Chapter 7: Cryptography (Ciphers)

Lecture 30

Types of ciphers

Stream ciphers	encrypt messages a symbol at a time, by substitution
Block ciphers	substitute whole messages as single symbols
Product ciphers	repeatedly combine block ciphers with transpositions

Often-used block codes include

DES	(1970s, 64 bits)	Data Encryption Standard
IDEA	(1991, 128 bits)	International Data Encryption Algorithm
AES	(2001, 128/192/256 bits)	Advanced Encryption Standard

A one-way function f satisfies these conditions:

- given x, it is easy to find $f(x)$
- given $f(x)$, it is hard to find x

Example

Given a large prime p and primitive element g in $\mathbb{Z}_p$, let

$$f(x) \equiv g^x \pmod{p}$$

It is easy ($\mathrm{O}(\log p)$) to calculate g^x
— but it is hard to find x given g^x.
(This is the Discrete Logarithm Problem.)

A one-way function f satisfies these conditions:

- given x, it is easy to find $f(x)$
- given $f(x)$, it is hard to find x

A one-way hash function f satisfies these conditions:

- f is a one-way function
- f takes a message of any length and maps it to a fixed length value

$$\left.\begin{array}{l}\bullet\ \text{given } x\text{, it is hard to find an } x' \text{ with } f(x) = f(x') \\ \bullet\ \text{it is hard to find any pair } x, x' \text{ with } f(x) = f(x')\end{array}\right\}\ \text{collision resistance}$$

One-way hash functions are used for UNIX passwords and ATM machines.

A (one-way) trapdoor function satisfies these conditions:

- f is a one-way function
- given $f(x)$ and some small extra information, it is easy to find x

A one-way function f satisfies these conditions:

- given x, it is easy to find $f(x)$
- given $f(x)$, it is hard to find x

A (one-way) trapdoor function satisfies these conditions:

- f is a one-way function
- given $f(x)$ and some small extra information, it is easy to find x

Example

Let p, q be two large primes with $p, q \equiv 3 \pmod 4$ and define $n = pq$.

- (Such primes are called Blum primes and n is a Blum integer.)

Then $f(x) \equiv x^2 \pmod n$ is a trapdoor function:

- finding square roots modulo n is hard
- finding square roots modulo p and q is relatively easy
- now find a square root modulo n with the Chinese Remainder Theorem

A one-way function f satisfies these conditions:

- given x, it is easy to find $f(x)$
- given $f(x)$, it is hard to find x

A (one-way) trapdoor function satisfies these conditions:

- f is a one-way function
- given $f(x)$ and some small extra information, it is easy to find x

Example

Let p, q be two large primes with $p, q \equiv 3 \pmod 4$ and define $n = pq$.

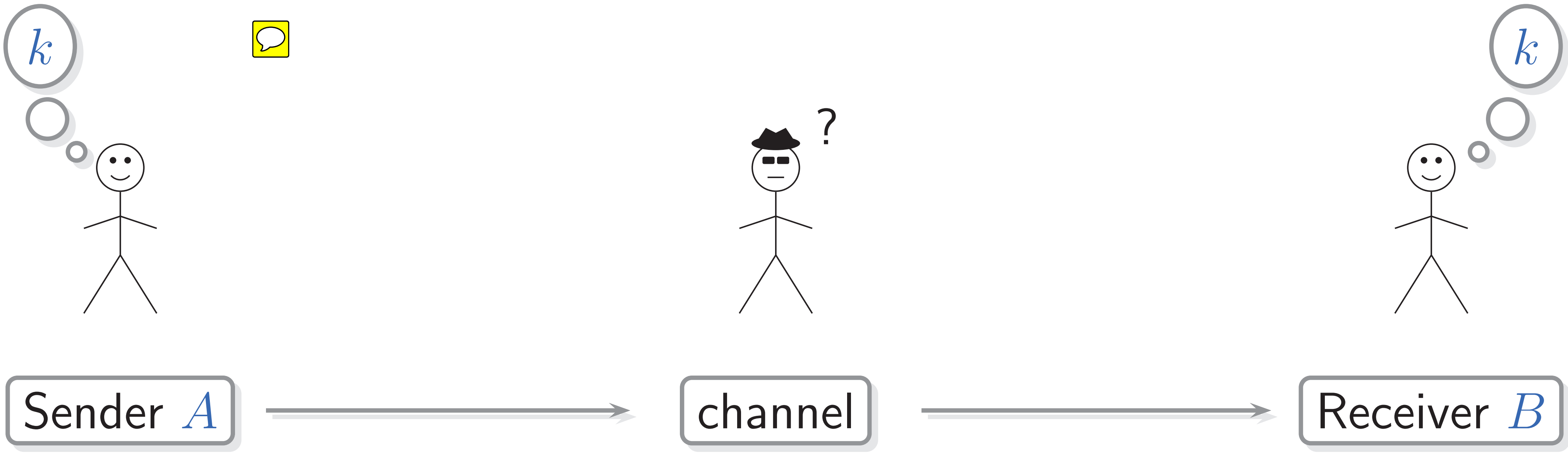
Then $f(x) \equiv x^2 \pmod n$ is a trapdoor function:

- finding square roots modulo n is hard
- finding square roots modulo p and q is relatively easy
- now find a square root modulo n with the Chinese Remainder Theorem

Here, (p, q) is the trapdoor.

In symmetric cryptosystems, the sender & receiver use the same secret key.

- All classical cryptosystems that we have seen so far are symmetric.
- These require safe, public ways for sender & receiver to agree on a key.



Diffie-Hellman key exchange protocol (1976)

① A sends to B a large prime p and a primitive element g of $\mathbb{Z}_p$

② A chooses a secret random number a
B chooses a secret random number b

③ A computes and sends to B the number $x \equiv g^a \pmod{p}$
B computes and sends to A the number $y \equiv g^b \pmod{p}$

④ A calculates the key $k \equiv y^a \equiv g^{ab} \pmod{p}$
B calculates the key $k \equiv x^b \equiv g^{ab} \pmod{p}$

⑤ A and B now communicate in secret with key k, say via AES 256.

Diffie-Hellman key exchange protocol (1976)

① A sends to B a large prime p and a primitive element g of $\mathbb{Z}_p$

② A chooses a secret random number a
B chooses a secret random number b

③ A computes and sends to B the number $x \equiv g^a \pmod p$
B computes and sends to A the number $y \equiv g^b \pmod p$

④ A calculates the key $k \equiv y^a \equiv g^{ab} \pmod p$
B calculates the key $k \equiv x^b \equiv g^{ab} \pmod p$

⑤ A and B now communicate in secret with key k, say via AES 256.

To find k, the listener must know either (x, b) or (y, a).
But finding a or b is hard: it is a discrete logarithm problem.

This protocol uses the one-way function $f(a) \equiv g^a \pmod p$.

Diffie-Hellman key exchange protocol (1976)

① A sends to B a large prime p and a primitive element g of $\mathbb{Z}_p$

② A chooses a secret random number a
B chooses a secret random number b

③ A computes and sends to B the number $x \equiv g^a \pmod{p}$
B computes and sends to A the number $y \equiv g^b \pmod{p}$

④ A calculates the key $k \equiv y^a \equiv g^{ab} \pmod{p}$
B calculates the key $k \equiv x^b \equiv g^{ab} \pmod{p}$

⑤ A and B now communicate in secret with key k, say via AES 256.

Example

① A chooses and sends to B the prime $p = 101$ and primitive $g = 12$

② A and B choose $a = 8$ and $b = 19$

③ A and B send $x = 12^8 = 52$ and $y = 12^{19} = 50$ in $\mathbb{Z}_{101}$

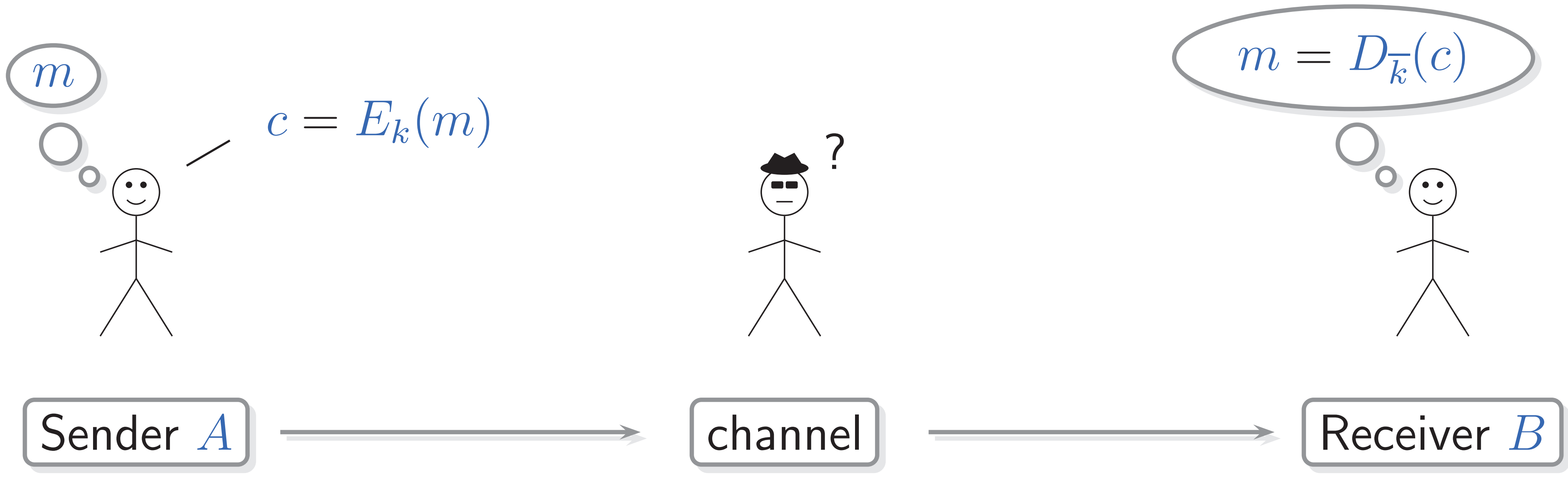
④ A and B calculate $k = 50^8 = 58$ and $k = 52^{19} = 58$ in $\mathbb{Z}_{101}$

⑤ A and B now communicate in secret with key $k = 58 \pmod{101}$

There are other key-exchange protocols, like the Massey-Omura Protocol, which is also based on the discrete logarithm problem.

Public key cryptography (Diffie & Hellman 1976)

- Each receiver generates a public key k and a private key $\overline{k}$
- The public key k is used to encrypt (E_k)
  The private key $\overline{k}$ is used to decrypt ($D_{\overline{k}}$)
- asymmetric cryptography: there is no longer a common key
  In fact, there is no need to exchange keys at all!
- The keys should satisfy the following conditions:
  - $D_{\overline{k}}(E_k(m)) = m$ for all messages m
  - $E_k(m) = c$ is easy to calculate
    $D_{\overline{k}}(c) = m$ is easy to calculate
  - $E_k(m) = c$ is hard to invert if $\overline{k}$ is not known
- Also, it should be easy to generate a random key-pair $(k, \overline{k})$
  – and it must be hard to determine $\overline{k}$ from k



PUBLIC KEY CRYPTOGRAPHY (Diffie & Hellman 1976)

- Each receiver generates a public key k and a private key $\overline{k}$
- The public key k is used to encrypt (E_k)

  The private key $\overline{k}$ is used to decrypt $(D_{\overline{k}})$

Public key cryptography

RSA (Rivest, Shamir, Adleman 1977)

El Gamal Public-Key Cryptosystem (1985)

McEliece Encryption (1978)

others...

RSA (Rivest, Shamir, Adleman 1977)

Each user does as follows:

1. Choose large random primes p, q
2. Calculate $n = pq$ and $\phi(n) = (p-1)(q-1)$
3. Choose random $e \in \{1, \dots, \phi(n)\}$ with $\gcd(e, \phi(n)) = 1$
4. Calculate $d \equiv e^{-1} \pmod{\phi(n)}$

Public key: $\underline{k} = (n, e)$
Private key: $\overline{k} = (n, d)$

Encryption: $E_{\underline{k}}(m) \equiv m^e \pmod{n}$
Decryption: $D_{\overline{k}}(c) \equiv c^d \pmod{n}$

Lemma $D_{\overline{k}}(E_{\underline{k}}(m)) = m$

Proof Now, $ed \equiv 1 \pmod{\phi(n)}$, so $ed = s\phi(n) + 1$ for some $s \in \mathbb{Z}$.

By Euler's Theorem, $m^{\phi(n)} \equiv 1 \pmod{n}$, so

$$D_{\overline{k}}(E_{\underline{k}}(m)) \equiv (m^e)^d \equiv m^{ed} \equiv m^{s\phi(n)+1} \equiv (m^{\phi(n)})^s m \equiv m \pmod{n} \quad \square$$

RSA (Rivest, Shamir, Adleman 1977)

Each user does as follows:

1. Choose large random primes p, q
2. Calculate $n = pq$ and $\phi(n) = (p-1)(q-1)$
3. Choose random $e \in \{1, \ldots, \phi(n)\}$ with $\gcd(e, \phi(n)) = 1$
4. Calculate $d \equiv e^{-1} \pmod{\phi(n)}$

Public key: $\underline{k} = (n, e)$
Private key: $\overline{k} = (n, d)$

Encryption: $E_{\underline{k}}(m) \equiv m^e \pmod{n}$
Decryption: $D_{\overline{k}}(c) \equiv c^d \pmod{n}$

Lemma $D_{\overline{k}}(E_{\underline{k}}(m)) = m$

RSA relies also on the discrete logarithm problem.

Example

First replace characters by numbers using standard encoding

0	1	_	A	B	⋯	Y	Z
0	1	2	3	4	⋯	27	28

Suppose that $p = 31$ and $q = 47$ (both secret).
Then $n = 1457$ (public) and $\phi(n) = 1380$ (secret).

Suppose that $e = 7$ (public). Then $d \equiv e^{-1} \equiv 1183 \pmod{1380}$.

Now that d has been found, we can forget about p, q and $\phi(n)$.

For someone to send us the message Y, they would encode and encipher:

$$Y \to 27 = m \to m^e \equiv 27^7 \equiv 914 \equiv c \pmod{1457}$$

and send us the code $c = 914$.
To receive the message, we would decipher:

$$c = 914 \to c^d \equiv 914^{1183} \equiv 27 \equiv m \pmod{1457} \to Y$$

RSA (Rivest, Shamir, Adleman 1977)

These days, it is recommended that n is at least 2048 bits long.

Messages first get split up into blocks of $\frac{1}{8}n \geq \frac{2048}{8} \geq 256$ bytes, or $\frac{1}{16}n \geq \frac{2048}{16} \geq 128$ Unicode characters, say.

Each block gets treated as a number, and gets substituted by RSA.

These numbers are so big that Kasiski's method, say, does not work.

In practice, it is sometimes an art to find good pairs p, q and to use clever techniques for fast calculations for RSA.

Different encryption techniques are often combined.
Eg., the PGP (Pretty Good Privacy) package sends messages m as follows:

① A randomly generates a 128-bit key k
② A encrypts the key k as $\mathrm{RSA}_k(k)$ and the message m as $\mathrm{IDEA}_k(m)$
③ A sends $(\mathrm{RSA}_k(k), \mathrm{IDEA}_k(m))$ to B
④ B finds $(\mathrm{RSA}_{\overline{k}}(k), \mathrm{IDEA}_k(m)) = k$
⑤ B finds $\mathrm{IDEA}_k^{-1}(\mathrm{IDEA}_k(m)) = m$

McEliece Encryption (1978)

B generates keys:

① B chooses a (large) (n, k)-Goppa code that can correct t errors
② B chooses a $k \times n$ generator matrix G for this code
③ B chooses a random $k \times k$ invertible binary matrix S
④ B chooses a random $n \times n$ permutation matrix P
⑤ B calculates $\widehat{G} = SGP$: A's public key is $(\widehat{G}, t)$

A sends the (row-vector) message $\mathbf{m}$ to B:

① A calculates $\widehat{\mathbf{m}} = \mathbf{m}\widehat{G}$
② A chooses t random errors, given by a vector $\mathbf{e}_t$
③ A sends $\mathbf{c} = \widehat{\mathbf{m}} + \mathbf{e}_t$ to A

B decodes $\mathbf{c}$:

① B calculates $\mathbf{c}P^{-1} = \mathbf{m}SG + \widehat{\mathbf{e}_t}$ where $\widehat{\mathbf{e}_t} = \mathbf{e}_t P^{-1}$ still has t errors
② B error-corrects and gets $\mathbf{m}S$
③ B multiplies by S^{-1} to receive the message $\mathbf{m}$.

McEliece Encryption (1978)

McEliece originally suggested using $n = 1024$, $t = 50$, and $k = 524$.

The system with these parametres was cracked in 2008 by Tanja Lange and students (Eindhoven University of Technology).