

Modern data encryption standards

Prepared by Dr. Ayman Khallel

IT373

Information Security

Lecture 3



Course content

1. Overview of security concepts and definitions
2. Symmetric cryptography; modern encryption algorithms (DES, AES)
3. **modern data encryption standards;**
4. block cipher cryptanalysis and usage
5. modern stream ciphers
6. Asymmetric cryptography; public key cryptography, key certificates and management
7. message authentication and hash functions
8. hash algorithms
9. digital signature and authentication protocols
10. electronic mail security, IP security, web security.

Agenda/Outline

- Recap
- Padding
- AES Modes
 - Electronic Codebook Mode (ECB)
 - Cipher Block Chaining Mode (CBC)
 - Output Feedback Mode (CFB)
 - Output Feedback Mode (OFB)
 - Counter Mode (CTR)
- Summary



Recap

Plaintext

- An original message

Ciphertext

- The coded message

Enciphering/encryption

- The process of converting from plaintext to ciphertext

Deciphering/decryption

- Restoring the plaintext from the ciphertext

Cryptography

- The area of study of the many schemes used for encryption

Cryptographic system/cipher

- A scheme

Cryptanalysis

- Techniques used for deciphering a message without any knowledge of the enciphering details

Cryptology

- The areas of cryptography and cryptanalysis

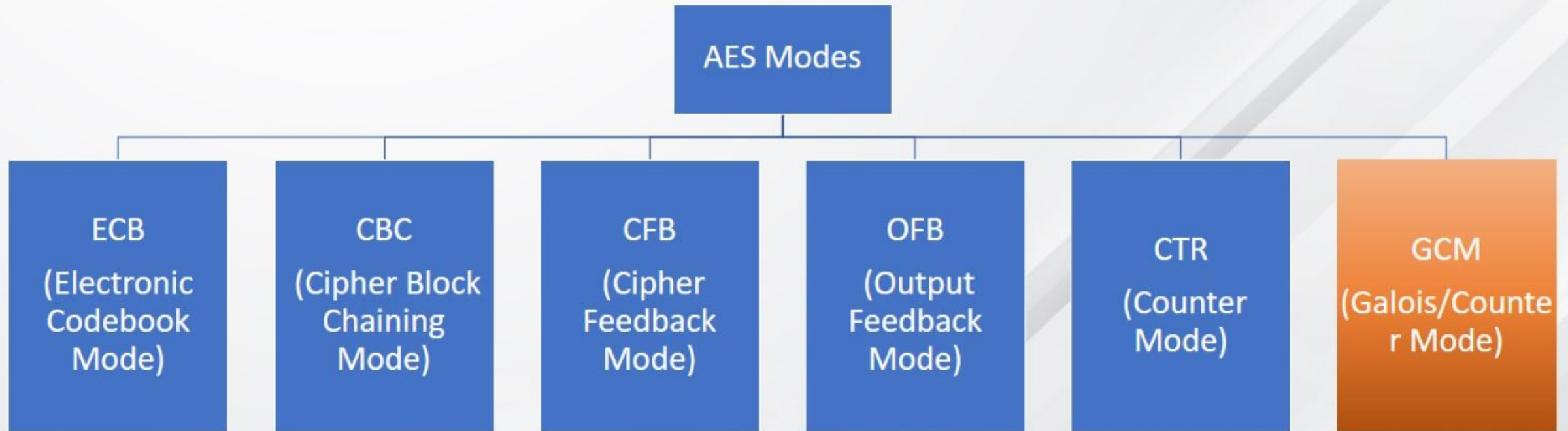
Recap: Practical Security Issues

- Typically, symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block
- Electronic codebook (ECB) mode is the simplest approach to multiple-block encryption
 - Each block of plaintext is encrypted using the same key
 - Cryptanalysts may be able to exploit regularities in the plaintext
- Modes of operation
 - Alternative techniques developed to increase the security of symmetric block encryption for large sequences
 - Overcome the weaknesses of ECB

Padding in AES

- AES (Advanced Encryption Standard) works on blocks of fixed size (128 bits). Padding ensures that plaintext data is of the required length (a multiple of 16 bytes).
- Common Padding Schemes:
 1. PKCS#7 Padding
 - Appends bytes where each byte's value equals the number of bytes that are added.
 - Example:
If 5 bytes are needed: 05 05 05 05 05.
 2. Zero Padding
 - Appends 0x00 bytes to fill the block.
- **Important:**
Padding must be removed correctly after decryption to avoid security issues and data corruption.

aa	aa	aa	05
aa	aa	aa	05
aa	aa	aa	05
aa	aa	05	05



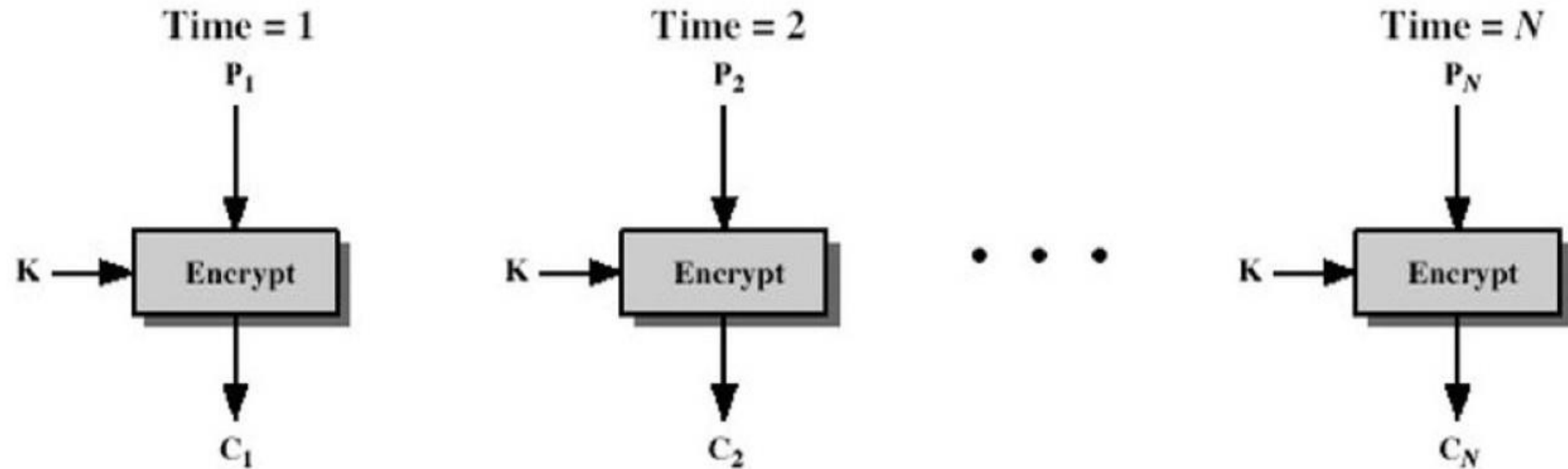
Electronic Codebook (ECB)

How it Works: ECB encrypts each block independently. If the same block of plaintext appears multiple times in the data, it will produce the same block of ciphertext.

Advantage: Simplicity and speed. Easy to parallelize since each block is treated separately.

Disadvantage: Vulnerable to pattern recognition because identical plaintext blocks generate identical ciphertext blocks. This makes it easy to reverse-engineer if there are repeated patterns in the plaintext.

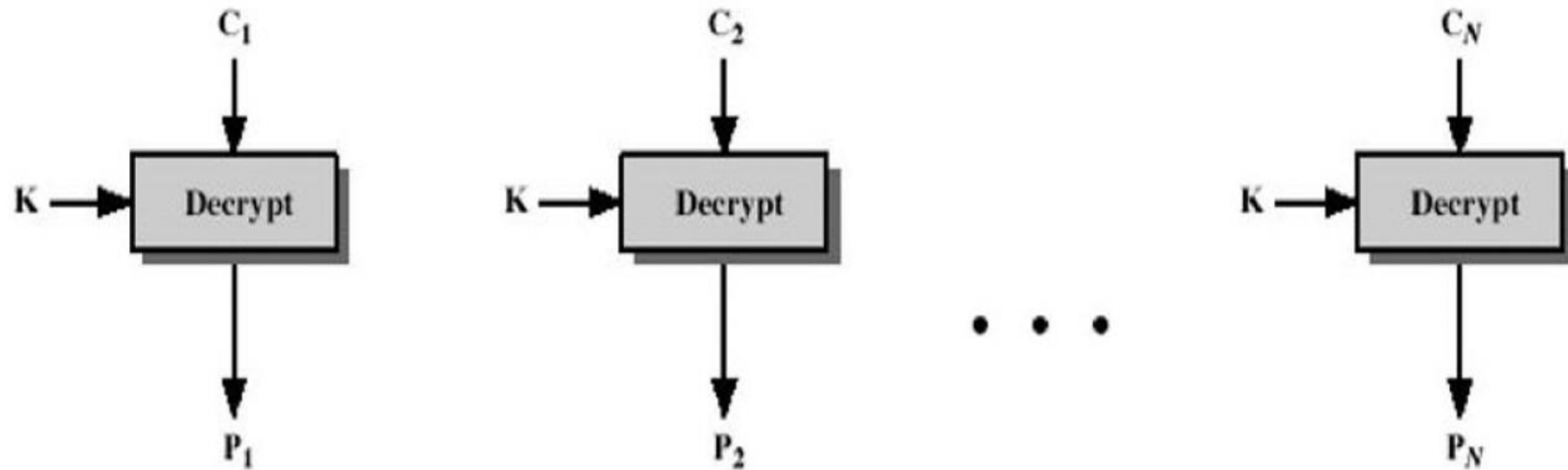
Electronic Codebook (ECB)



(a) Encryption



Electronic Codebook (ECB)



(b) Decryption



Python Code to Encrypt using AES (ECB Mode):

```
# pip install pycryptodome
from Crypto.Cipher import AES

# Key generation
key = b'1111222233334444' # 16 bytes = 128-bit key

# Ensure plaintext is exactly 16 bytes (or a multiple of 16)
plaintext = b'This is a test!!' # 16 bytes long

# Encrypt the data using ECB mode
cipher = AES.new(key, AES.MODE_ECB)
ciphertext = cipher.encrypt(plaintext)
print(f"Ciphertext: {ciphertext.hex()}")
```

Output:

```
Ciphertext: aec97d6c2150fa7baf97f37bfea3069c
```

Python Code to Decryption using AES (ECB Mode):

```
from Crypto.Cipher import AES

# Key generation
key = b'1111222233334444' # 16 bytes = 128-bit key

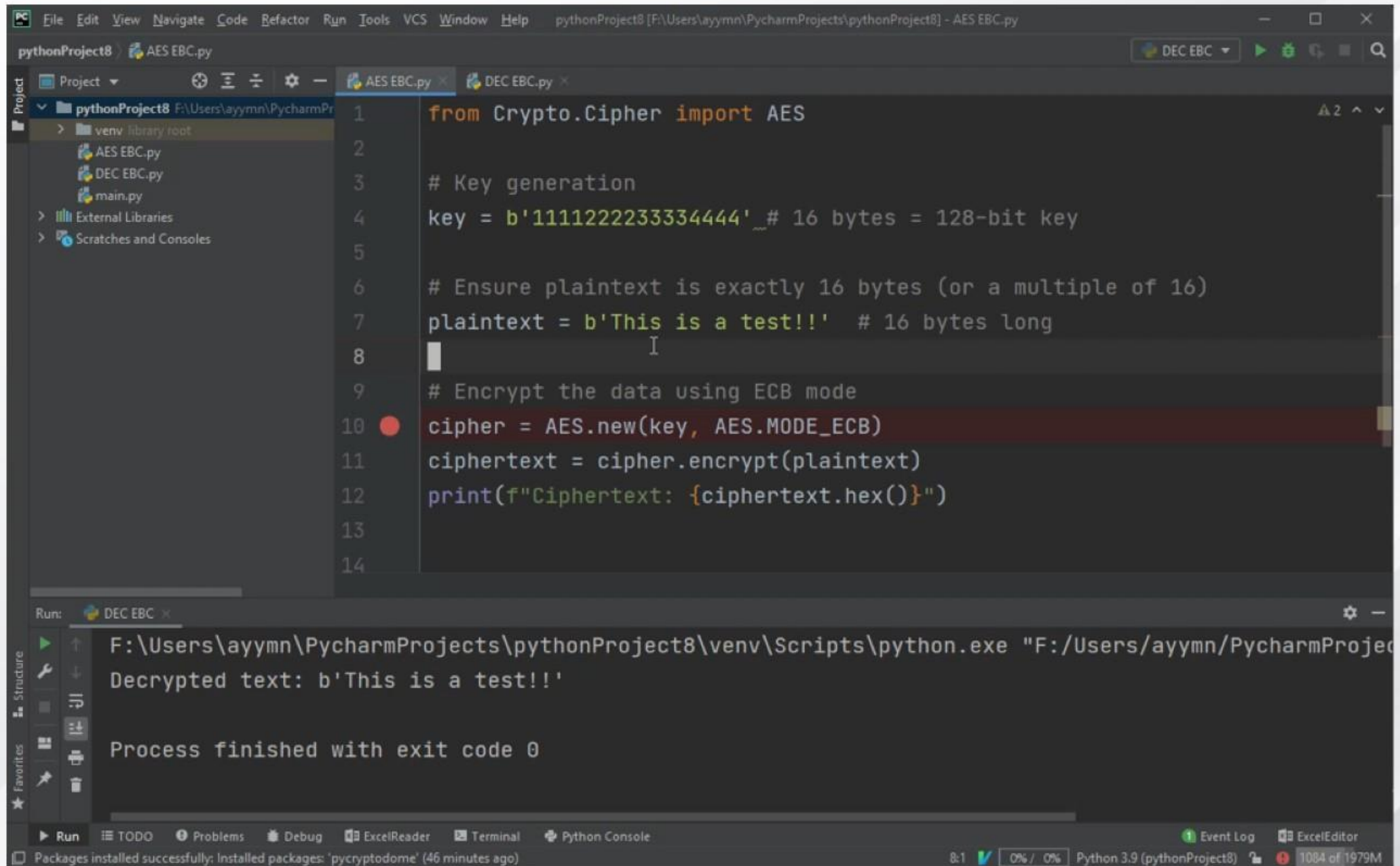
ciphertext_hex = "aec97d6c2150fa7baf97f37bfea3069c"
ciphertext = bytes.fromhex(ciphertext_hex)

# Decrypt the ciphertext
decipher = AES.new(key, AES.MODE_ECB)
decrypted_data = decipher.decrypt(ciphertext)

print(f"Decrypted text: {decrypted_data}")
```

Output:

```
Decrypted text: b'This is a test!!'
```



Cipher Block Chaining (CBC) Mode

- **How it Works:** Each block of plaintext is XOR-ed with the previous ciphertext block before encryption. The first block uses an initialization vector (IV) instead of a previous block.
- **Advantage:** Produces unique ciphertext even if plaintext blocks are identical, as each block depends on the previous one. Patterns are obscured effectively.
- **Disadvantage:** Cannot parallelize encryption easily. An error in one block can affect subsequent blocks during decryption.
- **IV Importance:** The IV ensures the same plaintext will produce different ciphertexts if encrypted again.

Cipher Block Chaining (CBC) Mode

- **Encryption:**

$$C_1 = E(K, [IV \oplus P_1])$$

$$C_2 = E(K, [C_1 \oplus P_2])$$

*

*

$$C_j = E(K, [C_{j-1} \oplus P_j])$$

where $E[K, X]$ is the encryption of plaintext X using key K ,
and $\oplus$ is the exclusive-OR operation

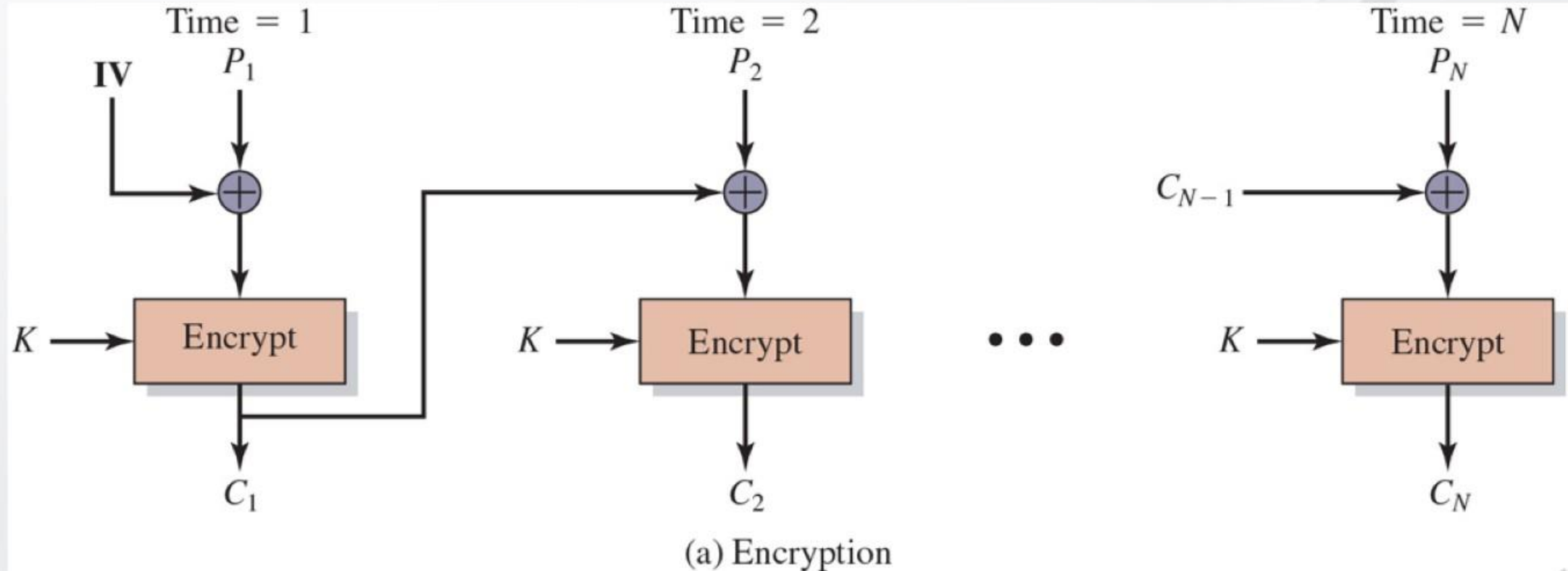
IV is initialization vector

C is cybertext

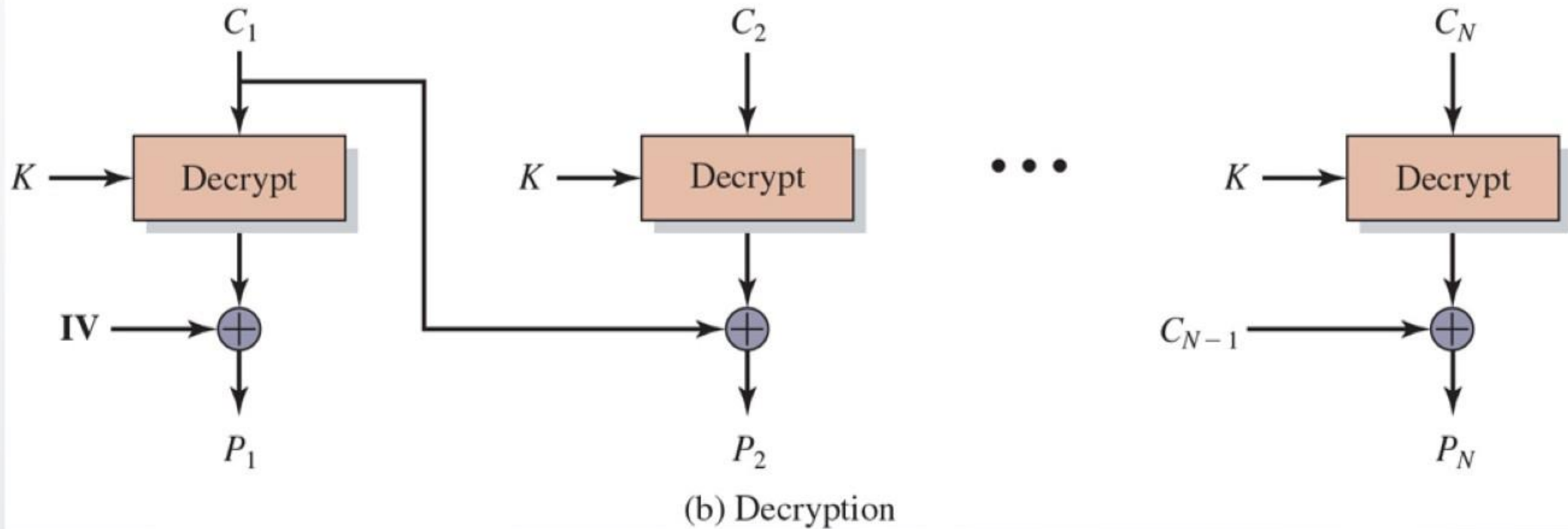
Cipher Block Chaining (CBC) Mode

- **Initialization Vector (IV):**
 - Used for the first plaintext block: $C_1 = E(K, IV \oplus P_1)$.
 - Protect the IV for maximum security (e.g., using ECB).
 - If IV is compromised, first plaintext block can be altered.
- **Key Points:**
 - The IV must be known to both sender and receiver.
 - IV tampering can lead to predictable bit changes in the decrypted plaintext.

Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) Mode



Python Code to Encrypt using AES (CBC Mode):

```
from Crypto.Cipher import AES # pip install pycryptodome
from Crypto.Random import get_random_bytes

# Key generation
key = b'1111222233334444' # 16 bytes = 128-bit key
# Ensure plaintext is exactly 16 bytes (or a multiple of 16)
plaintext = b'This is a test!!' # 16 bytes long
# Generate a random 16-byte IV (Initialization Vector)
iv = get_random_bytes(16)

# Encrypt the data using CBC mode
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(plaintext)

print(f"IV: {iv.hex()}")
print(f"Ciphertext: {ciphertext.hex()}")
```

Output:

```
IV: 8541eaaf99dd0e0445112409033943b0
Ciphertext: fb5a83dba322060bf18cf4b0e03bd234
```

Python Code to Decryption using AES (CBC Mode):

```
from Crypto.Cipher import AES

key = b'1111222233334444' # 16 bytes = 128-bit key
# Ciphertext and IV from the encryption process
ciphertext_hex = "fb5a83dba322060bf18cf4b0e03bd234"
ciphertext = bytes.fromhex(ciphertext_hex)

iv_hex = "8541eaaf99dd0e0445112409033943b0"
iv = bytes.fromhex(iv_hex)

# Decrypt the ciphertext using CBC mode
decipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_data = decipher.decrypt(ciphertext)

print(f"Decrypted text: {decrypted_data}")
```

Output:

```
Decrypted text: b'This is a test!!'
```

pythonProject8 AEC CBC E.py

Project pythonProject8 F:\Users\ayymn\PycharmProjects\pythonProject8

- venv library root
- AEC CBC D.py
- AEC CBC E.py
- AEC CFB Dec.py
- AEC CFB Enc.py
- AES EBC.py
- DEC EBC.py
- main.py
- External Libraries
- Scratches and Consoles

```
1 from Crypto.Random import get_random_bytes
2
3
4 # Key generation
5 key = b'1111222233334444' # 16 bytes = 128-bit key
6
7 # Ensure plaintext is exactly 16 bytes (or a multiple of 16)
8 plaintext = b'This is a test!!' # 16 bytes long
9
10 # Generate a random 16-byte IV (Initialization Vector)
11 iv = get_random_bytes(16)
12
13 # Encrypt the data using CBC mode
14 cipher = AES.new(key, AES.MODE_CBC, iv)
15 ciphertext = cipher.encrypt(plaintext)
16
17 print(f"IV: {iv.hex()}")
18 print(f"Ciphertext: {ciphertext.hex()}")
```

Terminal: Local +

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.
(venv) F:\Users\ayymn\PycharmProjects\pythonProject8>

TODO Problems Terminal ExcelReader Python Console

8:36 CRLF UTF-8 4 spaces 0% / 7% Python 3.9 (pythonProject8) 406 of 1979M

Cipher Feedback Mode (CFB)

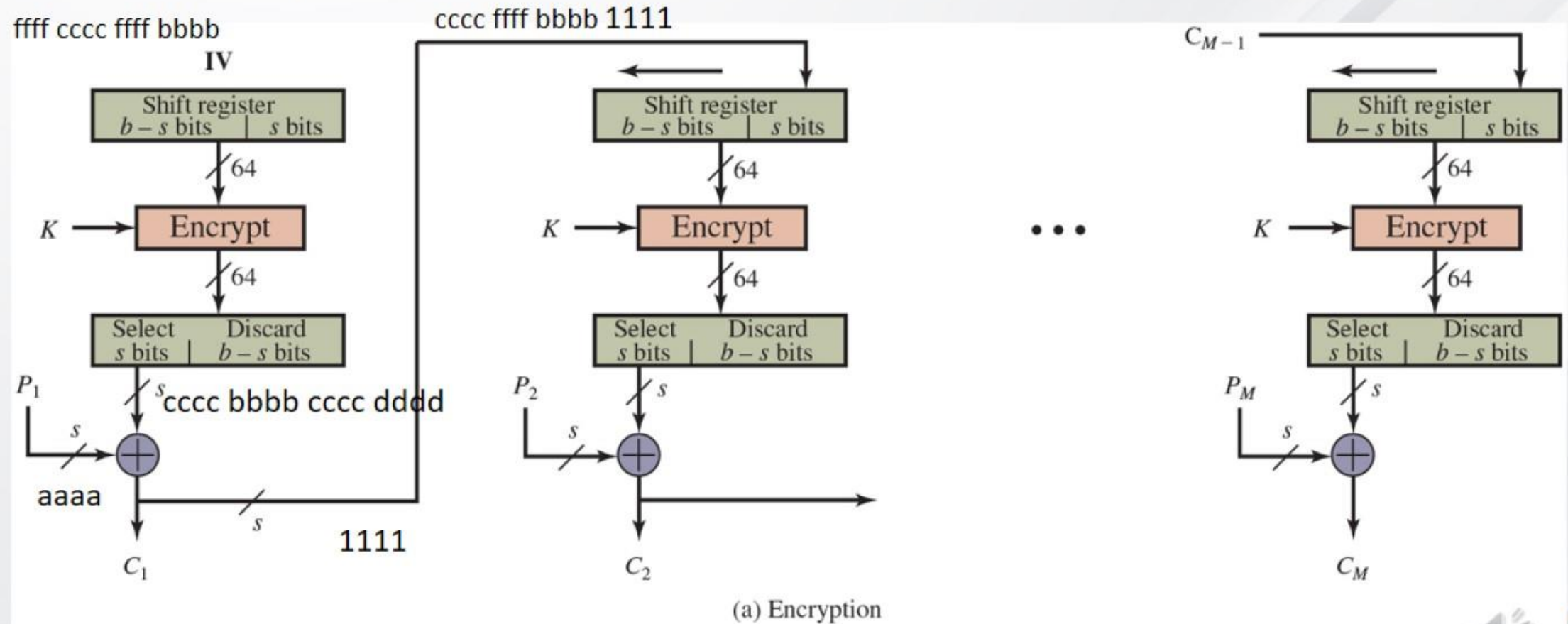
- Convert a block cipher to stream.
- Why stream cipher?
 - No need of padding
 - Real time
 - Length of plaintext = length of ciphertext
- General-purpose stream-oriented transmission
- Authentication

Cipher Feedback Mode (CFB) Encryption Process:

- **Shift Register Initialization:** The encryption process starts with an b -bit shift register initialized to an IV.
- **Encrypting the Register:** The entire content of the shift register is fed into the encryption function, producing an output block.
- **XOR Operation:** The leftmost s bits (where s is the number of bits processed at a time) of the encrypted output are XOR with the first unit of plaintext P_1 to produce the first unit of ciphertext C_1 .
- **Shifting the Register:** After generating C_1 , the contents of the shift register are shifted left by s bits. The newly generated ciphertext C_1 is then inserted into the least significant s bits of the register.
- **Continuation:** This process repeats for each subsequent unit of plaintext until all data is encrypted.

Figure 20.7

s-bit Cipher Feedback (CFB) Mode

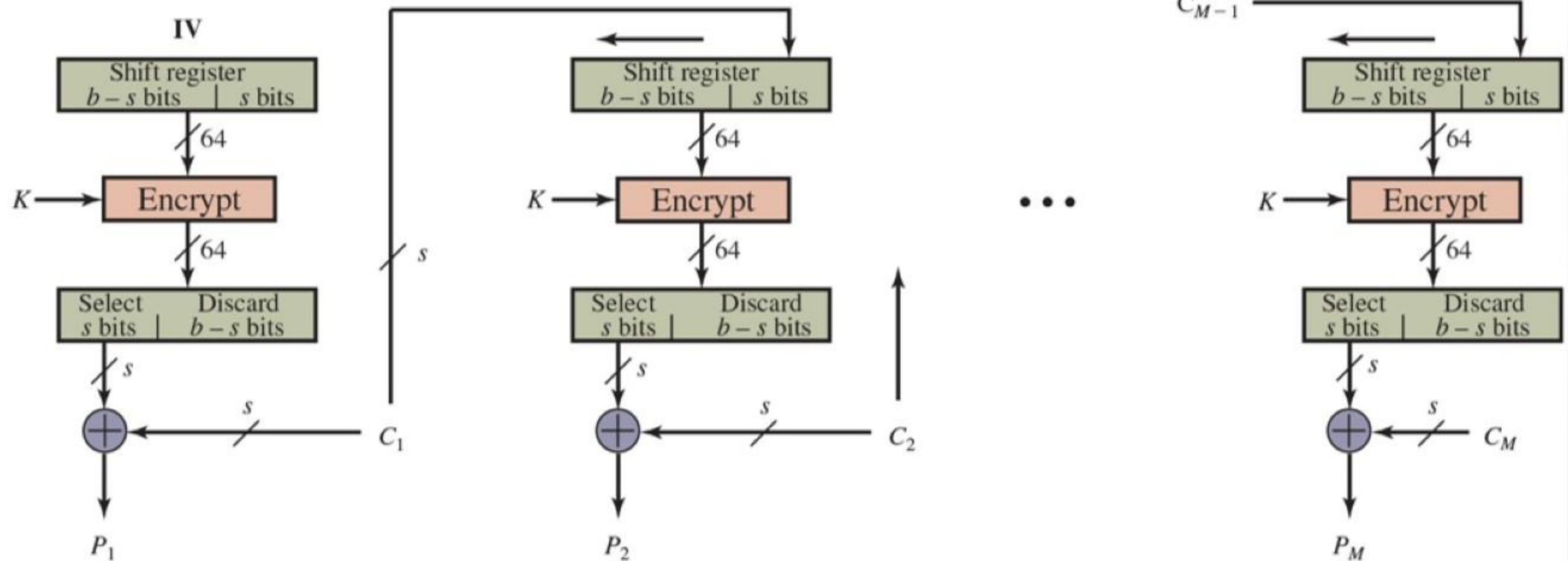


Cipher Feedback Mode (CFB) Decryption Process:

- **Shift Register Initialization:** The decryption process starts with a b -bit shift register, initialized to the same IV (Initialization Vector) used in encryption.
- **Encrypting the Register:** The entire content of the shift register is fed into the encryption function, producing an output block.
- **XOR Operation:** The leftmost s bits of the encrypted output are XOR with the first unit of ciphertext C_1 to recover the first unit of plaintext P_1 . This is because the encryption function's output is XOR with the ciphertext, reversing the encryption operation.
- **Shifting the Register:** After generating P_1 , the contents of the shift register are shifted left by s bits. The ciphertext unit C_1 is then inserted into the least significant s bits of the register.
- **Continuation:** This process repeats for each subsequent unit of ciphertext until all data is decrypted.

Figure 20.7

s-bit Cipher Feedback (CFB) Mode



(b) Decryption

- ## Security Considerations

- **Initialization Vector (IV):** Must be unique and unpredictable.
- **Error Propagation:** An error in ciphertext affects the corresponding plaintext and next unit.
- **Performance:** Can be slower than modes like Counter (CTR).

- ## Applications of CFB Mode

- Telecommunications
- Streaming services
- Secure communications

Cipher Feedback Mode (CFB)

- **Advantage:**
 - Can operate in real time
 - Need of padding is eliminated
 - Encryption function does decryption as well.
 - Length of plaintext = Length of cyphertext.
- **Disadvantage:**
 - Chances of wastage of transmission capacity.
 - Not a typical stream cipher.

Python Code to Encrypt using AES (CFB Mode):

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Key generation
key = b'1111222233334444' # 16 bytes = 128-bit key
# Plaintext (doesn't need to be a multiple of block size in CFB mode)
plaintext = b'This is a test!!'

# Generate a random 16-byte IV (Initialization Vector)
iv = get_random_bytes(16)

# Encrypt the data using CFB mode
cipher = AES.new(key, AES.MODE_CFB, iv)
ciphertext = cipher.encrypt(plaintext)

print(f"IV: {iv.hex()}")
print(f"Ciphertext: {ciphertext.hex()}")
```

Output:

```
IV: 3287a225906a0bba7d2b3f6b603f75c1
Ciphertext: d73eaf2255c418fca8c5451a6be0b5b6
```

Python Code to Decryption using AES (CFB Mode):

```
from Crypto.Cipher import AES

# Key generation
key = b'1111222233334444' # 16 bytes = 128-bit key

# Ciphertext and IV from the encryption process
ciphertext_hex = "d73eaf2255c418fca8c5451a6be0b5b6"
ciphertext = bytes.fromhex(ciphertext_hex)

iv_hex = "3287a225906a0bba7d2b3f6b603f75c1"
iv = bytes.fromhex(iv_hex)

# Decrypt the ciphertext using CFB mode
decipher = AES.new(key, AES.MODE_CFB, iv)
decrypted_data = decipher.decrypt(ciphertext)

print(f"Decrypted text: {decrypted_data.decode('utf-8')}")
```

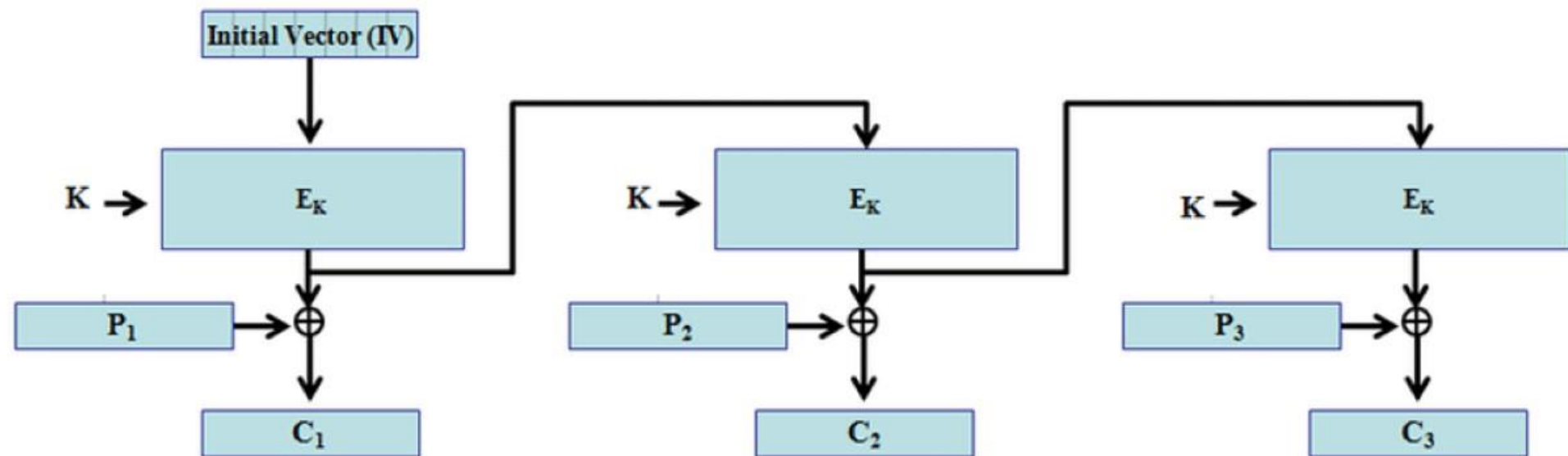
Output:

```
Decrypted text: b'This is a test!!'
```

Output Feedback (OFB)

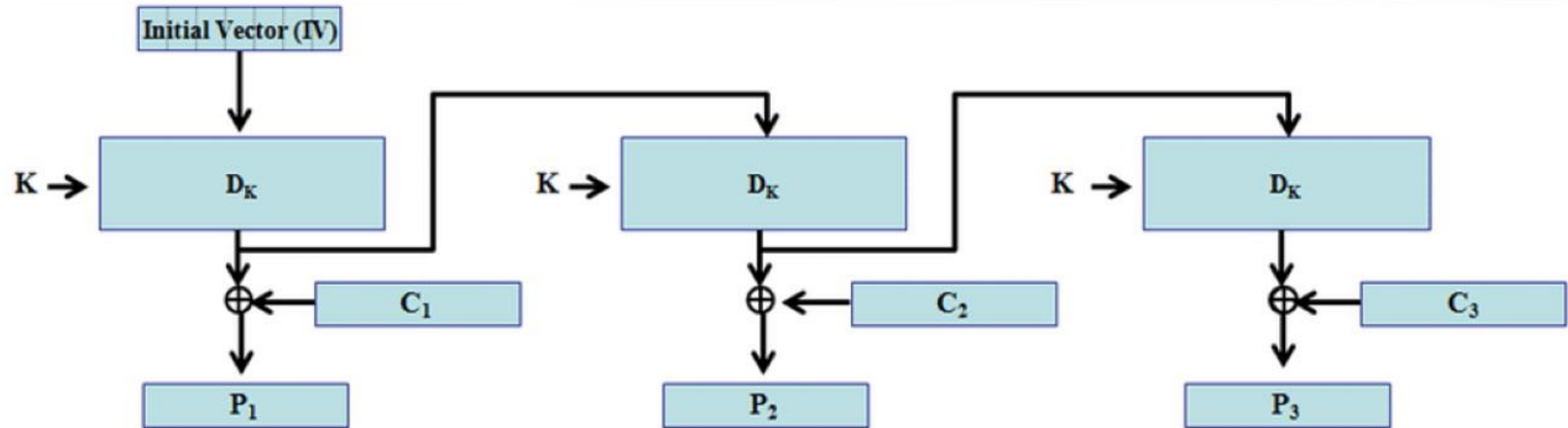
- **How it Works:**

- OFB generates keystream blocks by repeatedly encrypting an IV.
- The keystream is then XOR with the plaintext to produce ciphertext.



a) Output feedback (OFB) encryption mode

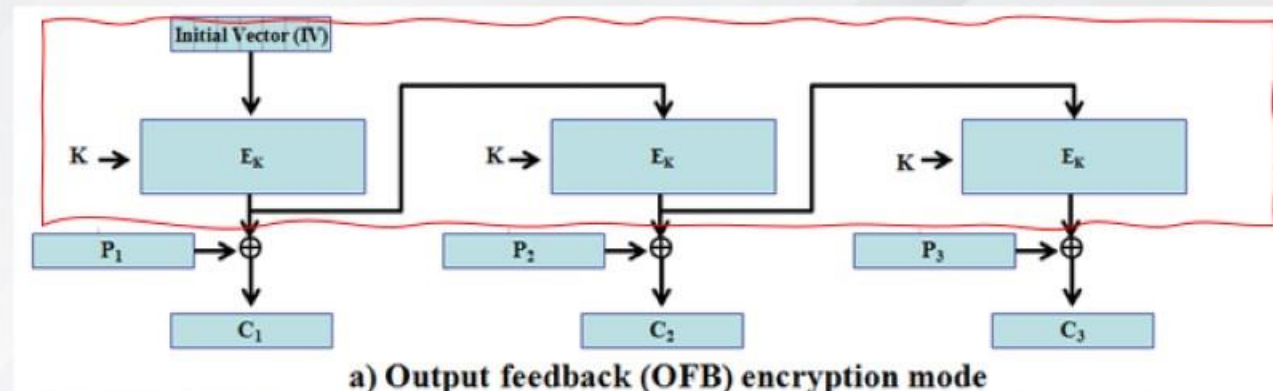
Output Feedback (OFB) Decryption



a) Output feedback (OFB) decryption mode

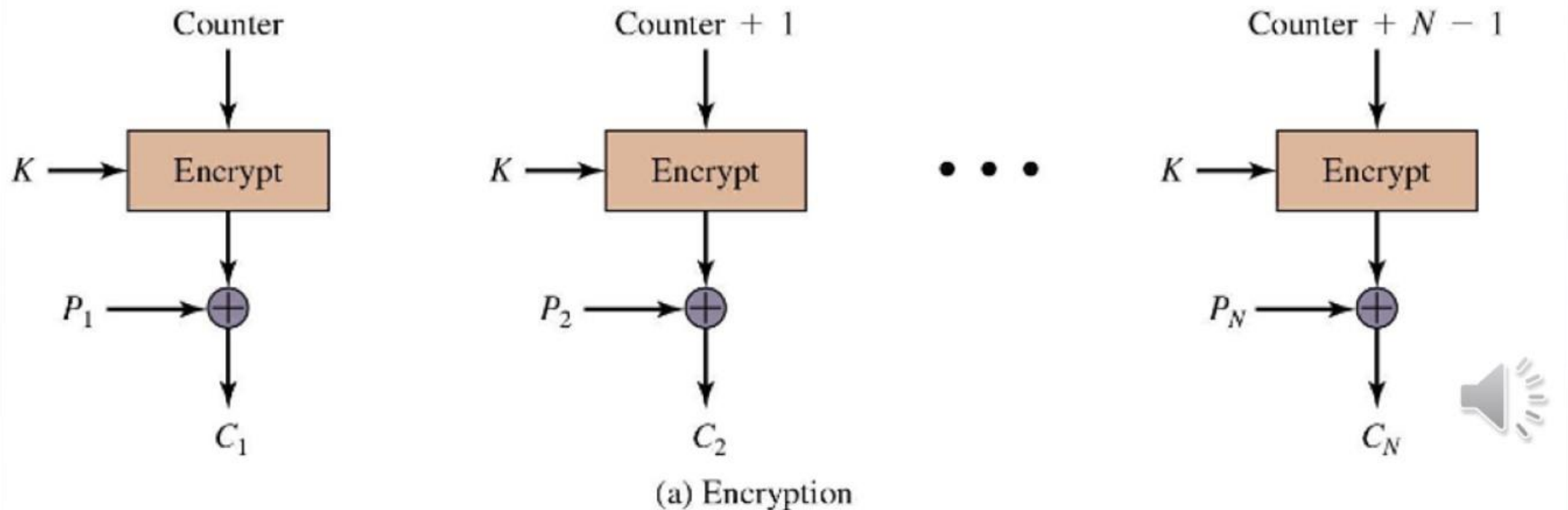
Output Feedback (OFB)

- **Advantage :**
 - Bit errors in transmission do not propagate
 - Same plaintext same key = different cyphertext
 - The plaintext length can be of random choice
- **Disadvantage :**
 - Sender and Receiver must be synced.
 - Move vulnerable to modification attack.
 - No parallelizable.
 - IV and Keys must be regenerated every time.

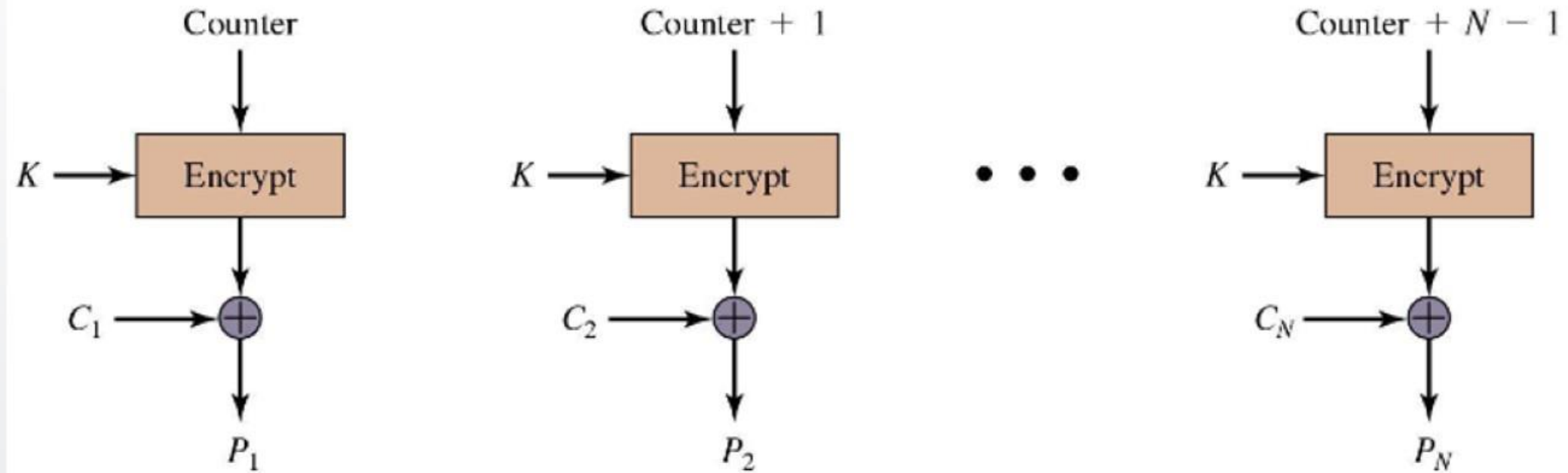


Counter (CTR)

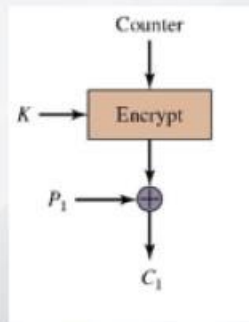
- **How it Works:**
 - CTR mode generates a unique keystream by encrypting a counter value.
 - The keystream is then XOR with the plaintext to produce ciphertext.



Counter (CTR) Mode



(b) Decryption



Advantages of CTR mode

- **Hardware Efficiency:** Parallel processing on multiple blocks, maximizing throughput based on parallelism, unlike chaining modes.
- **Software Efficiency:** Utilizes modern processor features (e.g., pipelining, SIMD) for parallel execution, improving performance.
- **Preprocessing:** Encryption can be preprocessed before plaintext/ciphertext is available, leaving only XOR operations, enhancing throughput.
- **Random Access:** Each block can be processed independently, allowing random access to specific blocks without needing to compute previous blocks.
- **Provable Security:** As secure as other encryption modes, ensuring strong protection.
- **Simplicity:** Requires only the encryption algorithm, simplifying implementation, especially for algorithms like AES.



Summary

Mode	How It Works	IV Required	Error Propagation	Parallelizable	Main Use Case	Pros	Cons
ECB	Encrypts each block independently	No	No	Yes (Encryption)	Simple data	Fast, simple	Insecure, reveals data patterns
CBC	XORs each block with the previous ciphertext	Yes	Affects current and next block	No	File encryption	Secure, hides data patterns	Slower, not parallelizable
CFB	Turns block cipher into a stream cipher	Yes	Affects a few bytes	No	Secure communications	No padding needed	Some error propagation
OFB	Generates keystream from IV	Yes	No	No	Noisy channels	No error propagation, no padding	Sensitive to IV reuse
CTR	Uses a counter with each block	Yes (Counter)	No	Yes	High-performance encryption	Fast, fully parallelizable	Needs unique counter per block



Thank You

