

# Introduction to Modern Cryptography

Benny Chor

Symmetric Encryption:  
Stream & Block Ciphers

Lecture 2

Tel-Aviv University

revised Jan. 29th, 2008

# Pseudo Random Generators

A **pseudo random generator** is a polynomial time computable function  $G : \{0, 1\}^n \mapsto \{0, 1\}^m$  (on input of length  $n$  it produces an output of length  $m$ ), where  $m = n^c$ ,  $c > 1$ , which satisfies: The output of  $G$  is **polynomial time indistinguishable** from **truly random strings** of length  $m$ .

Further explanation on the board.

Notice that the output of such  $G$  cannot be **truly random**!

# One Way Functions

A **one way function** is a polynomial time computable function  $f : \{0, 1\}^n \mapsto \{0, 1\}^n$  (on input of length  $n$  it produces an output of length  $n$ ), which satisfies: The output of  $f$  cannot be **inverted** in polynomial time.

Further explanation on the board.

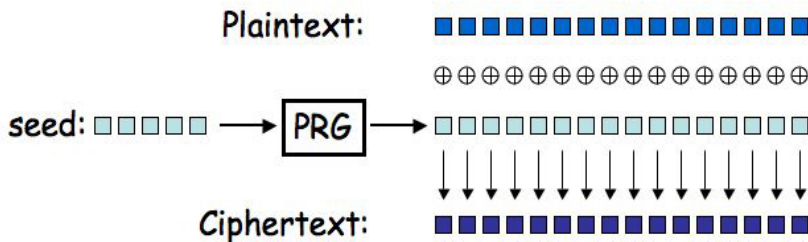
Remarks: It is often required that, in addition,  $f$  is a **permutation** of  $\{0, 1\}^n$ .

# PRGs and OW Functions

- Notice that if  $P=NP$  then neither PRGs nor OW functions exist (why?).
- So the existence of both can only be conjectured until P vs. NP is resolved.
- However there are good reasons to believe that both PRGs and OW functions do exist.
- Furthermore, it was shown that PRGs exist iff OW functions exist.
- The proof is too involved for our introductory course.
- But we will point out the relation of PRGs to stream ciphers, and of OW permutations to block ciphers.

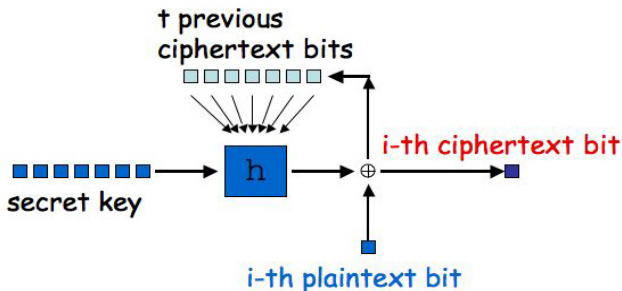
## Synchronous Stream Ciphers (“imitating” one-time pad)

- Start with a secret, random key (“seed”). Generate (online) a **keying stream** by applying the **PRG**,  $G$ , to the seed. The  $i$ -th bit of the keying stream is the  $i$ -th bit of  $G$ 's output.
- Combine the keying stream by bitwise XORing with the plaintext, to produce the ciphertext.
- This type of stream cipher is called **synchronous** (why?).
- Decryption is done in the same manner (XORing **ciphertext** with keying stream).



# Asynchronous (Self Synchronizing) Stream Ciphers

- Start with a secret, random key (“seed”),  $k$ .
- Generate (online) a **keying stream**. The  $i$ -th bit of the keying stream is a function of the seed and (possibly) the most recent  $t$  (constant) bits of the **ciphertext**,  
 $c_{i-t}, \dots, c_{i-1}$ .
- Specifically, encryption is  $c_i = m_i \oplus h(k, c_{i-t}, \dots, c_{i-1})$
- While decryption is  $m_i = c_i \oplus h(k, c_{i-t}, \dots, c_{i-1})$



# Synchronous vs. Asynchronous

- In **synchronous stream ciphers** the receiver must know the location in the ciphertext. Otherwise decryption is not possible, and states should be resynchronized.
- Insertion or deletion of any (non-zero) number of bits is fatal.
- Encrypting of **different messages** should be done using different “locations” on the keyed stream.
- Synchronous stream ciphers tend to be **faster**
- In **asynchronous stream ciphers**, it is possible to recover from lost or inserted bits after  $t$  correct ciphertext bits are received.
- So cipher does not require shared state, and is in fact **self synchronizing**.

# Real Synchronous Stream Ciphers

- Provide **concrete** implementations, each with fixed length key and fixed (maximum) key length.
- Formally there is nothing asymptotic, hence **cannot** be PRGs.
- Still, with a large key length  $\ell$  one hopes that the best way to break the code is by exhaustive search,  $2^\ell$ , or close to it.
- Concrete implementations usually have no theoretical foundations.
- Passing public scrutiny is a good measure, though certainly not a sufficient one.



# Real Synchronous Stream Ciphers

- Most pre-WWII machines



- German Enigma
- Linear Feedback Shift Register (LFSRs)
- A5 – encrypting GSM handset to base station communication
- RC4 – Ron's (Rivest) Code



# Linear Feedback Shift Registers and Stream Ciphers

- LFSRs have been investigated extensively.
- They have **extremely fast** implementations as hardware or software devices.
- With correct choice of wiring and initialization, output stream has a very long period.
- However, they are **way too weak** for cryptographic use – a relatively short output stretch allows to determine initial seed efficiently.
- Multiplexing or combining several LFSRs, and adding non-linear components, do produce good stream ciphers.

## Current Example: RC4

- Part of the RC family.
- Claimed by RSA as their IP.
- Between 1987 and 1994 its internal was not revealed – little analytic scrutiny.
- Had preferred export status.
- Code was released (anonymously) on the Internet.
- Used in many systems: Lotus Notes, SSL, etc.

## RC4: Initialization

1.  $j = 0$
2.  $S_0 = 0, S_1 = 1, \dots, S_{255} = 255$
3. Let the key be  $k_0, \dots, k_{255}$  (repeating bits if key has fewer bits)
4. For  $i = 0$  to 255
  - ▶  $j = (j + S_i + k_i) \bmod 256$
  - ▶ Swap  $S_i$  and  $S_j$

## RC4: Keying Stream Creation

An output byte  $B$  is generated as follows:

- $i = i + 1 \bmod 256$
- $j = j + S_i \bmod 256$
- **Swap**  $S_i$  and  $S_j$
- $r = S_i + S_j \bmod 256$
- $B = S_r$

$B$  is the next keying stream byte, and is XORed with the next plaintext byte to produce ciphertext byte.

## RC4 Properties

- Synchronous stream cipher, with byte oriented operations.
- Based on using a **randomly looking** permutation of the internal  $S_i$ .
- 8–16 machine operations per output byte.
- Very long cipher period (over  $10^{100}$ ).
- Widely believed to be secure (other than few initial bytes).
- Used for encryption in SSL web protocol.

# Keyed Functions and Permutations

- For each  $n > 0$ , let  $F : \{0, 1\}^n \times \{0, 1\}^n \Rightarrow \{0, 1\}^n$  be an efficiently computable function.
- For each length  $n$ ,  $F$  has two arguments. The first one is called the key.
- We will use the notation  $F_k(x) = y$ , where  $k$  is the key.
- For cryptographic uses, we often want that for every key  $k$ ,  $F_k$  is a permutation of  $\{0, 1\}^n$ .
- This way, the receiver of  $F_k(x) = y$ , who holds  $k$ , can (potentially) recover  $x$ .
- To enable this, we also require that for every key  $k$ ,  $F_k^{-1}$  is efficiently computable.
- But do we need anything else?



# Keyed Pseudo Random Functions and Permutations

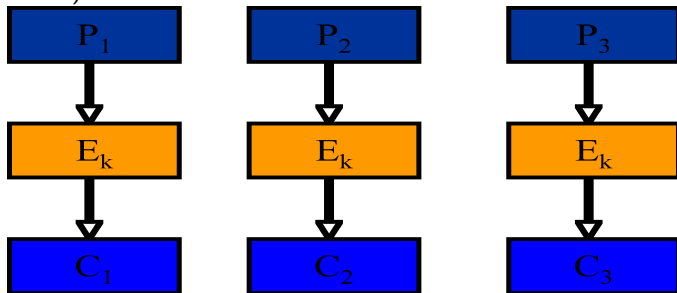
- Suppose  $F_k$  was a **truly random** function.
- Then given any set of pairs  $x_i, y_i$  with  $F_k(x_i) = y_i$ , this gives absolutely **no information** about the values  $F_k(x)$  or  $F_k^{-1}(y)$  for **new**  $x$  or  $y$ .
- However  $F_k$  cannot be **truly random** (why?).
- So will settle for  $F_k$  to be **pseudo random**.
- Should define what a pseudo random **function/permutation** means.
- Start with **truly random** function/permutation.
- Then require that it is **infeasible** to distinguish between the truly and pseudo random.
- Resort to whiteboard.

# Block Ciphers

- Encrypt a block of input to a block of output.
- Almost always the two blocks are of the same length.
- A block cipher is a concrete implementation of keyed pseudo random permutations, with concrete block sizes. Typically  $n = 64$  (DES) or  $n = 128$  (AES).
- Actual lengths of key and blocks may sometimes differ (slightly).

## Block Ciphers: Modes of Operation

- Different modes exist for encrypting plaintext longer than **one block**.
- Simplest mode is to encrypt each plaintext block **separately**.
- This is known as ECB mode encryption (Electronic Code Book).



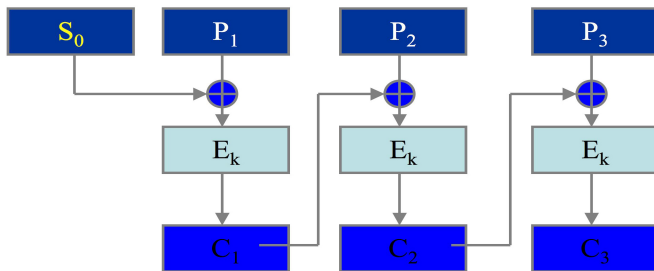
- Is there anything **wrong** with this mode?

# Block Ciphers: Modes of Operation

- If two plaintext blocks are **equal**, the corresponding ciphertext blocks will also be equal.
- This may be undesirable in some circumstances.
- Two plausible approaches to prevent this phenomena are
  - (a) Randomization (e.g. random padding of plaintext blocks).
  - (b) Introducing **state**.
- Most existing modes employ **state**.

## Block Ciphers: CBC Mode Encryption

In CBC mode (Cipher Block Chaining), previous ciphertext is XORed with current plaintext before encrypting current block.



An **initialization vector**,  $S_0$ , is used as a seed for the process. This seed can be openly transmitted.

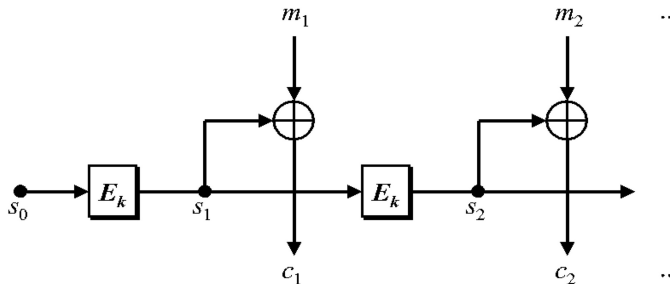
- **State** is last ciphertext.
- CBC is **self synchronizing**.

# Properties of CBC

- Asynchronous block/stream cipher.
- Errors in one ciphertext block **propagate**.
- Conceals plaintext patterns.
- Seems inherently sequential – no parallel implementation known.
- Plaintext cannot be easily manipulated.
- **Standard** in most systems: SSL, IPsec, etc.
- It is **proved** that if  $E$  is a pseudo random permutation, then CBC is **resistant** to **chosen plaintext attacks**.

## Additional Mode: OFB

In OFB (Output FeedBack) mode, an initialization vector  $S_0$  is use as a “seed” for a sequence of psuedo random blocks  $S_1, S_2, \dots$ . Each  $S_i$  is XORed with the  $i$ -th plaintext block  $P_i$  to produce the  $i$ -th ciphertext block  $C_i$ .



# Properties of OFB

- Synchronous block/stream cipher.
- Errors in one ciphertext block **do not propagate**.
- Conceals plaintext patterns.
- Seems inherently sequential – no parallel implementation known.
- It is **proved** that if  $E$  is a pseudo random permutation, then OFB is **resistant** to **chosen plaintext attacks**.
- Can be **batch preprocessed** (on both ends).

Will discuss additional modes when we get to DES and AES (lecture 3).