^5 \bmod 91 = 24$

RSA Security

- Security of RSA based on **difficulty of factoring**
  - Widely believed
  - Best known algorithm takes exponential time
- RSA Security factoring challenge (discontinued)
- In 1999, 512-bit challenge factored in 4 months using 35.7 CPU-years
  - 160 175-400 MHz SGI and Sun
  - 8 250 MHz SGI Origin
  - 120 300-450 MHz Pentium II
  - 4 500 MHz Digital/Compaq

- In 2005, a team of researchers factored the RSA-640 challenge number using 30 2.2GHz CPU

- In 2004, the prize for factoring RSA-2048 was $200,000
- Current practice is 2,048-bit keys
- Estimated resources needed to factor a number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^6	170GB
1,620	1.6×10^{15}	120TB

Data Integrity: Applications of Hash Functions

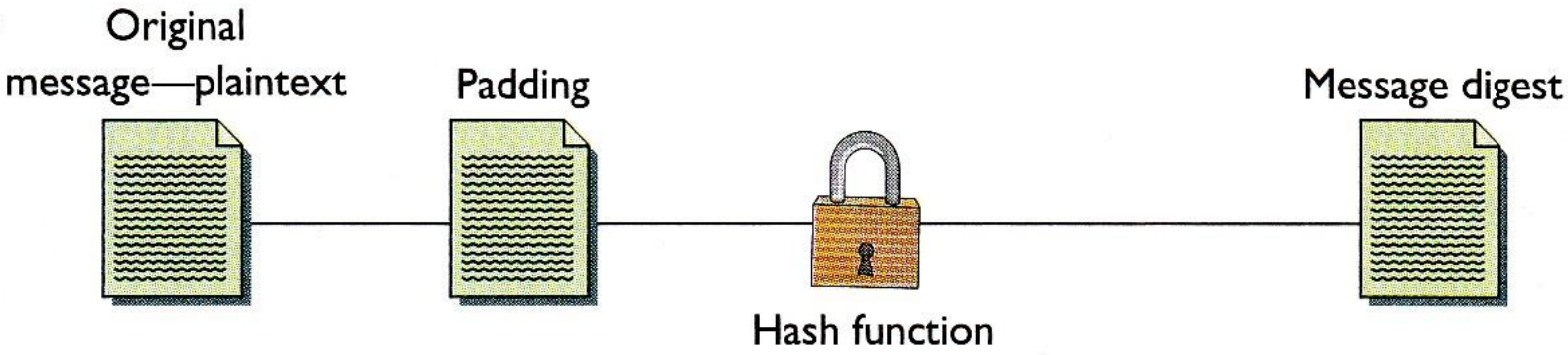
Hash Functions

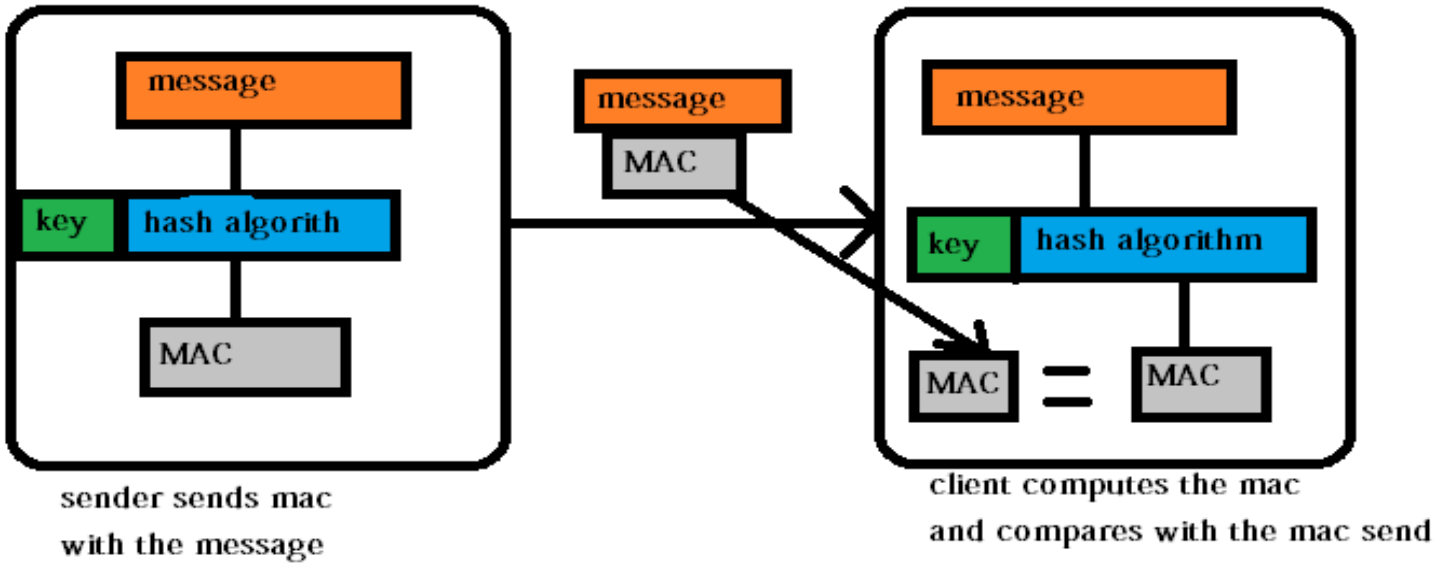
- Hashing functions commonly use encryption methods
- A hashing function or hash function is a special mathematical function that performs ***one-way encryption***, which means that once the algorithm is processed, there is no feasible way to use ciphertext to retrieve the plaintext that was used to generate it
- Also ideally, there is no feasible way to generate two different plaintexts that compute to the same hash value
- The hash value (Message Digest): is the output of the hashing algorithm for a specific input
  - The message digest has **a fixed length** regardless the data length

Hash Functions

- Common use of hashing algorithms
  - store computer passwords
  - to ensure message integrity
- Hash Function produces a unique message digest, but
  - reproducible by anyone else running the same algorithm against the same data.
    - We need a secure way!
- So you could hash a message to get a message authentication code (**MAC**), and the computational number of the message would show that no intermediary has modified the message
  - Hash function is public, and anyone can run it; so we need a secret key.

Hash Function vs Message Authentication Code (MAC





Hash Functions

- A hash algorithm can be compromised with what is called a collision attack, in which an attacker finds two different messages that hash to the same value.
- This type of attack is very difficult and requires generating a separate algorithm that attempts to find a text that will hash to the same value of a known hash
- The consequence of a hash function that suffers from collision is a loss of integrity

Secure Hash Algorithm (SHA)

- Developed by NSA and approved as a federal standard by NIST
- **SHA-1 (1993)**
  - 160-bits long message digest
  - Vulnerable to collision attacks
- **SHA-2 family (2002)**
  - 256 bits (SHA-256) or 512 bits (SHA-512)
  - More secure from collision attacks than SHA-1
- **SHA-3 family (2015)**
  - released by NIST on August 2015.
  - the standard accepts 224, 256, 384 or 512 bits

Message Digest (MD)

- **MD2 (1989)**
  - Produce 128 bit hash
  - Vulnerable without the checksum appended to the message
- **MD4 (1990)**
  - It is a fast algorithm, but it is subject to more attacks than more secure algorithms such as MD5
- **MD5 (1991)**
  - Structured after MD4 but with additional security to overcome the problems in MD4
  - Therefore it is very similar to the MD4 algorithm, only slightly slower and more secure

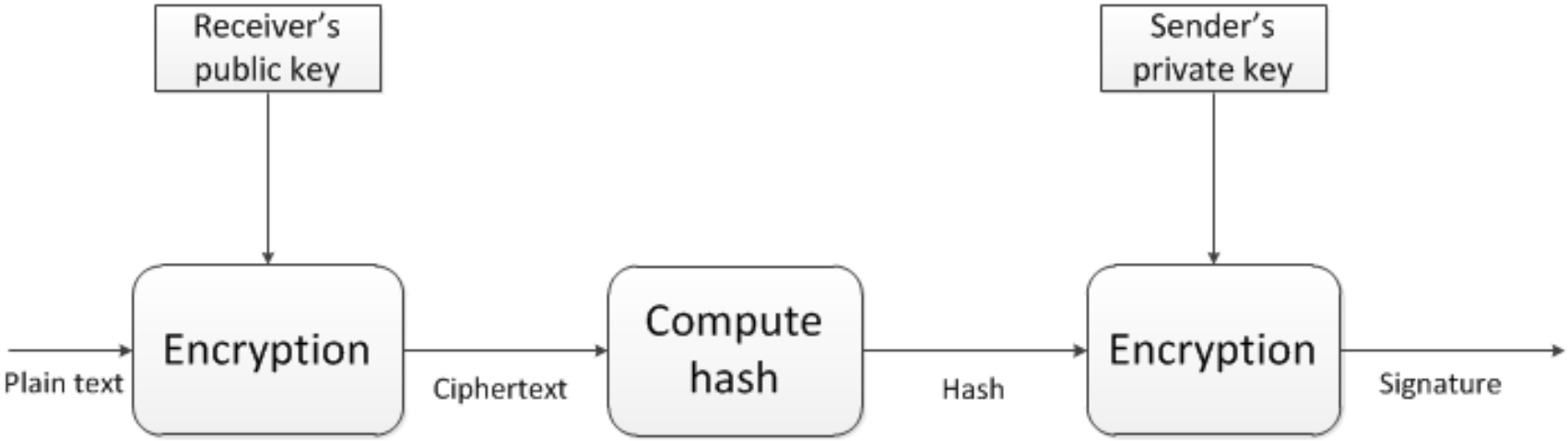
Examples

https://www.fileformat.info/tool/hash.htm

Hash vs. MAC vs. Signature

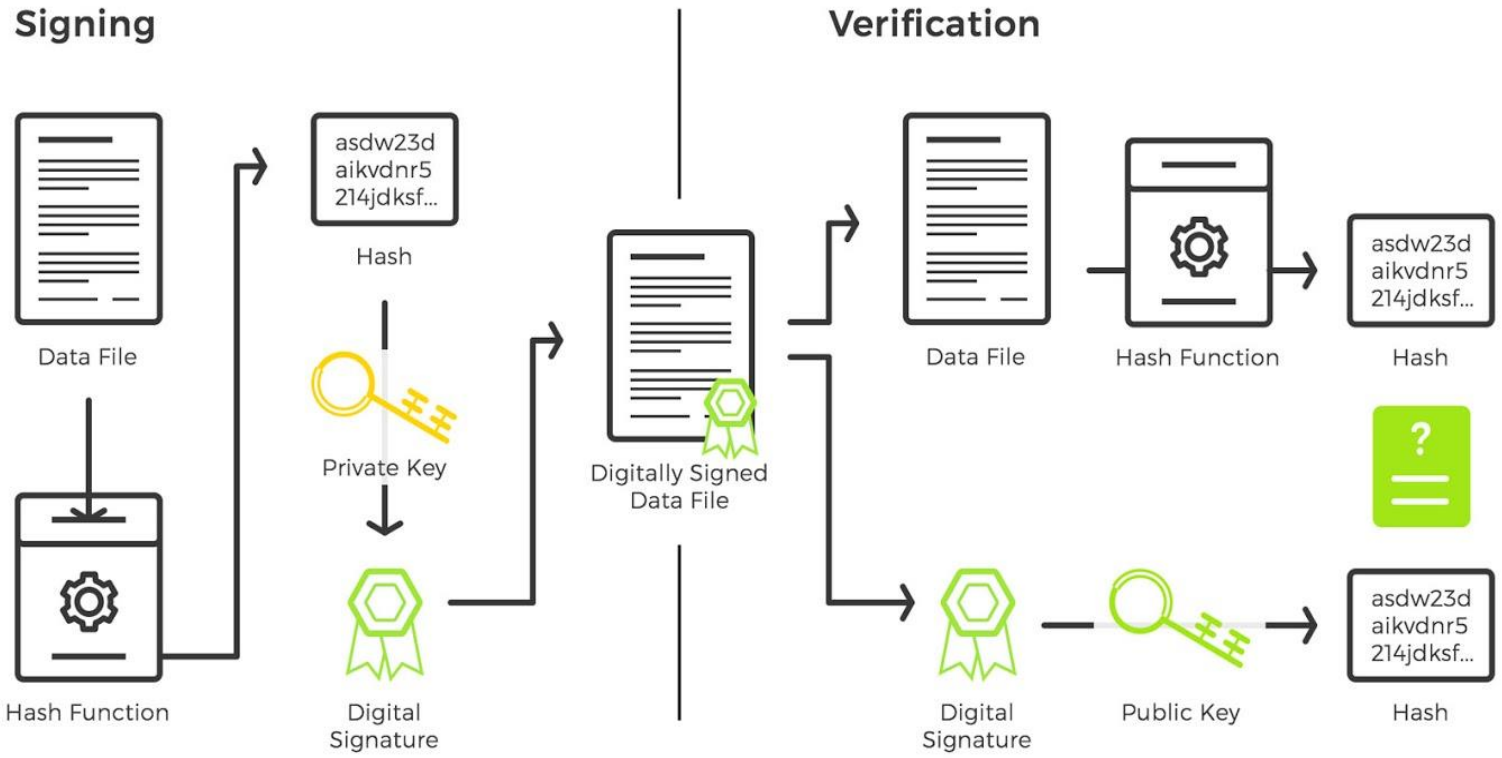
Cryptographic primitive / Security Goal	Hash	MAC	Digital signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Kind of keys	none	symmetric keys	asymmetric keys

Hash vs. MAC vs. Signature



Common Public Key

Digital Signature



Thanks