

CT437

# COMPUTER SECURITY AND FORENSIC COMPUTING

## BLOCK CIPHERS

Dr. Michael Schukat

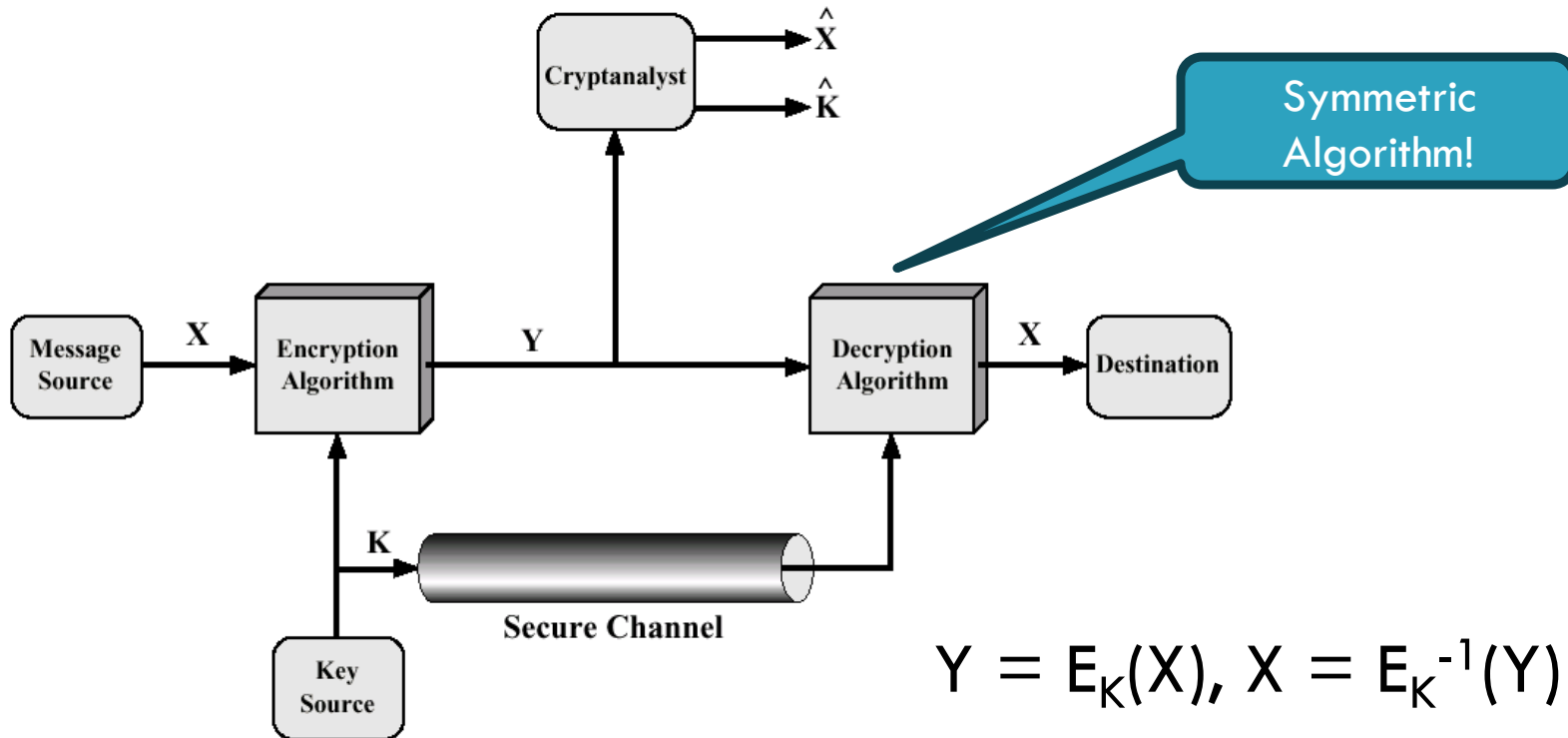


# Lecture Overview

2

- This lecture provides an introduction to one of the fundamental building blocks to provide confidentiality, namely **block ciphers**, thereby covering the following:
  - Symmetric versus Public Key Algorithms
  - Block ciphers versus Stream Ciphers
  - Building Blocks of modern Block Ciphers
  - Modes of operation of block ciphers
  - Examples for modern block ciphers

# Recall: Model of Conventional Cryptosystem



# Symmetric Key Algorithms

4

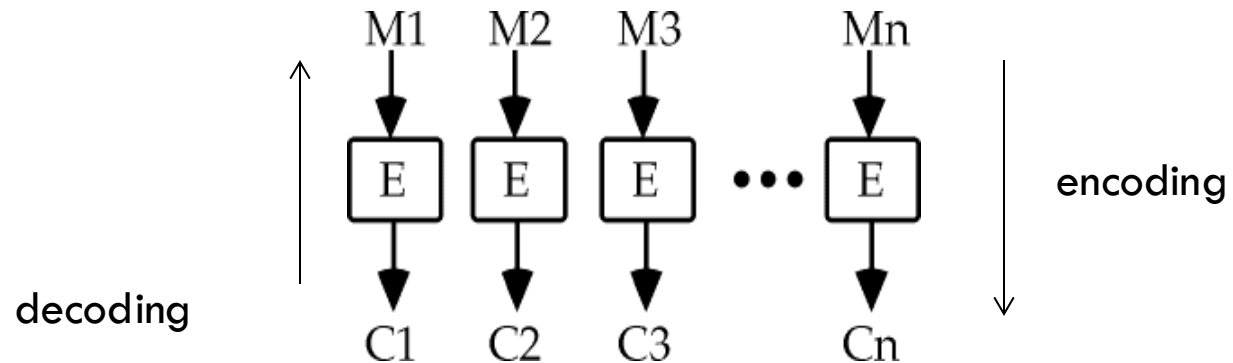
- Also called ciphers for traditional / conventional / single key / private key encryption
- Here the encryption key can be calculated from the decryption key and vice versa
  - ▣ Normally both keys are the same
- The algorithm / cipher itself is public, i.e. is not a secret
- If the key is disclosed, communications are compromised
- The key is also **symmetric**, parties are equal
- Hence methods does not protect sender from receiver forging a message & claiming is sent by sender  
→ nonrepudiation is usually not provided

# Public-Key Algorithms

- Also called ciphers for two key / asymmetric cryptography
- These involve the use of two keys:
  - ▣ a **public key**, which may be known by anybody, and can be used to encrypt messages, and verify signatures (later!)
  - ▣ a **private key**, known only to the recipient/owner, used to decrypt messages, and sign (create) signatures
- The keys are **asymmetric**, because they are not equal
- While the public and its private key are interlinked, it is mathematically very hard to recover the private key via its public key
- Public key algorithms are generally significantly slower than symmetric algorithms, therefore these are often used to securely convey symmetric algorithm' (session) keys
- More later!

# Block Ciphers versus Stream Ciphers

- In a block cipher the data (e.g. text, video, or a network packet) to be encrypted is broken into blocks  $M_1$ ,  $M_2$ , etc. of  $K$  bits length, each of which is then encrypted
- The encryption process is like a substitution on very big characters – 64 bits or more




- In contrast, **stream ciphers** ( $\rightarrow$  next lecture) only process one bit or one byte at a time

# Example Block Cipher Transformation

7

P: 0000000000000000 .... 1111111111111111



C: 0101001010100101 .... 0110110110110010

- Block size  $K$  is 16 bits
- If there wasn't a cipher available for this transformation, we'd require a table with  $2^{16}$  entries
  - Not feasible
- Note that there are  $(2^{16})!$  possible substitutions

# Block Ciphers and Padding

- Messages are usually not multiples of  $K$  bits
- Padding is a way to take data that may or may not be a multiple of the block size for a cipher and extend it out so that it is
  - ▣ It is only applied to the last block that is being encrypted
- Padding must be reversable, i.e., one must be able to distinguish between relevant content and padding bytes in a block



# Padding Algorithms

9

- Let  $N$  be the number of bytes required to make a final block of data the same size as the block size
  - PKCS7 padding works by appending  $N$  bytes with the binary value of  $N$ ; example:

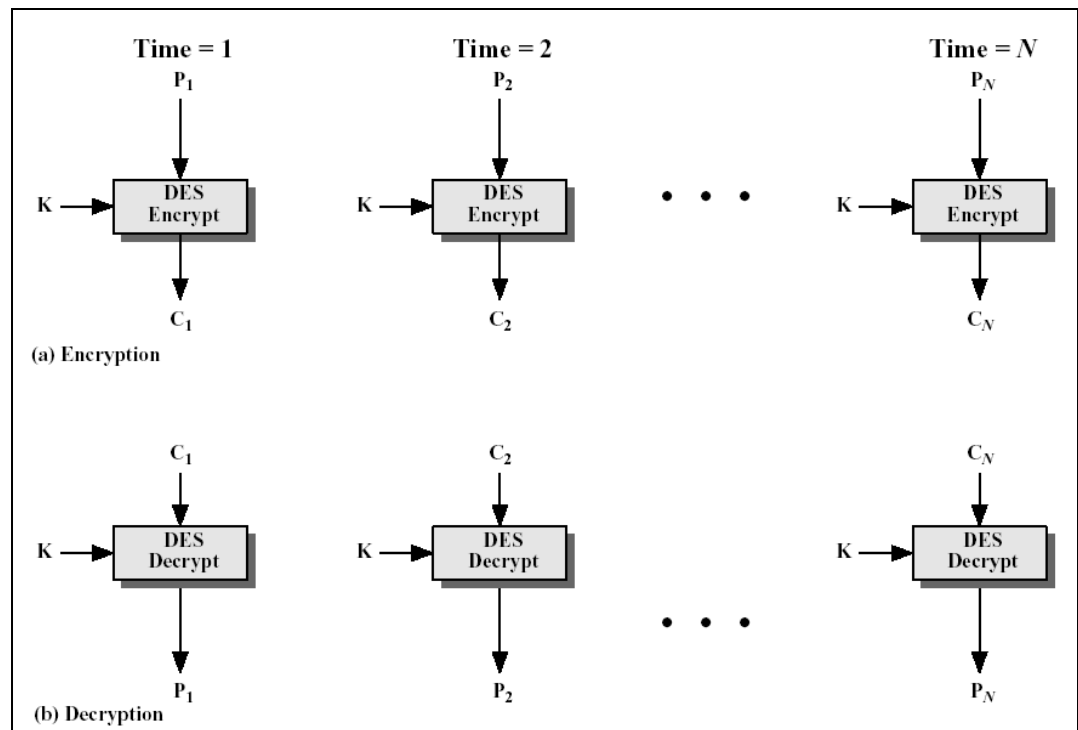
```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

- ANSI X9.23 padding works by appending  $N-1$  bytes with the value of 0 and a last byte with the value of the binary value of  $N$ ; example:

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |
```

# Modes of Operation: Electronic Codebook (ECB) Mode

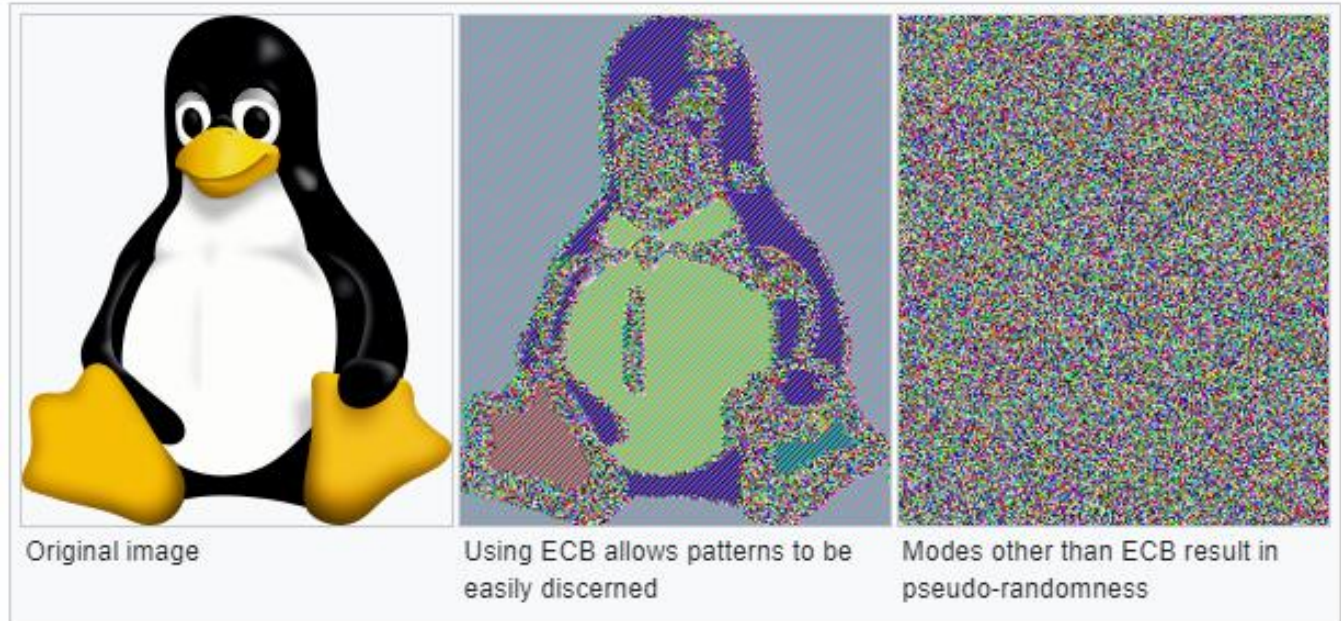
- These modes comprise different strategies on how to use block ciphers (to encode messages)
- What are the advantages / disadvantages of the ECB mode?
- Note that “DES” in the diagram on the right is just an example for a block cipher



# Characteristics and Limitations of ECB Mode

11

Source:  
Wikipedia



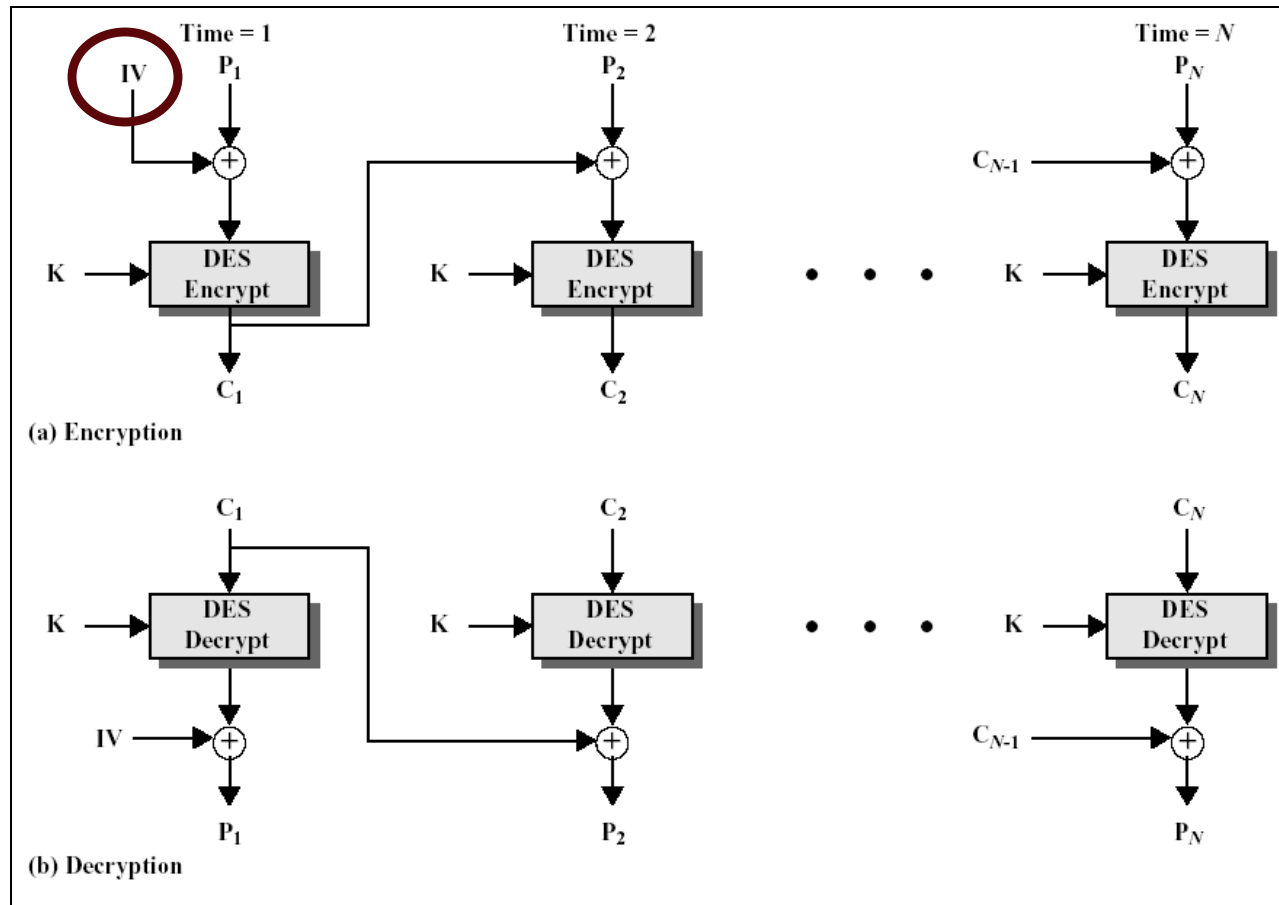
ECB	
Electronic codebook	
Encryption parallelizable	Yes
Decryption parallelizable	Yes
Random read access	Yes

# Why would one avoid the Electronic Codebook Mode?

12

- In ECB mode identical plaintext blocks result in identical ciphertext blocks
- An attacker, while not able to decode the ciphertext blocks, would conclude that the encoded data is repetitive / structured, i.e. could be an image
- However, random data (e.g. long cryptographic keys) could be still encoded in ECB
  - ▣ E.g., a 512-bit key could be encrypted using a 128-bit block cipher using ECB mode

# Modes of Operation: Cipher Block Chaining (CBC) Mode



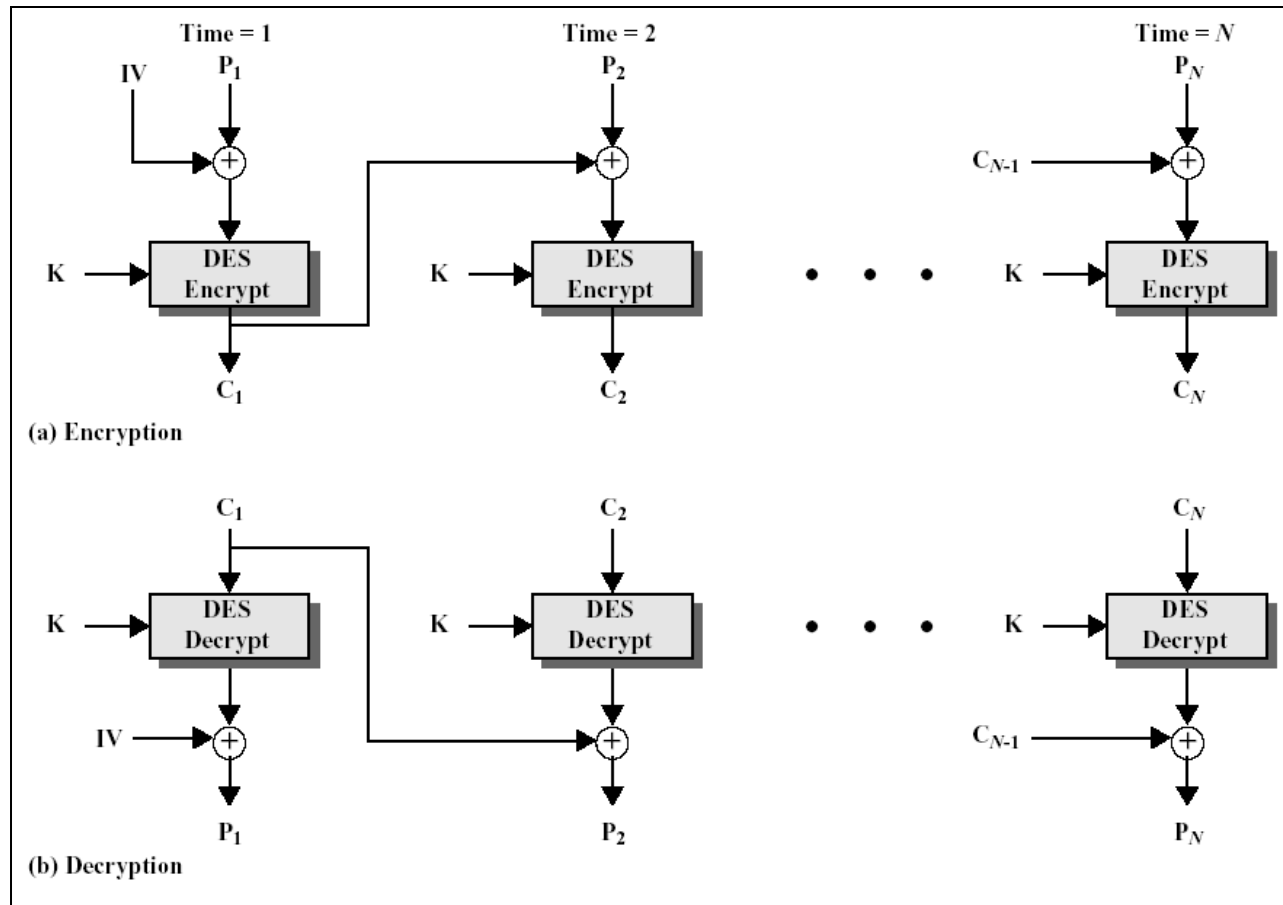
# The Initialisation Vector (IV)

14

- An IV is a block of bits that is used by several modes (including CBC) to randomise the encryption
- An initialization vector has different security requirements than a key, so the IV usually does not need to be secret
- For most block cipher modes it is important that an initialisation vector is never reused under the same key, i.e. it must be a **cryptographic nonce**
  - ▣ Hence distinct ciphertexts are generated even if the same plaintext is encrypted multiple times using the same key
- In data communication, the IV may be attached as plaintext to the encrypted data, and send to the receiver

# Modes of Operation: Cipher Block Chaining (CBC) Mode

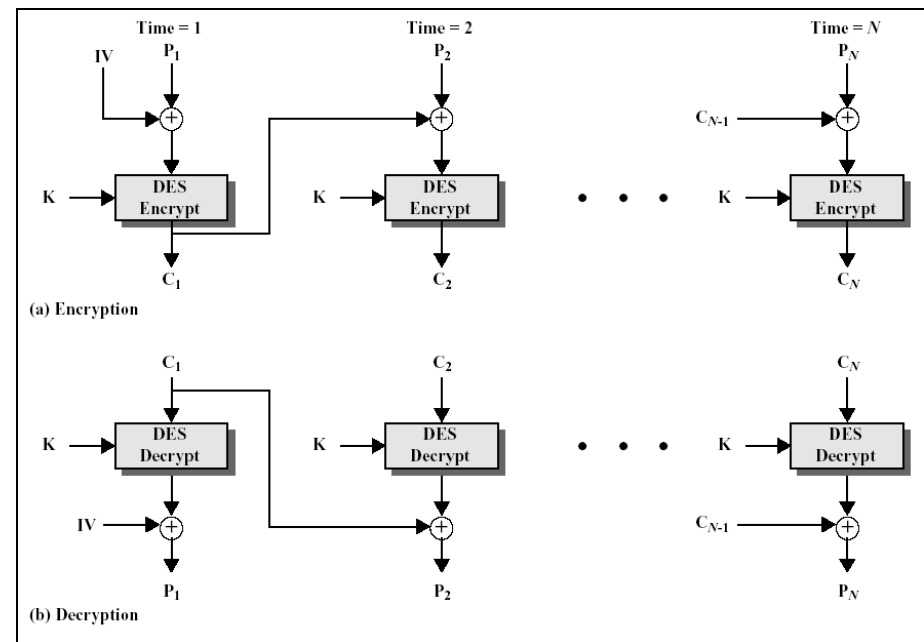
- What are the characteristics of the CBC mode?



# Modes of Operation: Cipher Block Chaining (CBC) Mode

- ❑ For encryption, a one-bit change in a plaintext or IV affects all following ciphertext blocks
- ❑ Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct
  - ▣ This could be problematic if the IV was kept a secret and is used to expand the algorithm's key space

CBC	
Cipher block chaining	
Encryption parallelizable	No
Decryption parallelizable	Yes
Random read access	Yes

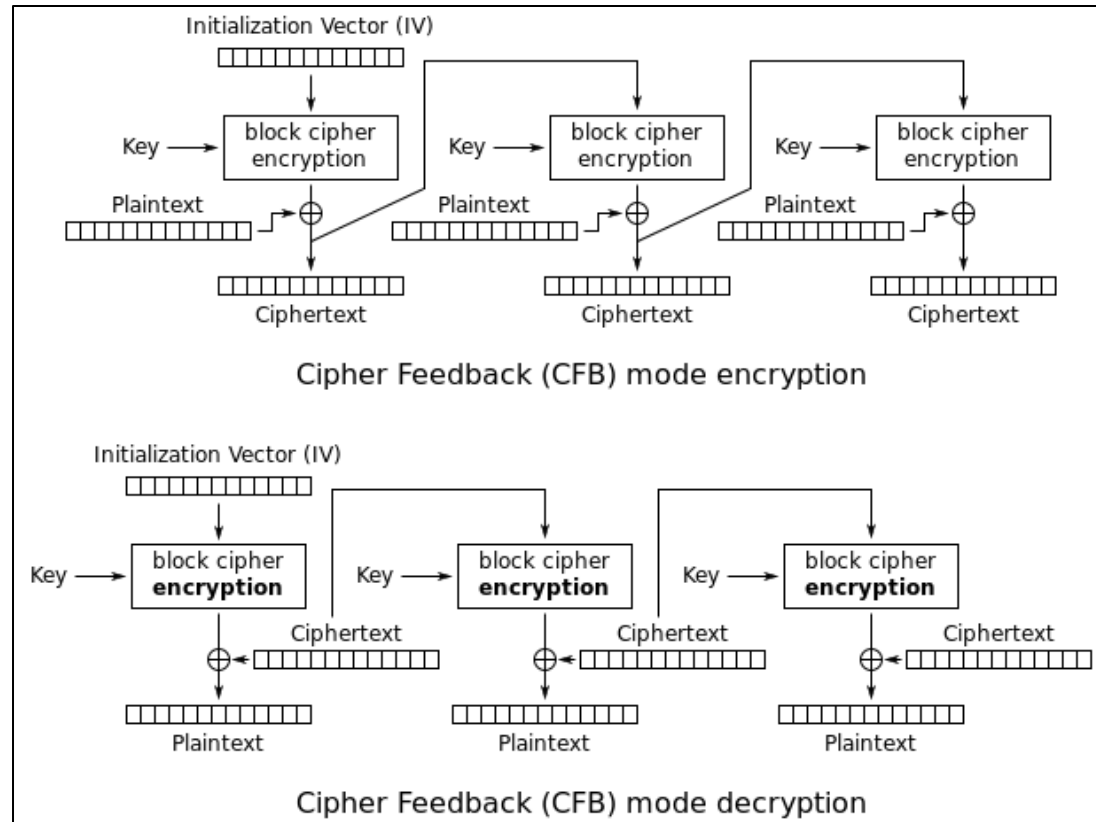




# Full Block Cipher Feedback (CFB) Mode

17

- Note that CFB only requires block encryption for both encoding and decoding

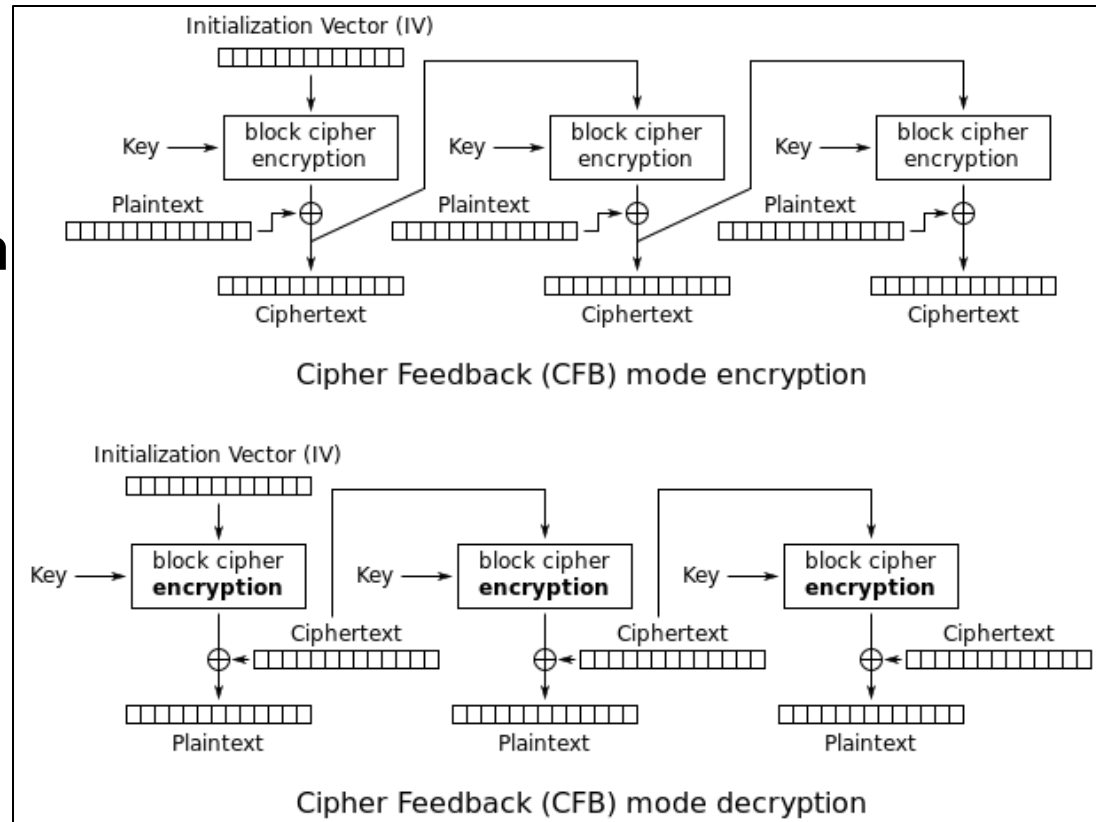


# Full Block Cipher Feedback (CFB) Mode

18

- This mode is particularly useful, when decryption needs to be fast (i.e., parallelisable)
- Note that CFB only requires block encryption for both encoding and decoding

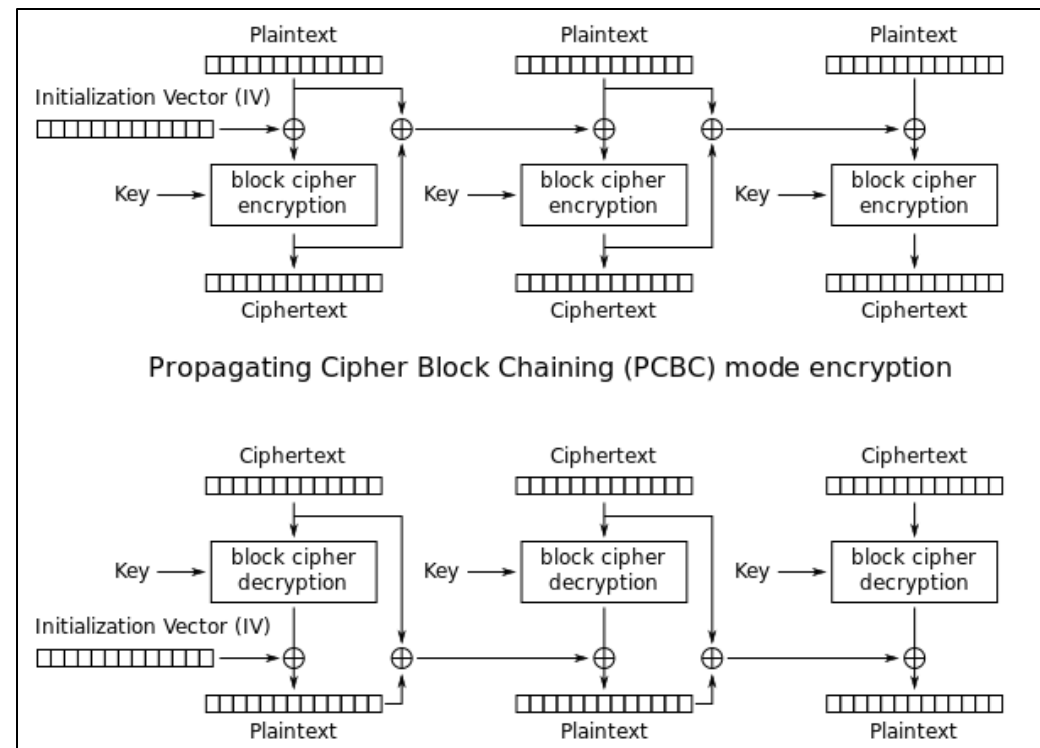
CFB	
Cipher feedback	
Encryption parallelizable	No
Decryption parallelizable	Yes
Random read access	Yes



# Propagating Cipher Block Chaining (PCBC) Mode

19

- This mode fixes the IV problem of CBC, i.e., decrypting PCBC with the incorrect IV causes all blocks of plaintext to be corrupt

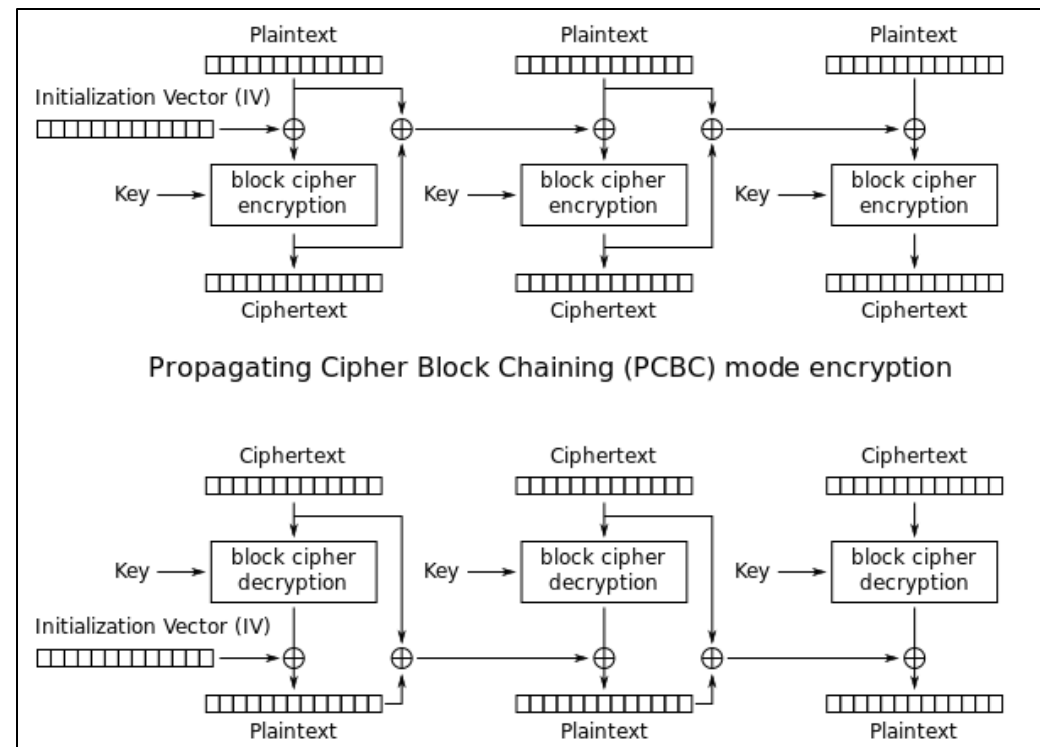


# Propagating Cipher Block Chaining (PCBC) Mode

20

- ❑ This mode fixes the IV problem of CBC, i.e., decrypting PCBC with the incorrect IV causes all blocks of plaintext to be corrupt

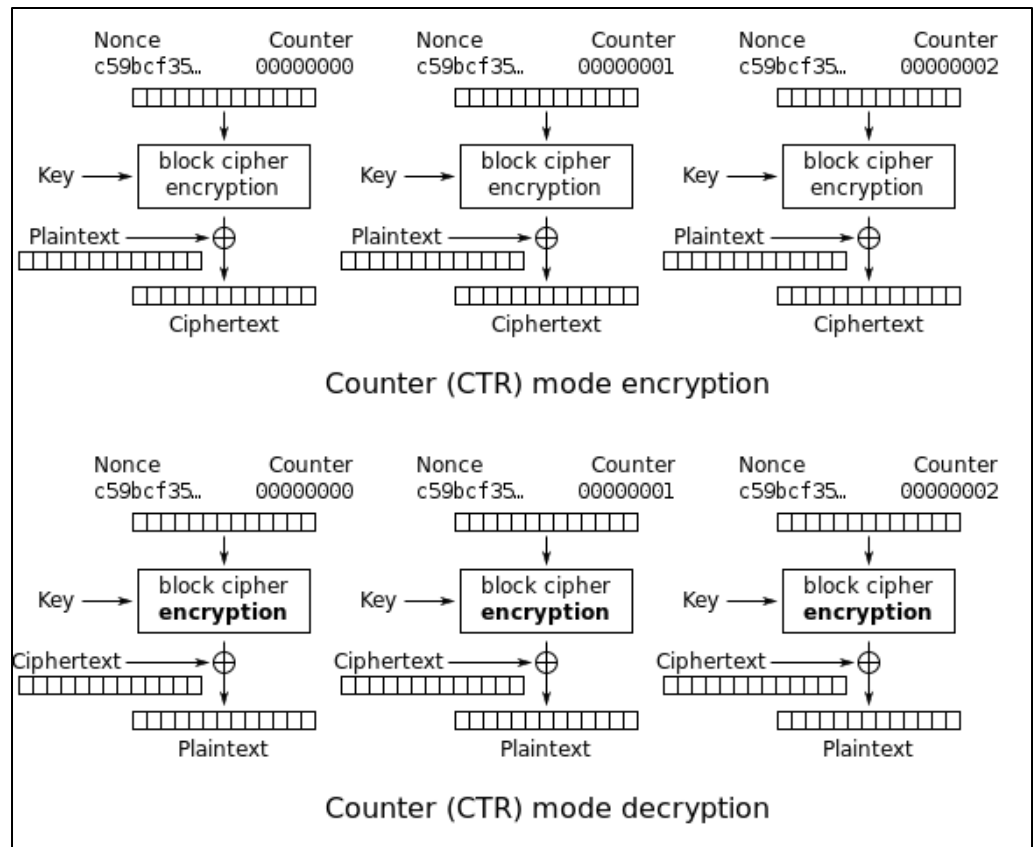
PCBC	
Propagating cipher block chaining	
Encryption parallelizable	No
Decryption parallelizable	No
Random read access	No



# Counter (CTR) Mode

21

- Here the random nonce (which is equivalent to an IV) is complemented with an incremented counter value
- Note that CTR only requires block encryption for both encoding and decoding

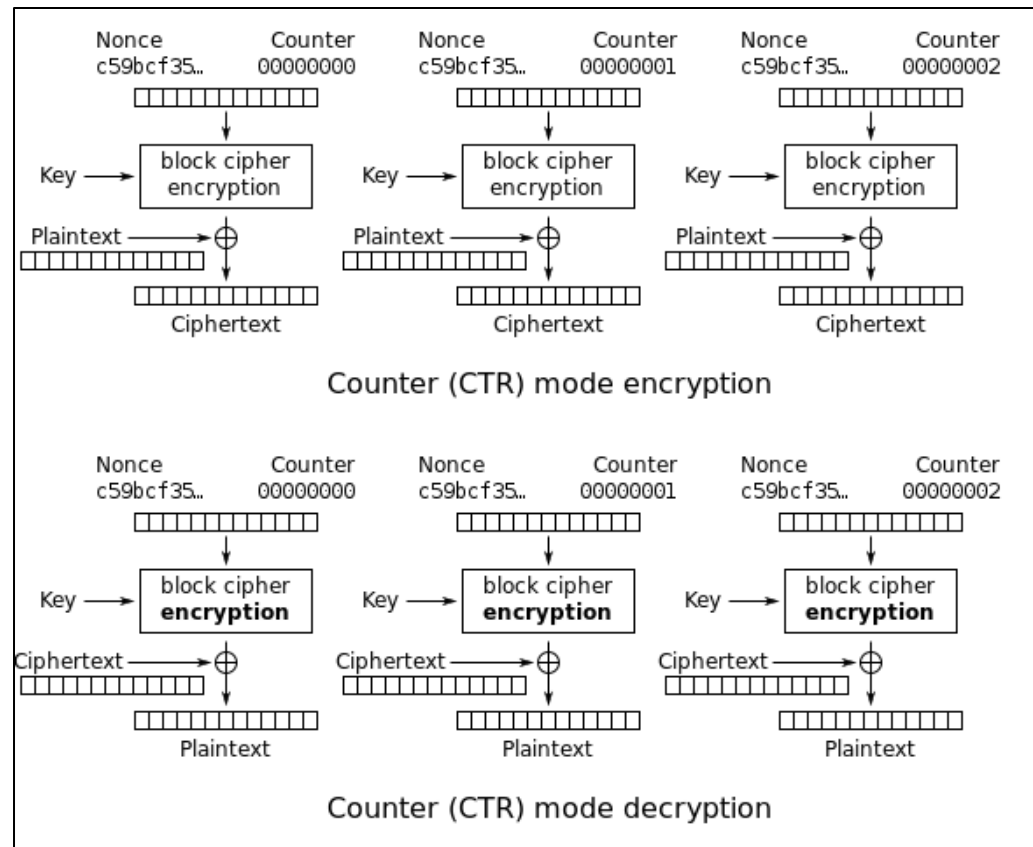


# Counter (CTR) Mode

22

- Here the random nonce (which is equivalent to an IV) is complemented with an incremented counter value
- Note that CFB only requires block encryption for both encoding and decoding

CTR	
Counter	
Encryption parallelizable	Yes
Decryption parallelizable	Yes
Random read access	Yes



# Another Question ...

23

- Assume you have a large data file stored on your computer that needs to be encrypted / decrypted on-the-fly, potentially with random block access
- You pick a suitable block cipher for this task
- **Which mode of operation would you choose?**

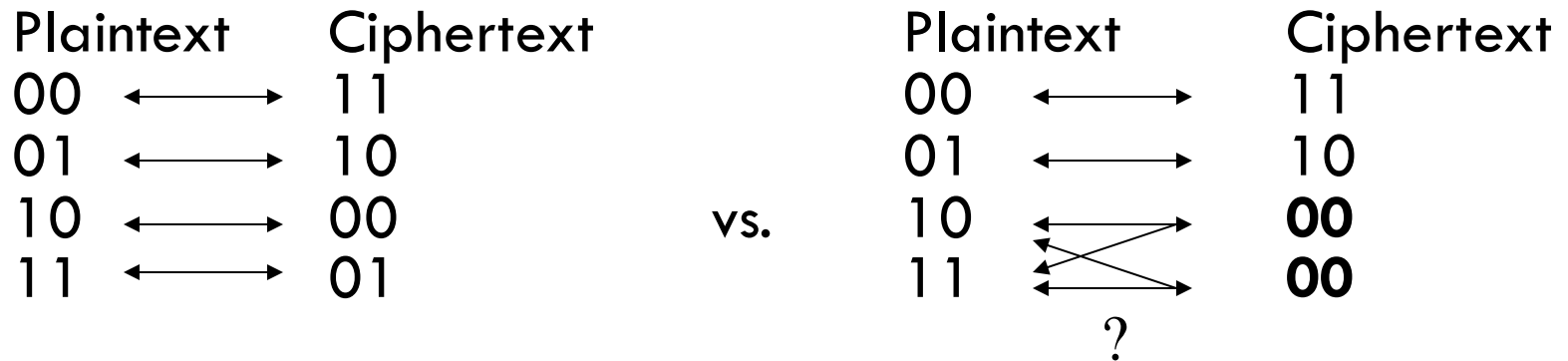
# Building Block Ciphers

- Confusion, diffusion and the avalanche effect
- Using SP-Networks
- Using Feistel Networks



# Important Block Cipher Principle

- Transformations must be reversible (“non-singular”), e.g.,



- There must be a 1:1 association between a n-bit plaintext and an-n-bit ciphertext, otherwise mapping (encryption) is irreversible

# Confusion and Diffusion

- A block cipher needs to completely obscure the statistical properties of original message (obviously)
- Claude Shannon introduced two terms:
  - ▣ **Diffusion** seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible
  - ▣ **Confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible
- Both thwart attempts to deduce the key
- This can be achieved, by applying a cryptographic operation iteratively multiple times, i.e. over multiple rounds
  - ▣ See also the avalanche effect (next slide)

# The Avalanche Effect

- Practically, confusion and diffusion provide for encryption algorithms, where a slight change in either the key or the plaintext results in a significant change in the ciphertext
- The table shows some characteristics of the DES algorithm (later), that encrypts a block over 16 rounds
- A swap of a single bit either in the key or in the plaintext results in an incrementally growing change in the ciphertext (avalanche effect)

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

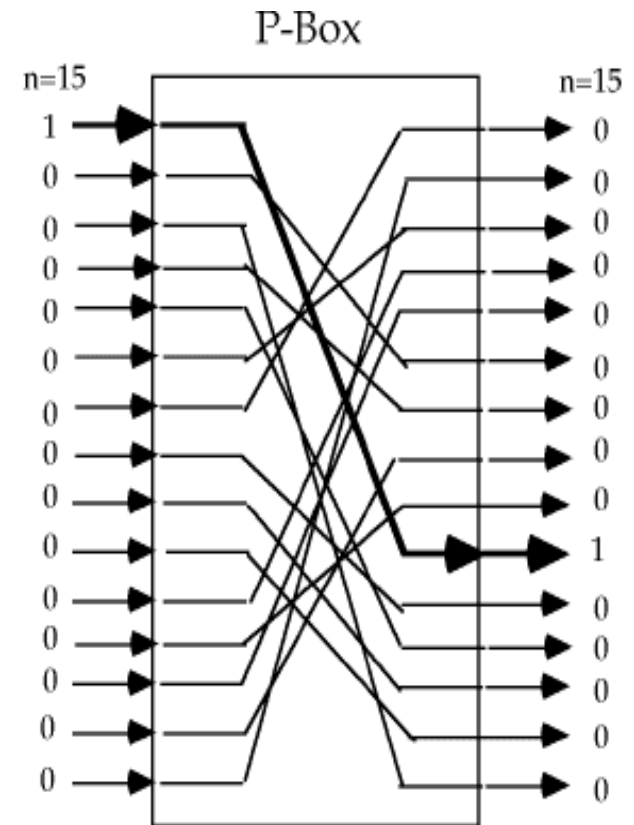
# Block Cipher Building Blocks

28

- In order to build a block cipher efficiently, one needs robust building blocks, that can be easily implemented and tested
  - ▣ The robustness of the block cipher depends on the robustness of its components
- These blocks are combined or iterated through creating a block cipher
- The most common building blocks are:
  - ▣ P-Boxes
  - ▣ S-Boxes
  - ▣ Feistel ciphers

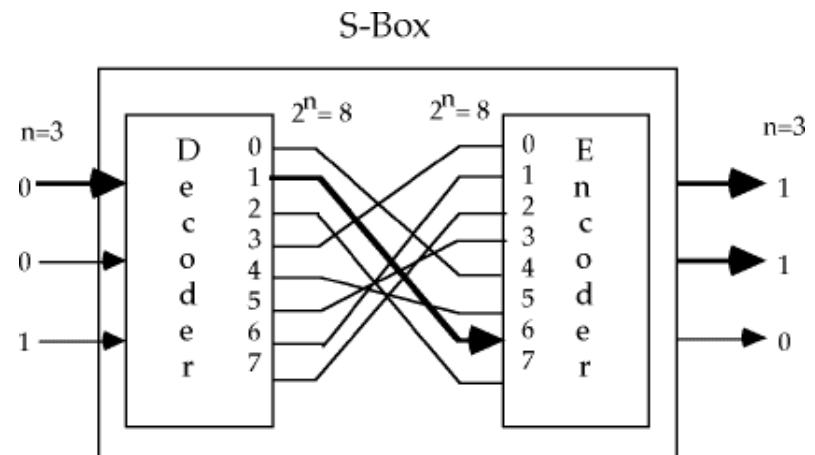
# The Permutation Operation

- A binary word (i.e. block) has its bits reordered (permuted)
  - ▣ Similar to classical transposition ciphers
- This operation is represented by a **P-box** (see diagram)
- Here the re-ordering / internal wiring forms the key
- The example shown allows for  $15! = 1,307,674,368,000$  combinations

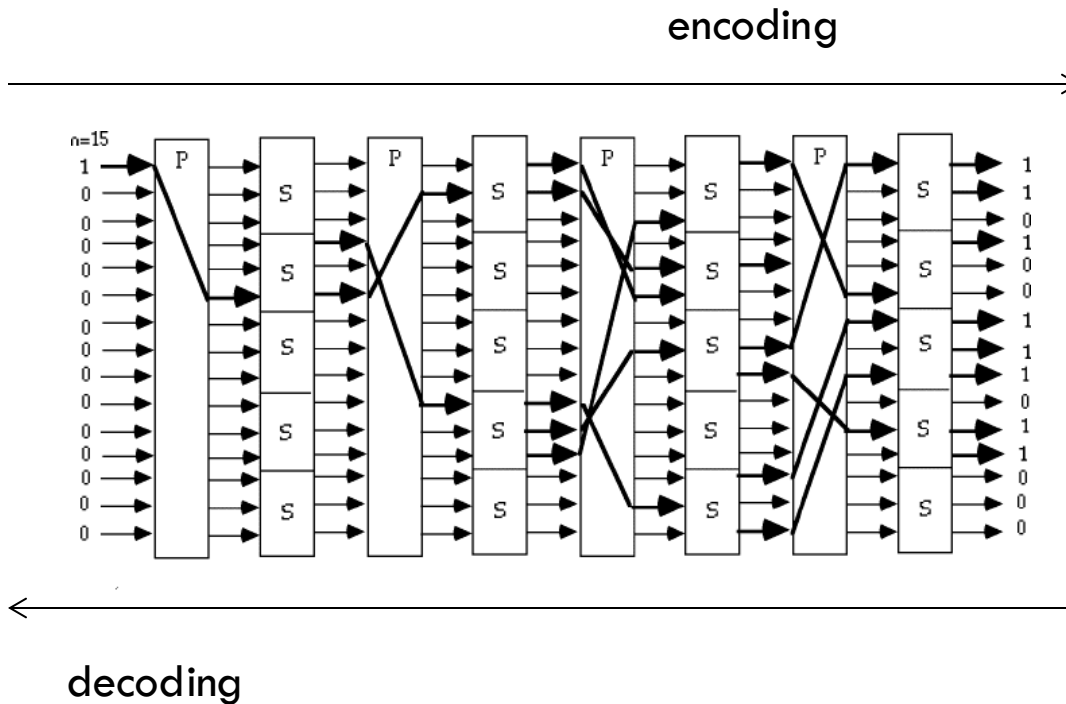


# The Substitution Operation

- A binary word is replaced by some other binary word
- Similar to classical substitution ciphers
- This operation is represented by an **S-box**
- Here the re-ordering / internal wiring forms the key
- The box shown allows for  $8! = 40320$  combinations



# Substitution-Permutation Network



- ◆ The key describes the internal wiring of all S-boxes and P-boxes
- ◆ The same key can be used for encoding and decoding, hence it is a **private key encryption algorithm**
- ◆ The direction of the process determines encoding / decoding

Question:

- How big is the key space for this arrangement?
- How many bits are needed to describe a single S or P box?
- What is the total number of bits required to describe all boxes?

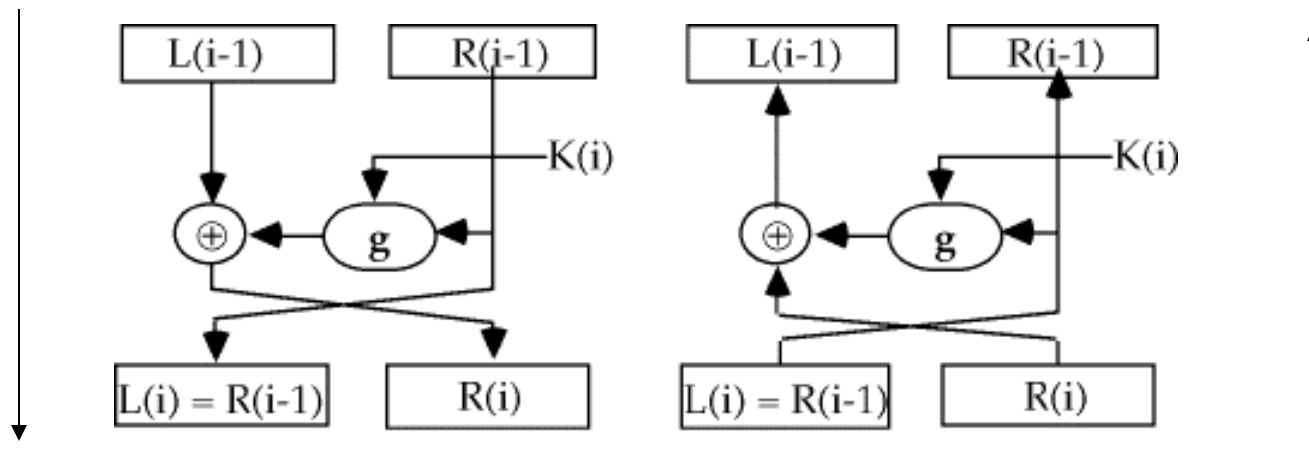
# The Feistel Cipher

- In practice, we need algorithms that can decrypt and encrypt messages using similar code / hardware for both
- An S-P network as seen in the example cannot be easily reversed when implemented in hardware / software
  - ▣ i.e. one needs different functions for encoding / decoding
- In contrast, a **Feistel cipher** is an invertible cipher structure which adapts Shannon's S-P network in an easily invertible structure for encoding and decoding
  - ▣ In fact, it can use other cryptographic building blocks
- It is based on the concept of the **invertible product cipher**
- It was invented by Horst Feistel, who worked at IBM Thomas J Watson Research Labs in early 70's



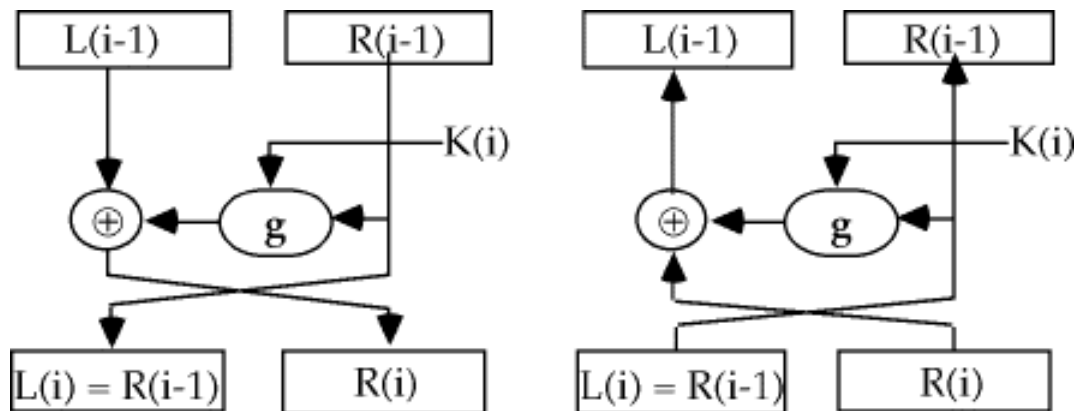
# The Feistel Cipher – A Single Round

- The idea is to partition the input block into two halves,  $L(i-1)$  and  $R(i-1)$ , and use only  $R(i-1)$  in the  $i^{\text{th}}$  round (part) of the cipher
- The function  $g$  incorporates the equivalent of one stage of the S-P network, controlled by part of the key  $K(i)$  known as the  $i^{\text{th}}$  subkey



# The Feistel Cipher – A single Round

- A round of a Feistel cipher can be described functionally as:
  - ▣  $L(i) = R(i-1)$
  - ▣  $R(i) = L(i-1) \text{ EXOR } g(K(i), R(i-1))$



# Recap: Symmetry of Bitwise EXOR

- $A \text{ EXOR } B = C$
- $A \text{ EXOR } C = B$
- $C \text{ EXOR } B = A$

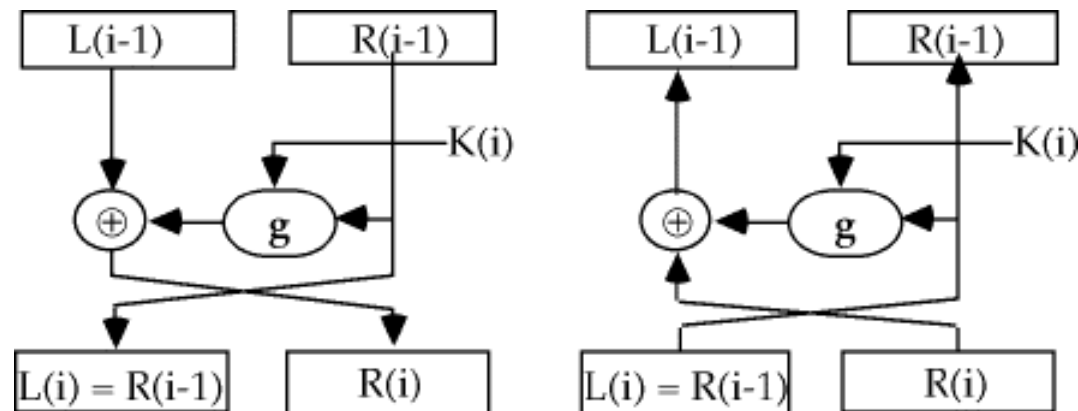
	0	1
0	0	1
1	1	0

# In-Class Activity: Feistel Cipher – Single Round

- Encoding of 01011110:
  - ▣  $L(i - 1) = 0101$
  - ▣  $g(K(i), R(i-1)) = 1001$
  - ▣  $R(i) = ?$
  - ▣ Therefore 01011110 becomes ?

$$R(i - 1) = 1110$$

$$L(i) = ?$$



# Example Feistel Cipher – Single Round

## □ Encoding of 01011110:

$$\square L(i - 1) = 0101 \qquad R(i - 1) = 1110$$

$$\square g(K(i), R(i-1)) = 1001 \qquad L(i) = 1110$$

$$\square R(i) = 0101 \text{ XOR } 1001 = 1100$$

□ Therefore 01011110 becomes 11101100

## □ Decoding of 11101100:

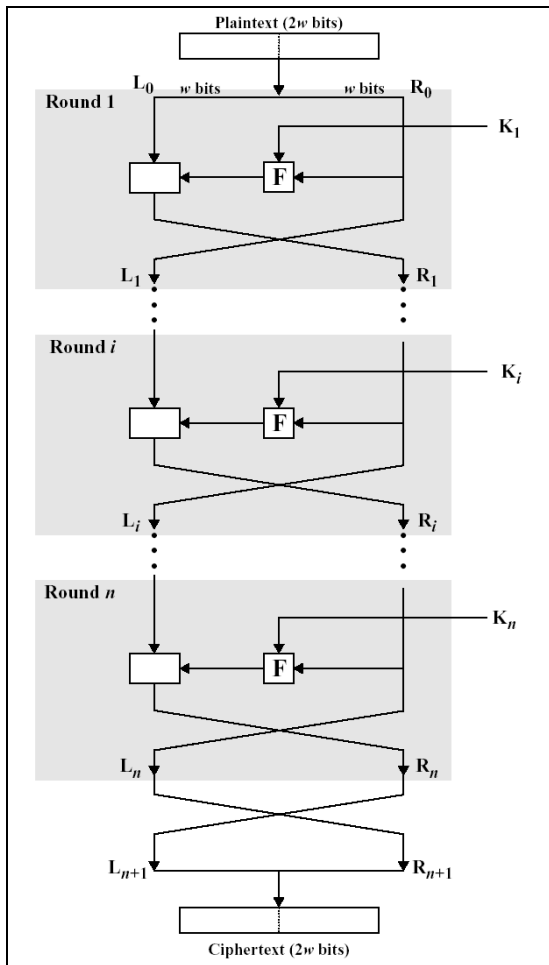
$$\square L(i) = 1110 \qquad R(i) = 1100$$

$$\square g(K(i), R(i-1)) = 1001 \qquad R(i - 1) = 1110$$

$$\square L(i - 1) = 1100 \text{ XOR } 1001 = \underline{0101}$$

□ Therefore 1110 1100 becomes 01011110

# Feistel Network



- Common structure of many modern block ciphers
- It performs multiple transformations (single rounds) sequentially, whereby output of  $i^{\text{th}}$  round becomes the input of the  $(i+1)^{\text{th}}$  round
- Every round gets its own subkey, which is derived from master key
- Decryption process goes from bottom to top

# Feistel Cipher Design Elements

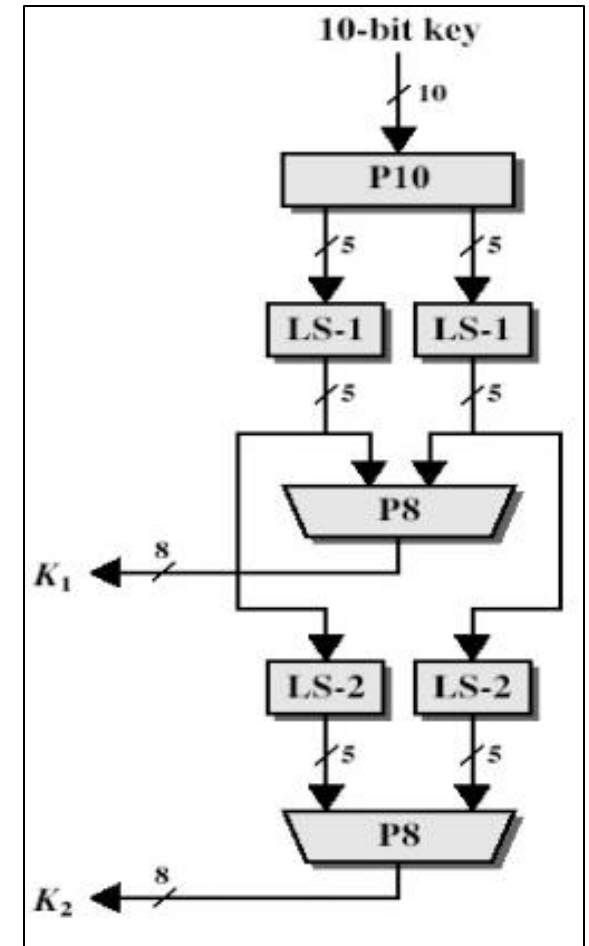


These include

- ❑ Block size (typically 64 – 256 bits)
- ❑ Key size (typically 80 – 256 bits)
- ❑ Number of rounds (typically  $> 16$ )
- ❑ Subkey generation algorithm
- ❑ Round function

# Simple Methods for Subkey Generation

- Here two 8-bit round keys ( $K_1$  and  $K_2$ ) are derived from a 10-bit (master) key:
  - The 10-bit master enters the permutation box (P10)
  - The output is split into 2 parts
  - Each part is left-rotated by one bit (LS-1)
  - Both parts are concatenated and passed into a permutation box (P8)
  - P8 has eight outputs, which make  $K_1$





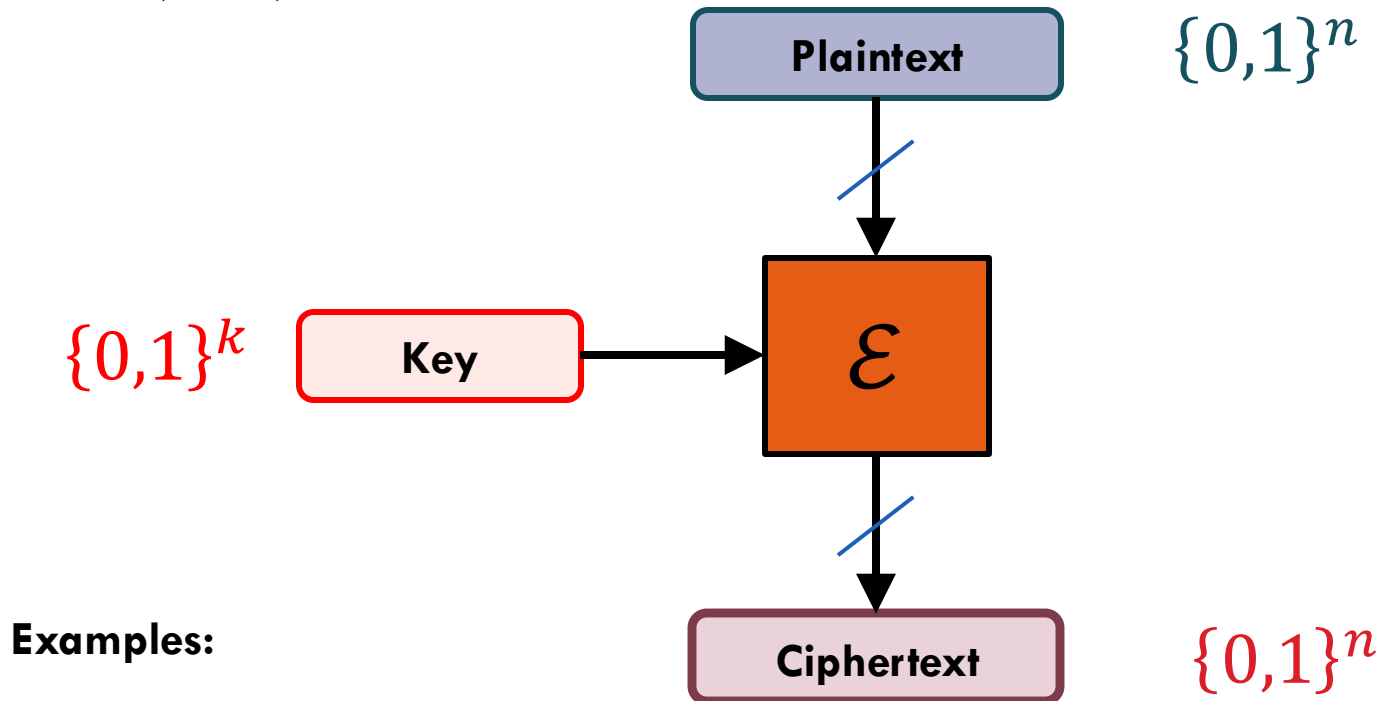
# Block Cipher Examples

- Data Encryption Standard (DES)
- AES

# Common Block Cipher Key and Block Lengths

*Key length*  $k = 80, 128, 192, 256$

*Block length*  $n = 64, 128, 256$

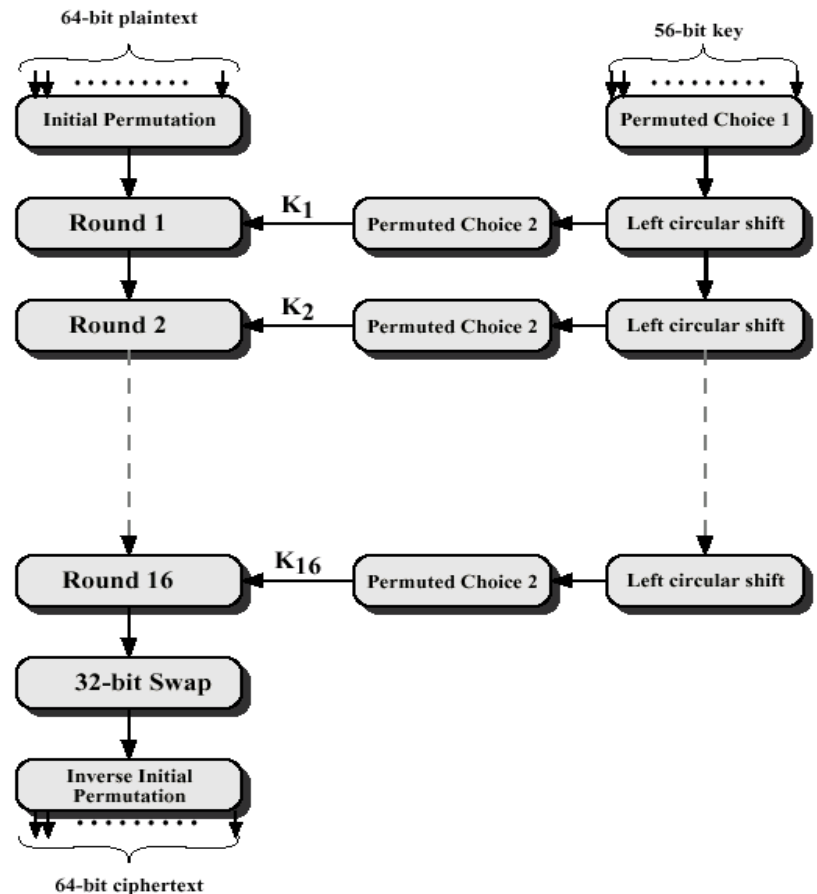


DES:  $k = 56, n = 64$

AES:  $k = 128, 192, 256, n = 128$

# Data Encryption Standard (DES)

- ❑ DES was the first block cipher widely used in industry
- ❑ Introduced in 1976
- ❑ 64-bit block length
- ❑ 56-bit key length
- ❑ Feistel network with 16 rounds and 48-bit subkeys



# The DES Challenge

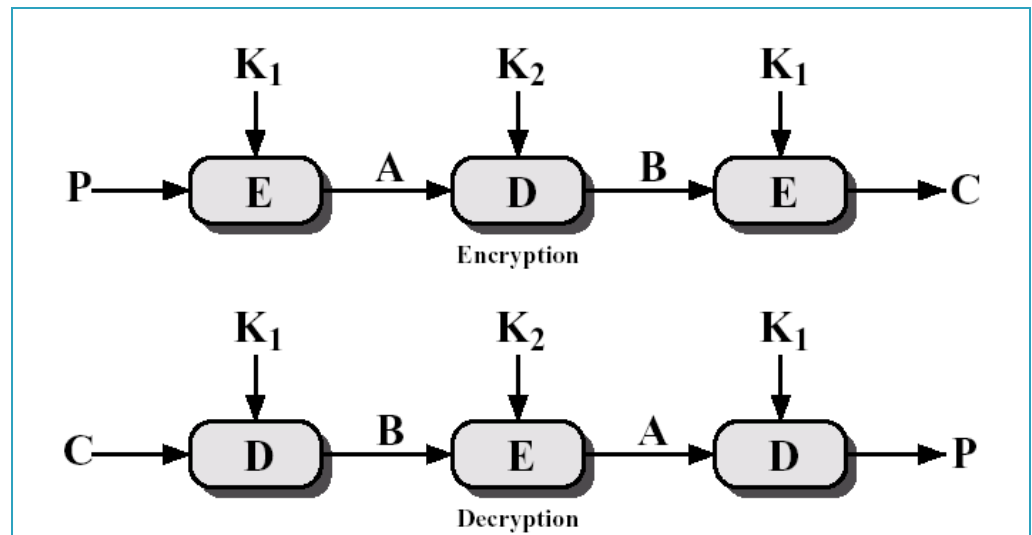
45

- Contest to demonstrate to the US government that 56-bit DES is an ineffective form of encryption
- The goal of the challenge was to decrypt secret messages which had been encrypted with DES

Name	When	Duration	Hardware used
DES I Challenge	June 1997	140 days	Up to 70,000 PC
DES II Challenge	February 1998	41 days	?
DES Challenge II-2	July 1998	56 hours	Custom FPGA Design
DES Challenge III	January 1999	22 hours	~100,000 PC

# Triple DES

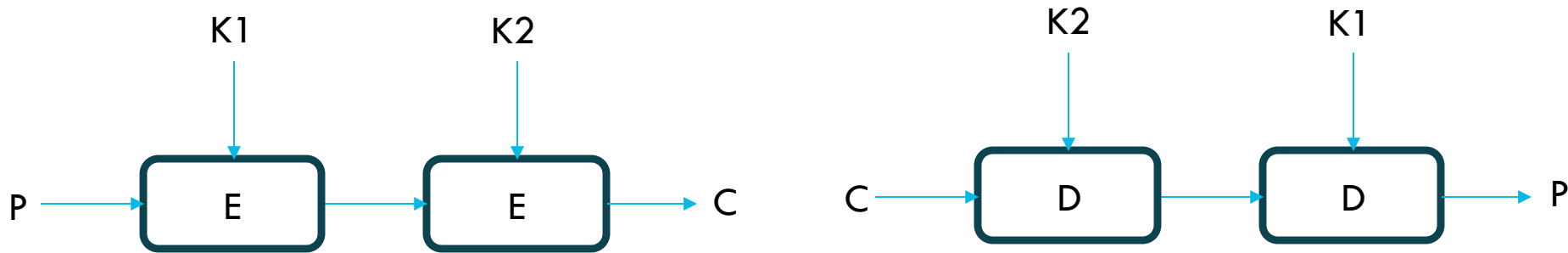
- ❑ DES has been widely used and implemented across many OS and crypto libraries, so attempts were made to increase its active life span
- ❑ This resulted in Triple-DES
- ❑ It is based on three processing stages
- ❑ Note the symmetry in the encoding and decoding process
- ❑ In principal, this concatenation can be applied to every private key block cipher
- ❑ There are 2 common keying options:
  - ▣ 2 keys (as shown in the figure)
  - ▣ 3 keys (one for each stage)



# Double-DES and the Meet-in-the-Middle Attack

47

- Double DES uses two instances of DES with different keys



- While this algorithm uses two independent keys, it is not as sound as it looks
- It is vulnerable to the meet-in-the-middle attack, where an attacker has access to  $P$  and  $C$ , and tries to determine  $K1$  and  $K2$

# Double-DES and the Meet-in-the-Middle Attack

48

- This attack is an example of a space-time tradeoff, where the adversary does the following:
  1. Encrypt  $P$  using every possible key, and copy each key and the resulting cyphertext into a table  $T1$ 
    - $T1$  will have 2 columns and  $2^{56}$  rows
  2. Decrypt  $C$  using every possible key, and copy each key and the resulting plaintext into a table  $T2$ 
    - Again,  $T2$  will have 2 columns and  $2^{56}$  rows
  3. Check for identical cyphertext / plaintext entries in  $T1$  and  $T2$
  4. Their corresponding keys  $K1$  and  $K2$  are key candidates and can be further validated using other plaintext/cyphertext pairs
- Overall, this process requires  $2^{56}$  encryption and  $2^{56}$  decryption attempts, so overall  $2 \times 2^{56} = 2^{57}$  attempts (rather than  $2^{112}$  attempts) are required
- Note that this attack can also be applied to Triple DES, but it would require  $2^{2 \times 56}$  attempts

# Advanced Encryption Standard (AES)

- Successor of DES since 2002
- Based on a S-P network
- Block size is 128-bit
- Key length is configurable can be 128, 192 or 256 bit
- Stronger & faster than Triple-DES
  - ▣  $2 * 56! \ll 128!$
- Envisaged active life until ~2030
- Full specification & design details public
- Algorithm has reference implementations across many programming languages



50

# Breaking Block Ciphers

# Why does Block and Key Length matter?

- Cryptographic algorithms with short block length can be tackled as seen with the substitution cipher
- Large keys and large blocks prevent **brute-force attacks / searches**
  - ▣ Take the ciphertext and try all possible key combinations (or block permutations), until the text is successfully decoded (e.g. until the decryption provides meaningful text)

# Brute Force Search / Attacks

- DES uses 56-bit key has a key space that contains  $2^{56}$  ( $= 7.2 \times 10^{16}$ ) keys
  - ▣ Deemed unsafe since the 1990s
- Triple-DES uses two 56-bit keys. and its key space contains  $2^{112}$  ( $= 5.1 \times 10^{33}$ ) keys
  - ▣ Its use will be prohibited from 2024!
- AES-128 key space contains  $2^{128}$  ( $= 3.4 \times 10^{38}$ ) keys
  - ▣ Generally accepted minimum key length today
- Top secret information requires the use of either AES-192 or AES-256

# Brute Force Search / Attacks

- Always possible to simply try every key
- Most basic attack, effort proportional to key size
- Assume that you either know or recognise plaintext
- GPUs are very good at this task, for example a single RTX 3070 GPU can crack a DES key in  $\sim 215$  days

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/ $\mu$ s	Time required at $10^6$ decryptions/ $\mu$ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu$ s = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu$ s = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu$ s = $5.4 \times 10^{24}$ years	$5.4 \times 10^{18}$ years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu$ s = $5.9 \times 10^{36}$ years	$5.9 \times 10^{30}$ years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu$ s = $6.4 \times 10^{12}$ years	$6.4 \times 10^6$ years

# Side-Channel Attacks

54

- AES is cryptographically sound and there is no practical cryptographic "break" that is faster than a brute-force attack
- However, there are possible side-channel attacks
- Generally, these are attacks on implementations of a cipher on hardware or software systems that inadvertently leak data, e.g.
  - ▣ Timing information (how long does an encryption take)
  - ▣ Cache and memory content (→ HeartBleed)

# Timing Attacks

55

- Here the attacker attempts to compromise a cryptosystem by analysing the time taken to execute a cryptographic algorithm
- Every logical operation in a computer takes time to execute, and the time can differ based on the input
- With precise measurements of the time for each operation, an attacker can work backwards to the input

# Example: Insecure String Comparison (Wikipedia)

56

- Spot the difference?

```
bool insecureStringCompare(const void *a, const void *b, size_t length) {
    const char *ca = a, *cb = b;
    for (size_t i = 0; i < length; i++)
        if (ca[i] != cb[i])
            return false;
    return true;
}
```

versus

```
bool constantTimeStringCompare(const void *a, const void *b, size_t length) {
    const char *ca = a, *cb = b;
    bool result = true;
    for (size_t i = 0; i < length; i++)
        result &= ca[i] == cb[i];
    return result;
}
```

- Note that many such functions in normal (rather than crypto-) libraries are unsafe
  - ▣ Example memcpy() as used in C

# Timing Attacks

57

- In principal, timing attacks can be performed
  - ▣ remotely (e.g. a client measures the response time of a server that encrypts a message)
  - ▣ locally (i.e. in the host machine itself)
- Remote timing attacks are not practical, as variable OS and network latencies effect any measurement
- Local attacks are better, but require the exploit to be installed on the host under attack
- Saying this, many modern CPUs have built-in hardware instructions for AES, which protect against timing-related side-channel attacks



# FYI: More Side-Channel Attacks

58

- **Transient execution CPU vulnerabilities** are vulnerabilities in a computer system in which a speculative execution optimisation implemented in a microprocessor is exploited to leak secret data to an unauthorized party
  - ▣ Example Meltdown and Spectre attack
- In **cache timing attacks** an attacker process deliberately causes page faults and/or cache misses in the target process, and monitors the resulting changes in access times
  - ▣ This can be done despite both processes being otherwise isolated