CISE 7350
Network Security and Risk Analysis

Chapter 2 - Cryptography

Combining Plaintext with Keystream

- Can do it a different ways:
  - XOR
  - If text, can add to key (mod 26)

XoR Example

- can encrypt by XORing plaintext with keystream
- Example: plaintext = “Chappelle”

c	*C*	*h*	*a*	*p*	*p*	*e*	*l*	*l*	*e*
c(bin)	01000011	01101000	01100001	01110000	01110000	01100101	01101100	01101100	01100101
key	33	72	31	79	82	74	126	89	2
key(bin)	00100001	01001000	00011111	01001111	01010010	01001010	01111110	01011001	00000010
c XOR k	01100010	00100000	01111110	00111111	00100010	00101111	00010010	00110101	01100111

Question: if I have the keystream, how do I decrypt?

– XOR it the keystream with the ciphertext

Combining plaintext with keystream

- Besides XOR, for text, you can add key to data mod 26
- Example:

Plaintext	D	A	V	E	A	T	T	E	L
	3	0	21	4	0	19	19	4	11
Key	J	O	M	P	K	R	L	Q	E
	9	14	12	15	10	17	11	16	4
P + K (mod 26)	12	14	7	19	10	10	4	20	15
ciphertext	M	O	H	T	K	K	E	U	P

Vernam Cipher

- type of one-time pad
- combine an arbitrarily long nonrepeating series of numbers with the plaintext to form ciphertext
- originally implemented as a paper tape attached to a teletype machine

Book Ciphers

- construct a poor man's one-time pad
- get “randomness” from:
  - Novels
  - Newspapers
  - telephone books
  - pieces of music
  - decks of cards

Key Reuse

- What happens if you use the same key twice?

$$C_1 = P_1 \oplus K \qquad C_2 = P_2 \oplus K$$

$$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K$$

$$\ldots = P_1 \oplus P_2$$

much easier to solve

Characteristics of Good Cipher

- Shannon *Communication Theory of Secrecy Systems (1949), pg. 15*
  - Amount of secrecy should be proportional to value
  - Key needs to be transmitted/memorized → should be as short as possible
  - Encryption/decryption should be as simple as possible
  - Errors shouldn't propagate
  - Size of the ciphertext should be the same as plaintext

Stream and Block Ciphers

- **stream ciphers**
  - encrypt one symbol (bit, byte, or word) at a time
  - encrypt the *ith symbol with the ith part of the keystream*
- **block ciphers encrypt larger blocks of plaintext**
  - block size → usually 64 bits or more
  - encrypt all blocks with the same key

Block Ciphers

- Plain substitution ciphers that we've discussed
  - example:
    - A→K
    - B→D
    - C→Q
    - ...
- ciphers that operate on 64-bit blocks
  - example:
    - 0x0000 0001 → 0x81A7 C961
    - 0x0000 0002 → 0xB132 8DC5
    - ...

Bits to encode 64-bit block ciphers

- ciphers that operate on 64-bit blocks
  - example:
    - 0x0000 0001 → 0x81A7 C961
    - 0x0000 0002 → 0xB132 8DC5
    - ...
- How many bits would it take to encode this?
  - If we made a table, there would be:
    - 2^{64} entries
    - each entry would be 64 bits long
    - $2^{64} * 2^{6} = 2^{70}$ bits

Block Cipher Features

- **Block size:** in general larger block sizes mean greater security.
- **Key size:** larger key size means greater security (larger key space).
- **Number of rounds:** multiple rounds offer increasing security.
- **Encryption modes**: define how messages larger than the block size are encrypted, very important for the security of the encrypted message.

Bits to encode 64-bit block ciphers

- So for larger block sizes, we have to do something different
- Goal:
  - generate a 1→1 mapping
  - make it look as random as possible
  - don't store all possible input/output pairs

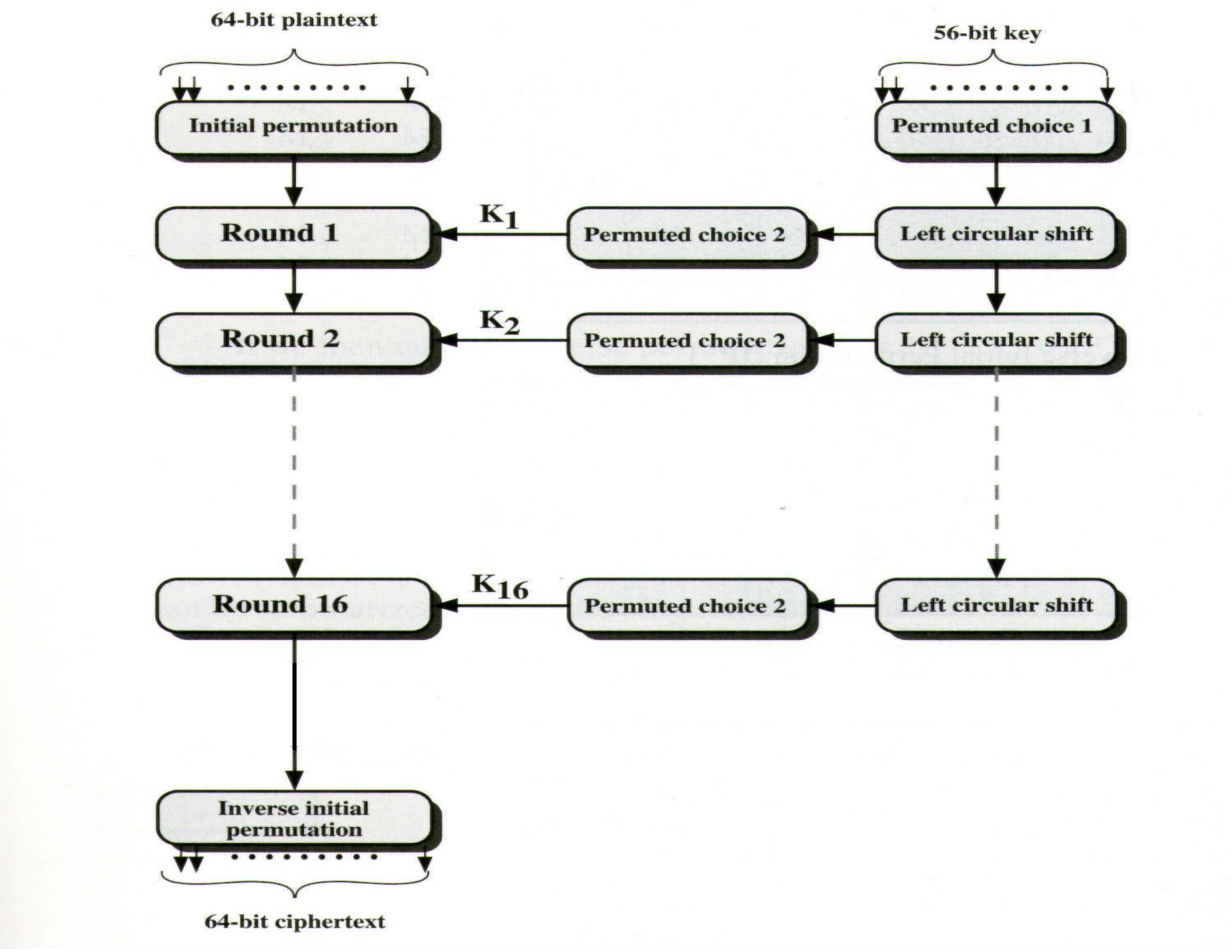
Data Encryption Standard (DES)

- Early 70s non-military crypto research unfocused
- National Bureau of Standards (now NIST) wanted algorithm which:
  - is secure
  - Open
  - Efficient
  - useful in diverse applications
- IBM Lucifer algorithm submitted
- DES based on Lucifer controversies over:
  - reduced key size
  - design (of S-boxes)

DES Features

- Block size = 64 bits
- Key size = 56 bits
- Number of rounds = 16
- 16 intermediary keys, each 48 bits

DES Rounds



64-bit plaintext
56-bit key
Initial permutation
Permuted choice 1
Round 1
K1
Permuted choice 2
Left circular shift
Round 2
K2
Permuted choice 2
Left circular shift
Round 16
K16
Permuted choice 2
Left circular shift
Inverse initial permutation
64-bit ciphertext

DES Decryption

- Decryption uses the same algorithm as encryption, except that the subkeys K1, K2, …K16 are applied in reversed order

Cryptlanalysis of DES

- Brute Force
  - Known-Plaintext Attack
  - Try all 2^{56} possible keys
  - Requires constant memory
  - Time-consuming
  - DES challenges: (RSA)
    - msg="the unknown message is :xxxxxxxx"
    - CT=" C1 | C2 | C3 | C4"
    - 1997 Internet search: 3 months
    - 1998 EFF machine (costs $250K): 3 days
    - 1999 Combined: 22 hours

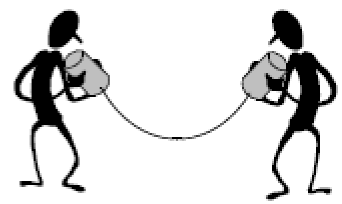
Cryptlanalysis of DES

- DES uses a 56-bit key, this raised concerns about brute force attacks.
- One proposed solution:double DES.
- Apply DES twice using two keys, K1 and K2.
  - $C = E_{K2} [E_{K1} [P]]$
  - $P = D_{K2} [D_{K1} [C]]$
- This leads to a 2x56=112 bit key, so it is more secure than DES. **Is it?**

Meet-in-the-Middle Attack

- Goal: given the pair (P, C) find keys K1 and K2.
- Based on the observation:

$$C = E_{K_2}[\ E_{K_1}[\ P\ ]\ ]$$
$$D_{K_2}[\ C\ ] = E_{K_1}[\ P\ ]$$



The attacks has higher chance of succeeding if another pair (P’, C’) is available to the cryptanalysis

Meet-in-the-Middle Attack

$C = E_{K2}[\, E_{K1}[\, P \,]\,]$
$E_{K1}[\, P \,] = D_{K2}[\, C \,]$

Attack, assumes the attacker knows two pairs (P,C) and (P'C'):

- Encrypt P with all 2^{56} possible keys K_1
- Store all pairs (K_1, $E_{K1}[P]$), sorted by $E_{K1}[P]$.
- Decrypt C using all 2^{56} possible keys K_2
- For each decrypted result, check to see if there is a match $D_{K2}(C) = E_{K1}(P)$.
- If yes, try another pair (P', C')
- If a match is found on the new pair, accept the keys K_1 and K_2.

Why two Pairs (P,C)?

- DES encrypts 64-bit blocks, so for a given plaintext P, there are 2^{64} potential ciphertexts C.
- Key space: two 56-bit key, so there are 2^{112} potential double keys that can map P to C.
- Given a pair (P, C), the number of double keys (K_1, K_2) that produce $C = E_{K_2}[E_{K_1}[P]]$ is at most $2^{112}/2^{64} = 2^{48}$
- Therefore, for a pair (P, C), 2^{48} false alarms are expected.

Why two Pairs (P,C)?

- With one more pair (P', C'), extra 64-bit of known text, the alarm rate is $2^{48}/ 2^{64} = 1/2^{16}$
- If meet-in-the-middle is performed on two pairs (P, C) and (P', C'), the correct keys K_1 and K_2 can be determined with probability $1 - 1/2^{16}$.
- Known plaintext attack against double DES succeeds in 2^{56} as opposed to 2^{55} for DES (average).
- **The 112-bit key provides a security level similar to the 56-bit key.**

Triple DES

- Use three different keys

Encrypt: $C = E_{K_3}[\, D_{K_2}[\, E_{K_1}[P]\,]\,]$
Decrypt: $P = D_{K_3}[\, E_{K_2}[\, D_{K_1}[C]\,]\,]$

- Key space is 56 x 3 = 168 bits
- No known practical attack
  against it.

Modes of Operation

- Not in the textbook (but useful. used in many contexts.)
- Suppose that we have a message longer than 64 bits.
  - How do we use a 64-bit block cipher to encrypt it?
- Modes of operation:
  - Electronic Code Book Mode (ECB)
  - Cipher Block Chaining Mode (CBC)
  - Output Feedback Mode (OFB)
  - Cipher Feedback Mode (CFB)
  - Counter Mode (CTR)

Electronic Code Book

- Message is broken into independent blocks of *block_size* bits;
- Electronic Code Book (ECB): each block encrypted separately.
- **Encryption: $c_i = E_k(x_i)$**
- **Decryption: $x_i = D_k(c_i)$**

Electronic Code Book

Pros

- simple
- encrypt in any order
- encrypt in parallel
- example (database):
  - database stored in encrypted form
  - can change a single record without having to re-encrypt the other records
- no error propagation

Cons

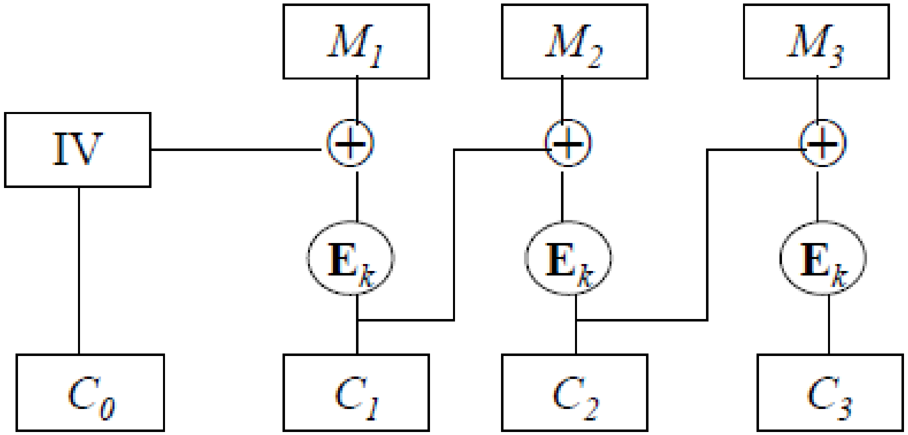
- plaintext block always encrypts to the same ciphertext block
  - could theoretically create a codebook of plaintext → ciphertext pairs
- patterns aren't hidden
  - tcp headers, mail headers, etc., long strings of 0's.
- insertion attacks
- replay attacks

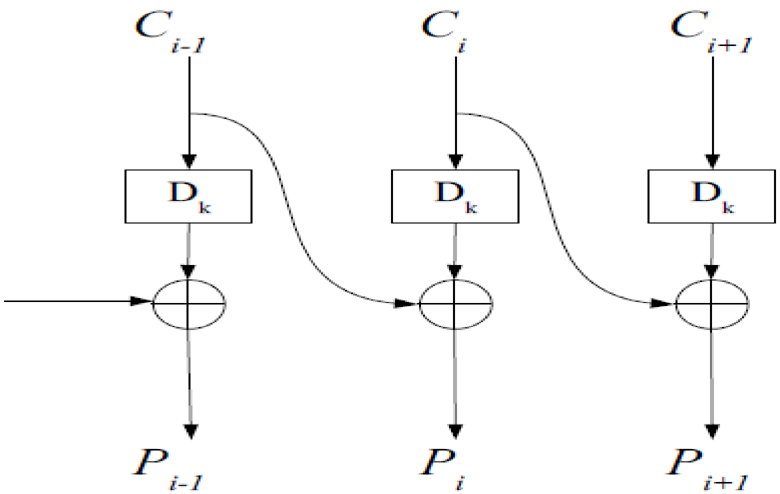
CBC

- Cipher Block Chaining (CBC): next input depends upon previous output

  Encryption: $C_i = E_k(M_i \oplus C_{i-1})$, with $C_0 = IV$

  Decryption: $M_i = C_{i-1} \oplus D_k(C_i)$, with $C_0 = IV$





CBC: Why does it work?

Encryption

$$C_i = E_k(P_i \oplus C_{i-1})$$

Decryption

$$P_i = C_{i-1} \oplus D_k(C_i)$$

$$\ldots = C_{i-1} \oplus (P_i \oplus C_{i-1})$$

$$\ldots = P_i$$

Initialization Vector

- To form the ciphertext of block *i*
  - XOR the plaintext of block *i* with the ciphertext of block *i-1*.
- What do we do with the 1st block?
  - use block of random data known to both the sender and receiver
  - called **initialization vector (IV)**

- Make two identical plaintext blocks encrypt to two different ciphertext blocks
- But if all of the preceding ciphertext blocks are also the same, we're in trouble
- What if the entire message is the same?
  - The entire cipher text will repeat
- Solution?
  - Use Different IVs

CBC: Error Propagation

- What happens if there is an error in block *i*?
  - Error affects block *i* and block *i+1*?
- Why does it only affect block *i* and *i+1* and nothing later?

CBC: Error Propagation

- block i is flawed: so C_i becomes C_i^1

$$C_{i-1} \oplus D_k(C_i^1) = P_i^1$$

- block i+1 arrives

$$C_i^1 \oplus D_k(C_{i+1}) = C_i^1 \oplus P_{i+1} \oplus C_i = P_{i+1}^1$$

- block i+2 arrives

OK

$$C_{i+1} \oplus D_k(C_{i+2}) = C_{i+1} \oplus P_{i+2} \oplus C_{i+1} = P_{i+2}$$

Problem

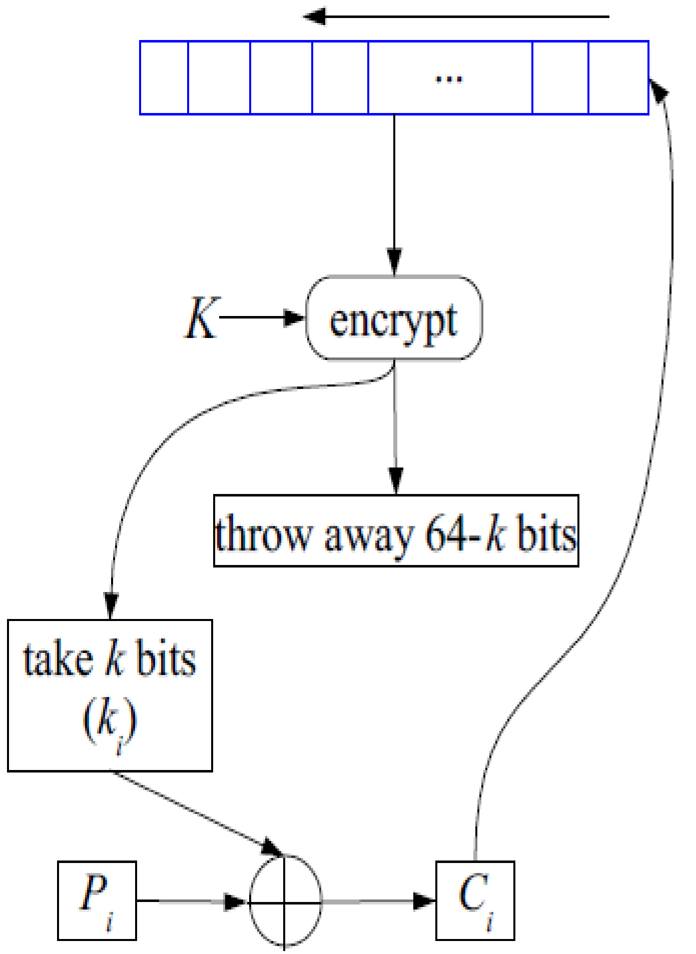
- Suppose that we're doing telnet and we'd like to use CBC mode?
- Blocks are 64 bits
- 1)We'd have either:
  - wait until we've typed several characters OR
  - pad each so that we have a full block
- 2) We'd have to transmit 64-bits of ciphertext for every 8 bits of plaintext

Use Block Ciphers to construct Stream Ciphers

- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter Mode (CTR)
  - Common properties:
    - uses only the encryption function of the cipher
    - both for encryption and for decryption
    - malleable: possible to make predictable bit changes

CFB Encryption

- Fill up a block sized IV
- Encrypt it
- Take the left-most *k bits*
  - throw away the rest
  - left bits are next bits of keystream
- XOR with plaintext
- Result is ciphertext
- Feed it back into queue

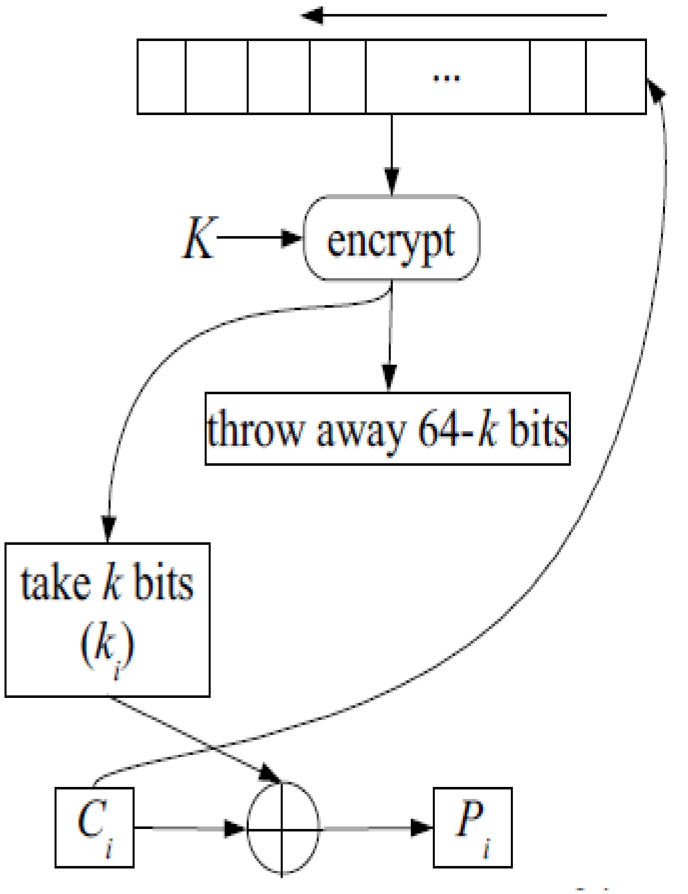


CFB Decryption

- Recall: with stream ciphers
  - decrypt by XOR'ing the keystream with ciphertext
- CFB decryption:
  - receiver starts with the same IV
  - encrypt IV
  - select left-most *k* bits
  - XOR with ciphertext to recover plaintext
  - feed *k* bits of ciphertext back into queue

CFB Decryption

- receiver starts with same IV
- encrypt IV
- select left-most *k bits*
- XOR with ciphertext to recover plaintext
- feed *k bits of ciphertext* back into queue



CFB Properties

- Randomized encryption
- A ciphertext block depends on all preceding plaintext blocks; reorder affects decryption
- Errors propagate for several blocks after the error, but the mode is selfsynchronizing (like CBC).
- Decreased throughput.
  - Can vary the number of bits feed back, trading off throughput for ease of use
- Sequential encryption

Output Feedback Mode

- *Idea*: run a block cipher as a synchronous stream cipher
- *Encryption*

$$C_i = P_i \oplus S_i$$

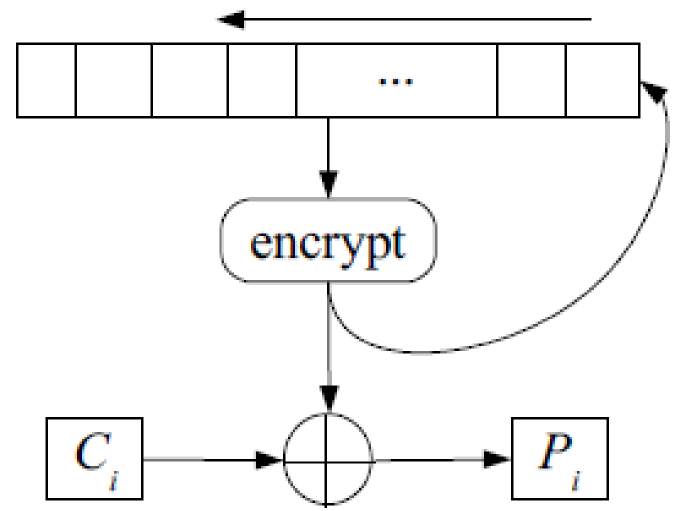
$$S_i = E_K(S_{i-1})$$

- *Decryption:*

$$P_i = C_i \oplus S_i$$

$$S_i = E_K(S_{i-1})$$

Update internal state



- IV should be unique, but doesn't have to be secret

OFB Errors

- Error propagation
  - no error extension
  - single bit error in ciphertext causes single bit error in corresponding plaintext
- What happens if the sender and receiver lose sync?
  - *disaster*
  - must be able to:
    - detect sync errors
    - automatically recover with a new IV to regain sync

OFB Security Problems

- Don't want keystream to repeat
- Should chose the feedback size to be the same as the block size
  - *e.g.* so if you're using a 64-bit block size, you should use 64-bit OFB
  - the smaller the block size, the more often the keystream will repeat

Counter Mode (CTR)

- Use sequence numbes as input to the algorithm
- Just like OFB, except:
  - you don't feed the output back into the shift register
  - just add a counter to the register
- It doesn't matter
  - what the starting counter value is
  - what the increment amount is
- Only requirement: sender and receiver must agree

Counter Mode

- Synchronization problems: same as OFB
- Why use it?
  - compute keystream in parallel
  - precompute the keystream
  - random access
  - simple

Summary

- Block ciphers encrypt chunks of plaintext at a time all with the same key
- Stream ciphers encrypt symbol *i* of the plaintext by combining it with symbol *i* of the key
- With very simple primitive ops (substitutions, permutations, shifts, XORs) DES was strong
- DES insecure by today's standards (56-bit keys too short). 3DES strong but slow.
- CBC, OFB, CFB, CTR → hide patterns
  - Additionally OFB, CFB, CTR fast
  - Get the best of both stream and block ciphers