ENCS5121
Information Security and
Computer Networks Laboratory

EXPERIMENT #2
Secret-Key Encryption

Slides By: Tariq Odeh

جامعة بيرزيت BIRZEIT UNIVERSITY

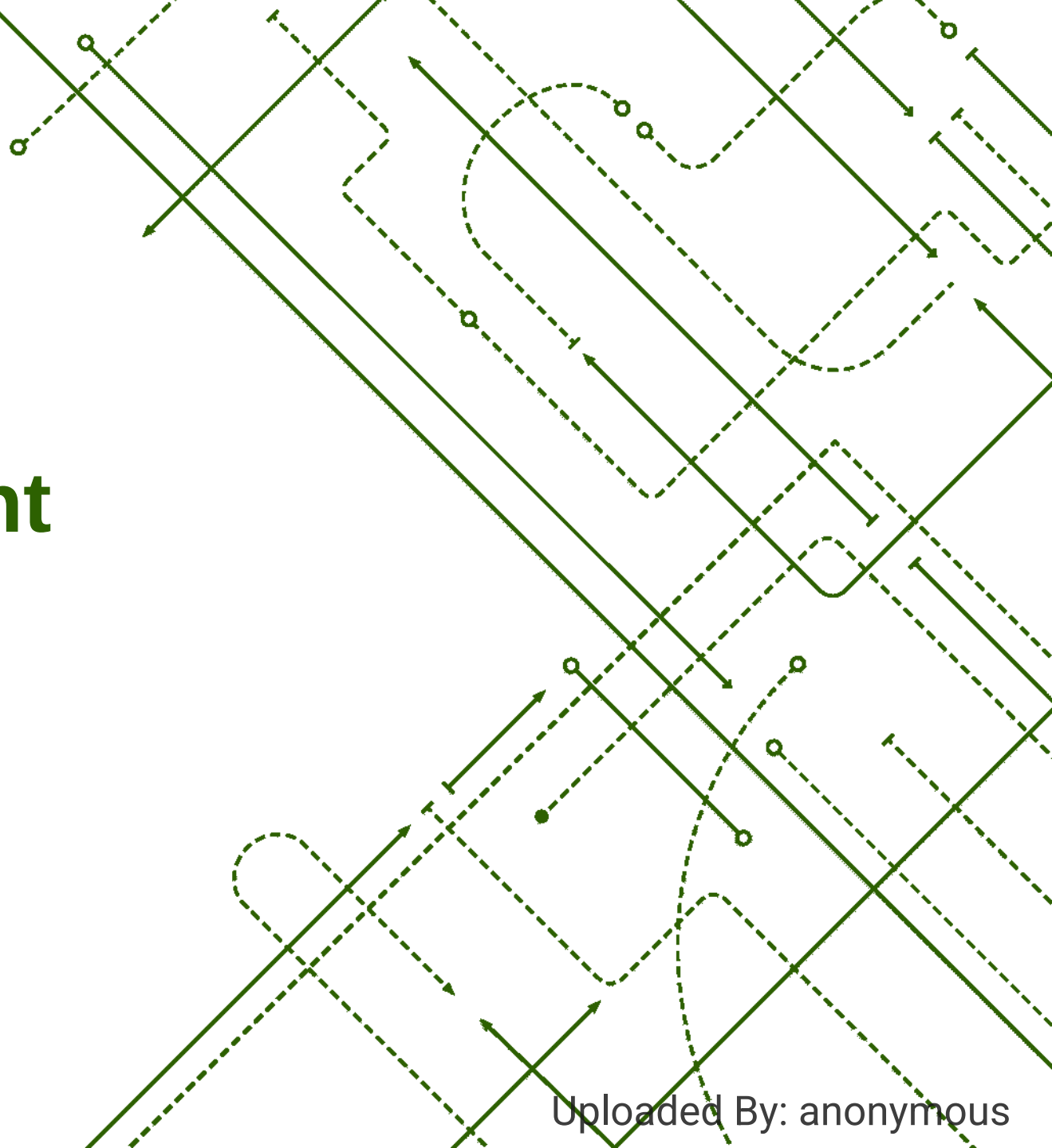
Overview

- **Learning Objective:** Familiarize with secret-key encryption, encryption modes, and common attacks.
- **Key Topics:**
  - Secret-key encryption basics
  - Substitution cipher & frequency analysis
  - Encryption modes (IV & paddings)
  - Common encryption mistakes & vulnerabilities
- **Hands-on Experience:** Encrypting/decrypting messages, identifying mistakes in encryption, and exploiting vulnerabilities.
- **Tools & Programming:** Use of crypto libraries for practical implementation.

Outline

- **Lab Environment**
- **Task 1: Frequency Analysis**
  - Mono-Alphabetic substitution cipher
  - Frequency Analysis.
- **Task 2**: **Encryption using Different Ciphers and Modes.**
- **Task 3: Encryptions Mode– ECB vs. CBC.**
  - Electronic Code Book (ECB)
  - Cipher Block Chaining (CBC)
- **Task 4:** Padding.
- **Task 5:** Error Propagation– Corrupted Cipher Text.
- **Task 6:** Initial Vector (IV) and Common Mistakes.
- **Task 7:** Programming using the Crypto Library.

Lab Environment



Container Setup and Commands

Setup Instructions:

- Download Labsetup.zip to your VM.
  - (https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Encryption/)
- Unzip and enter the Labsetup folder.
- Use docker-compose.yml to set up the lab environment

```
$ docker-compose build  # Build the container images
$ docker-compose up     # Start the containers
$ docker-compose down   # Shut down the containers

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

Container Setup and Commands (Cont.)

```
$ dockps         // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>    // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

Task 1: Frequency Analysis

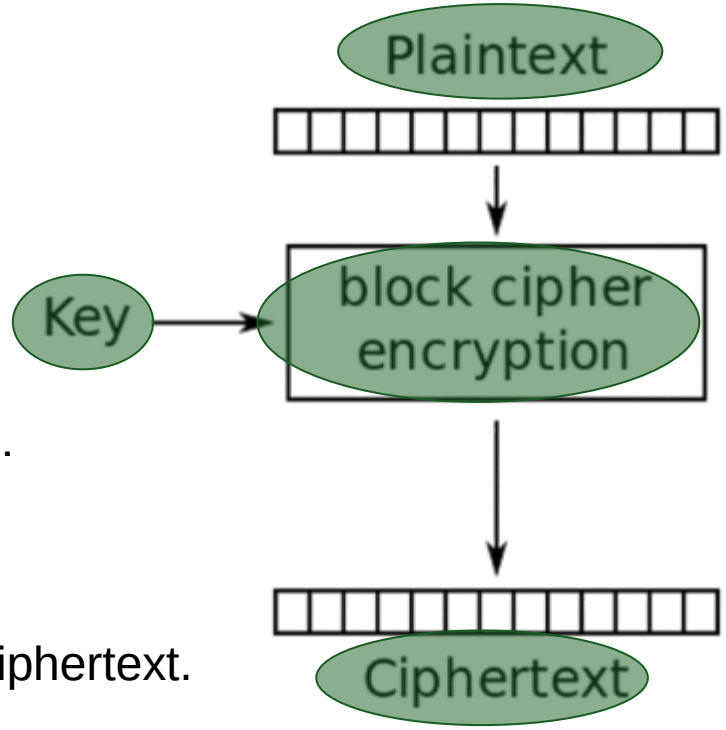
Mono-Alphabetic substitution cipher

Plaintext:

- The original, readable message that needs to be encrypted.
- Example: "HELLO WORLD"

Key:

- Length: The key should be exactly 26 characters long.
- Uniqueness: Each letter must be used only once in the key.
- Example: If the key is "QWERTYUIOPASDFGHJKLZXCVBNM“.



Encryption Operation:

- Each letter in the plaintext is **replaced** according to the key.
- The same letter in plaintext always maps to the same letter in ciphertext.

Ciphertext:

- The output of the encryption operation.
- the encrypted message that is not readable without the key.
- Example: "QEBBX"

Mono-Alphabetic substitution cipher (Cont.)

Example:

Plaintext: “HELLO”

Encryption

Alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Key: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

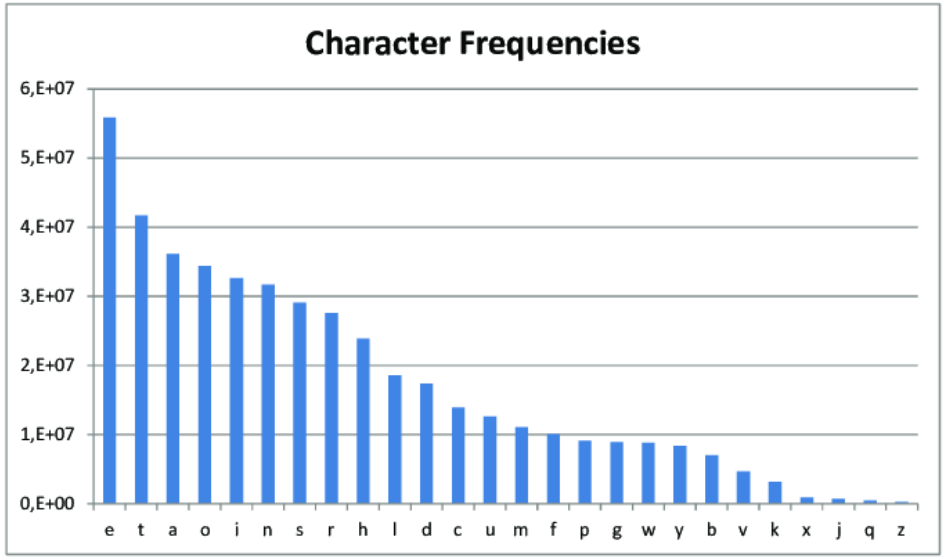
Decryption

Ciphertext: “ITSSG”

Frequency Analysis

- Frequency analysis uses letter frequencies to break ciphers.

- **How it works:**
  - Count letter occurrences in ciphertext.
  - Compare to known language frequencies.
  - Identify patterns.

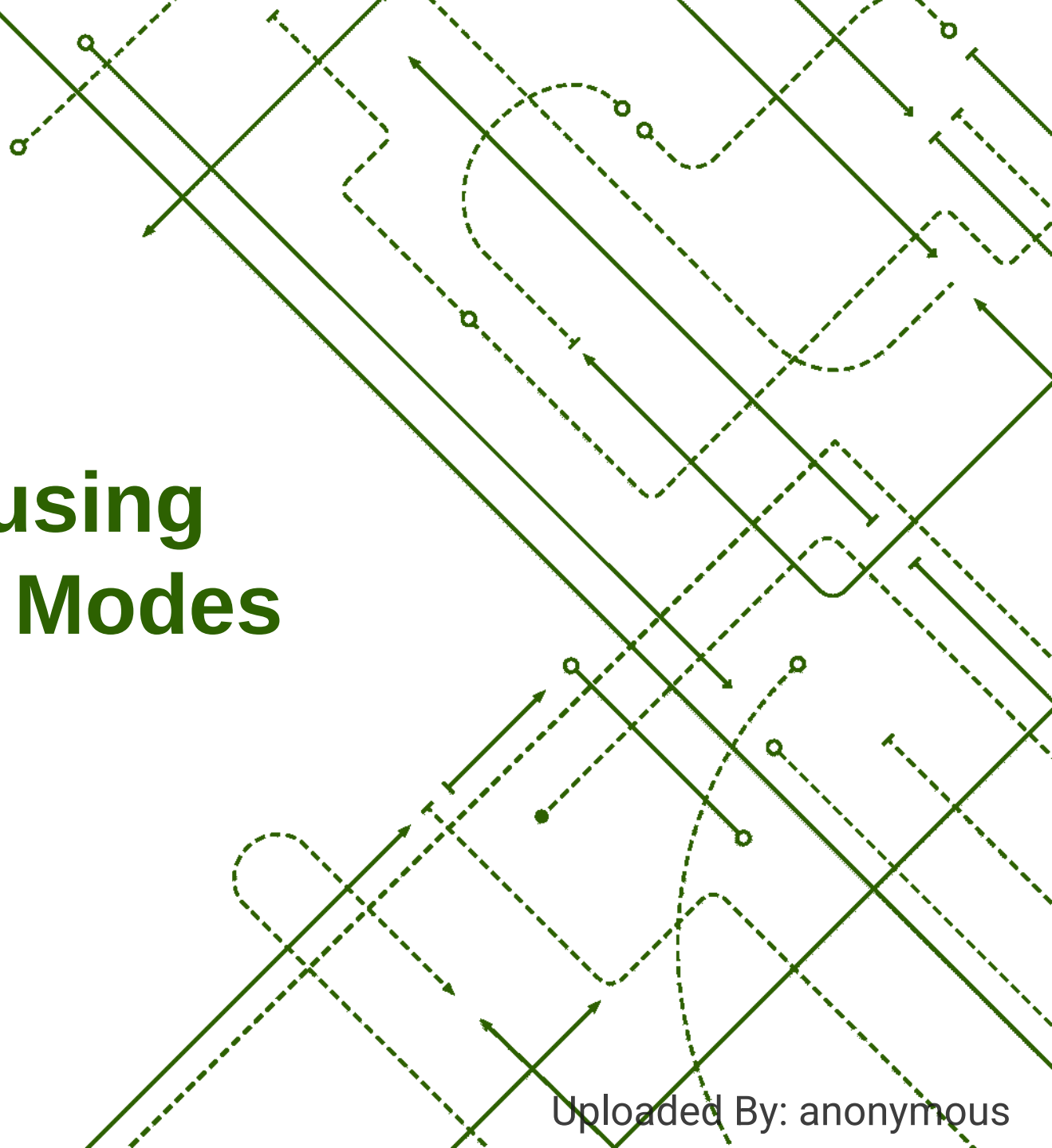



```
$ ./freq.py
--------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
...
--------------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
...
--------------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
...
```

Steps:

#	Command
1.	python3 key.py
2.	tr [:upper:] [:lower:] <story.txt> PlainLowerCase.txt
3.	tr -cd '[a-z][\n][:space]' <PlainLowerCase.txt> plaintext.txt
4.	tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' < plaintext.txt > ciphertext.txt
5.	python3 freq.py
6.	Make replaces based on frequency results

Task 2: Encryption using Different Ciphers and Modes

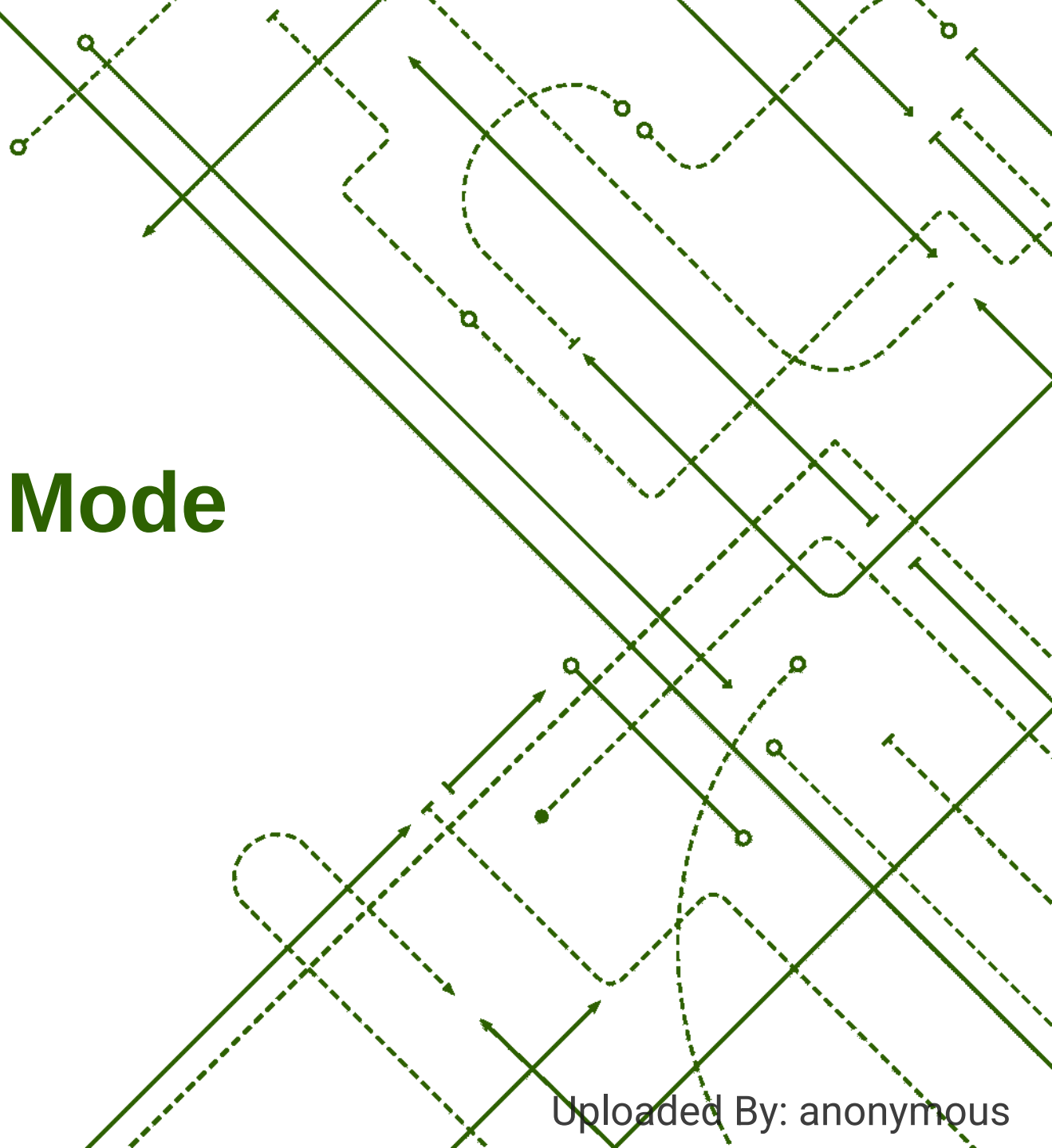


Encryption using Different Ciphers and Modes

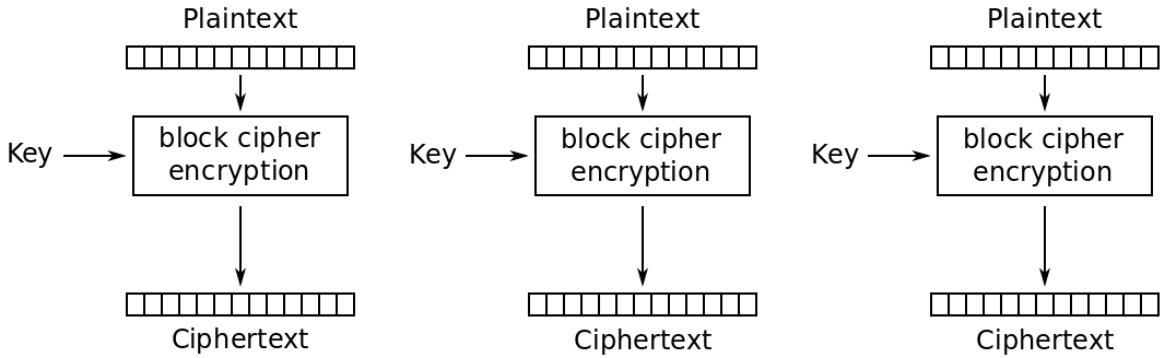
- **Various encryption algorithms and modes.**
  - **AES-128-CBC:** Advanced Encryption Standard with a 128-bit key in Cipher Block Chaining mode.
  - **AES-128-CFB:** Advanced Encryption Standard with a 128-bit key in Cipher Feedback mode.
  - **BF-CBC:** Blowfish in Cipher Block Chaining mode.

AES-128-CBC	Encryption	openssl enc -aes-128-cbc -in plain.txt -out cipher_aes128cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
	Decryption	openssl enc -d -aes-128-cbc -in cipher_aes128cbc.bin -out decrypted_aes128cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
AES-128-CFB	Encryption	openssl enc -aes-128-cfb -in plain.txt -out cipher_aes128cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
	Decryption	openssl enc -d -aes-128-cfb -in cipher_aes128cfb.bin -out decrypted_aes128cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
BF-CBC	Encryption	openssl enc -bf-cbc -in plain.txt -out cipher_bfcbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
	Decryption	openssl enc -d -bf-cbc -in cipher_bfcbc.bin -out decrypted_bfcbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

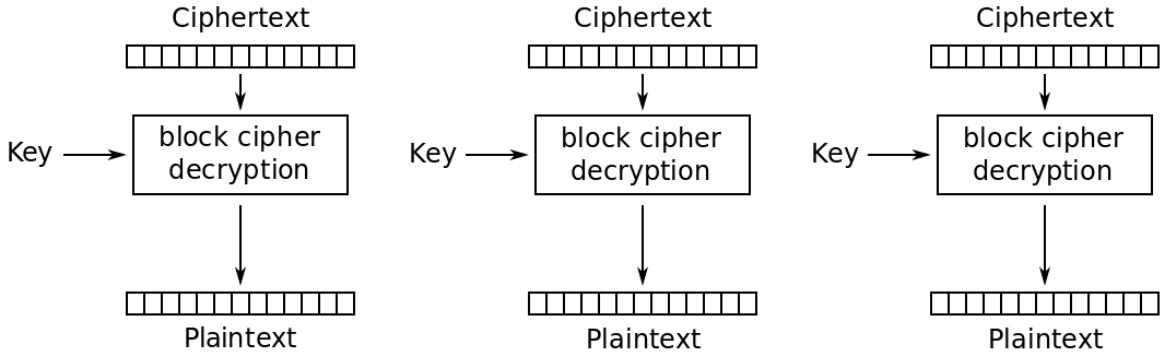
Task 3: Encryptions Mode ECB vs. CBC



Electronic Code Book (ECB)

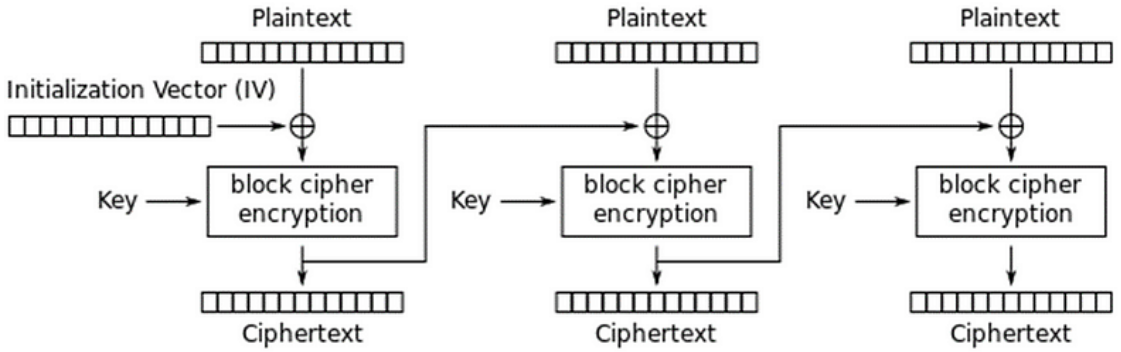


Electronic Codebook (ECB) mode encryption

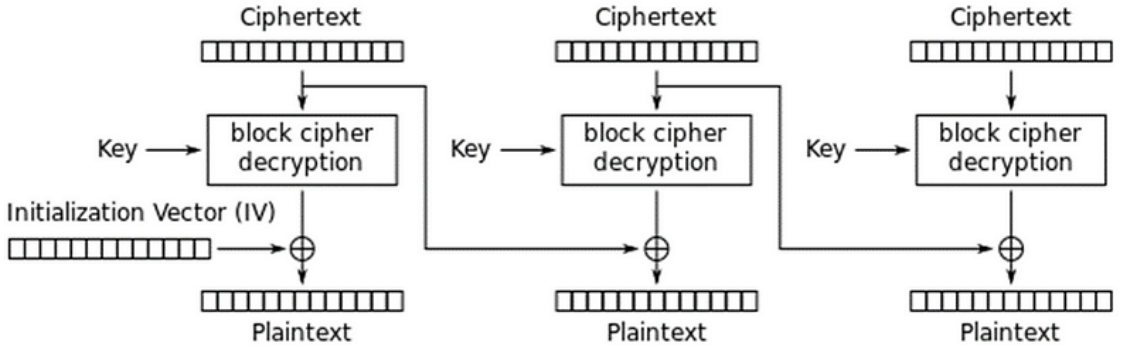


Electronic Codebook (ECB) mode decryption

Cipher Block Chaining (CBC)



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

ECB vs. CBC

Original image

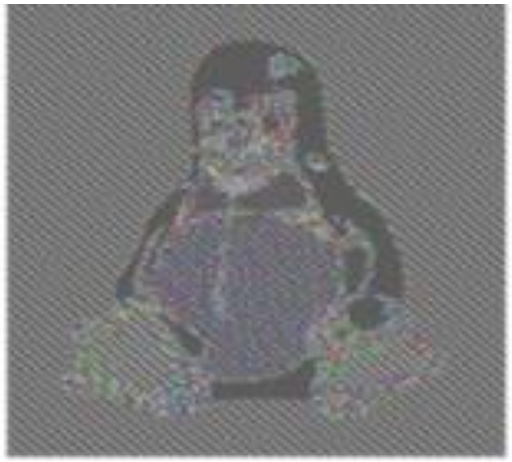


Image with AES using ECB

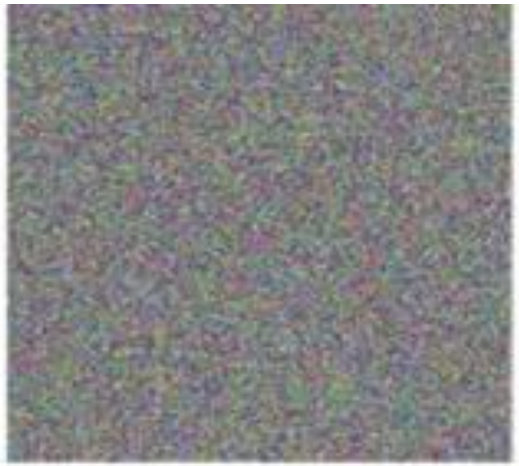
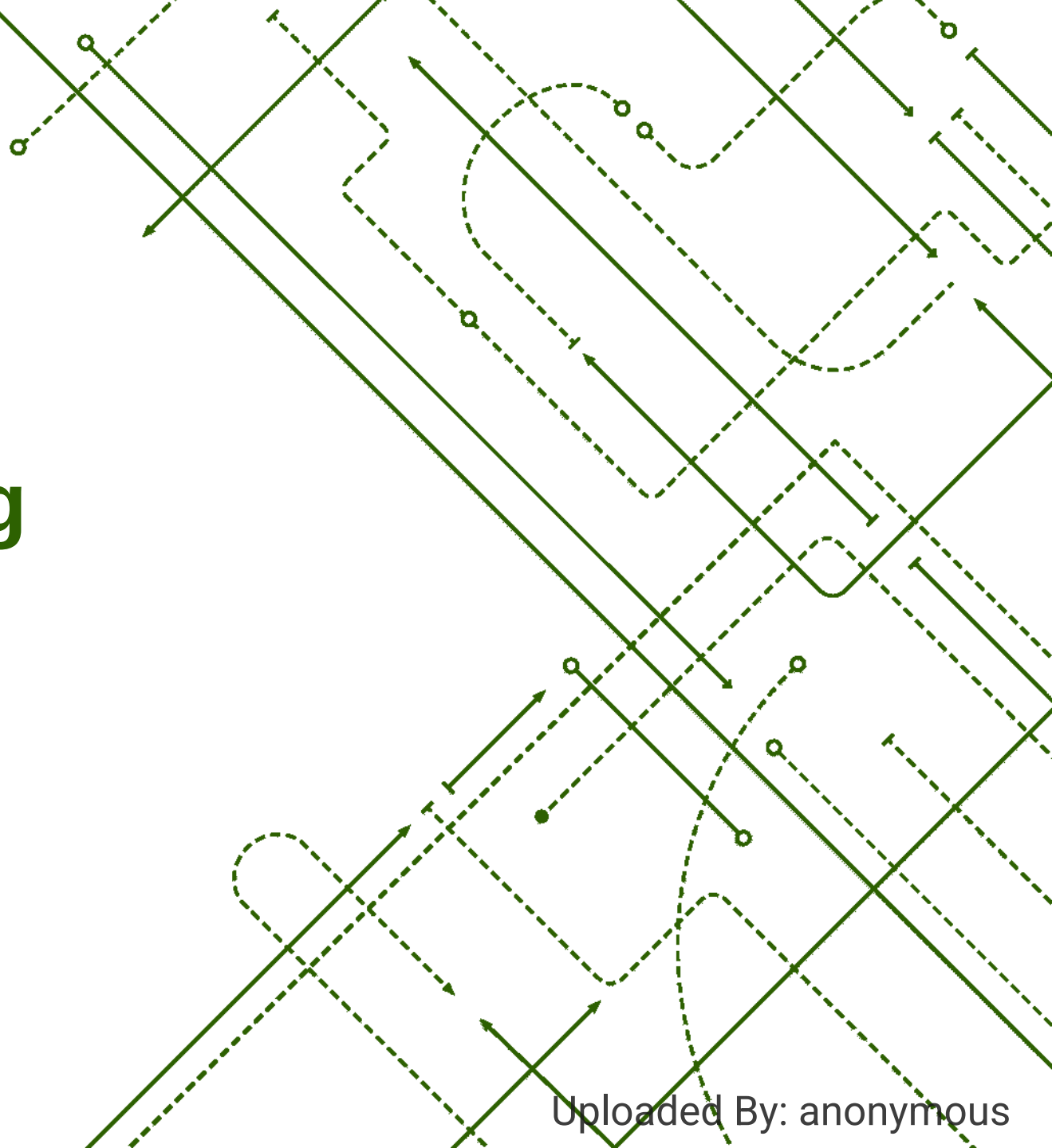


Image with AES using CBC

Task 3: Commands

Explanation	Command
Encrypts the image using AES-128 in **ECB** mode. The result is saved as ECB1.bmp.	openssl enc -aes-128-ecb -e -in original.bmp -out ECB1.bmp -K 00112233445566778889aabbccddeeff
Encrypts the image using AES-128 in **CBC** mode with an IV (Initialization Vector). The result is saved as CBC1.bmp.	openssl enc -aes-128-cbc -e -in original.bmp -out CBC1.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
Saves the first 54 bytes (the image header) from original.bmp to a file called header.	head -c 54 original.bmp > header
Saves the encrypted part (starting from byte 55) to a file called body.	tail -c +55 ECB1.bmp > body tail -c +55 CBC1.bmp > body
Combines the header and body to make a new image file.	cat header body > ECB_encrypted.bmp cat header body > CBC_encrypted.bmp
Opens the encrypted image for viewing.	eog ECB_encrypted.bmp eog CBC_encrypted.bmp

Task 4: Padding



Identify which modes need padding

- **ECB and CBC:**
  - Modes will have padding because they work in block units (AES block size is 16 bytes).
- **CFB and OFB:**
  - Modes do not need padding because they operate on smaller units (stream-like behaviour).

Experiment with different file sizes

Explanation	Command
Create files with 5, 10, and 16 bytes	echo -n "12345" > f1.txt # 5 bytes echo -n "1234567890" > f2.txt # 10 bytes echo -n "1234567890123456" > f3.txt # 16 bytes
Encrypt the files using AES-128-CBC	openssl enc -aes-128-cbc -e -in f1.txt -out f1_enc.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10 openssl enc -aes-128-cbc -e -in f2.txt -out f2_enc.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10 openssl enc -aes-128-cbc -e -in f3.txt -out f3_enc.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10

Experiment with different file sizes (Cont.)

Explanation	Command
View the padding data by decrypting with -nopad	openssl enc -aes-128-cbc -d -in f1_enc.txt -out f1_dec.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10 -nopad openssl enc -aes-128-cbc -d -in f2_enc.txt -out f2_dec.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10 -nopad openssl enc -aes-128-cbc -d -in f3_enc.txt -out f3_dec.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f10 -nopad
Use hexdump to inspect the padding	hexdump -C f1_dec.txt hexdump -C f2_dec.txt hexdump -C f3_dec.txt

Experiment with different file sizes (Cont.)

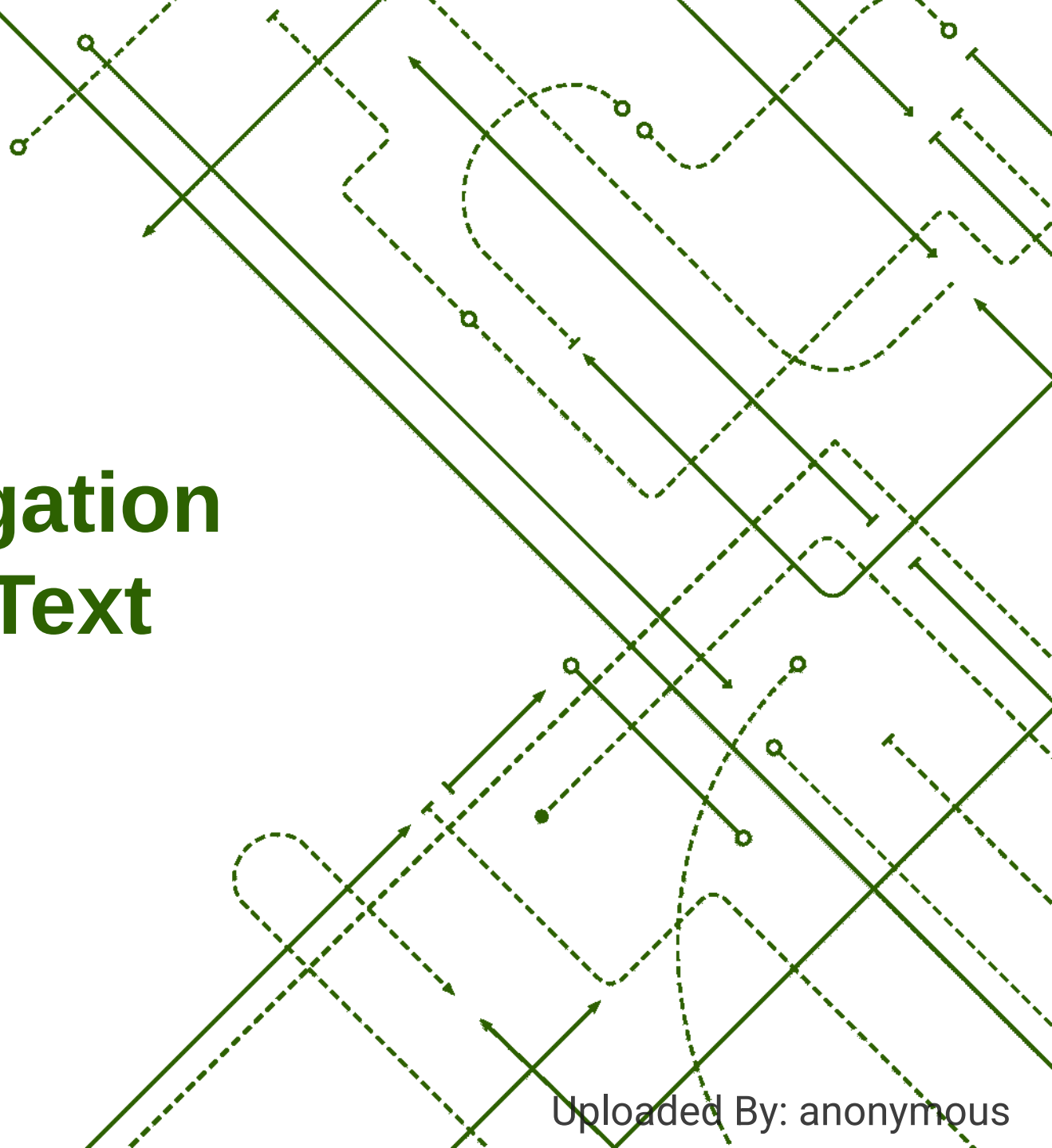
- **Expected Padding for the Files:**
  - File 1 (5 bytes): 11 bytes of padding, all with value 0x0b.
  - File 2 (10 bytes): 6 bytes of padding, all with value 0x06.
  - File 3 (16 bytes): No padding, since the file is already a multiple of the block size.

- **Padding Value:** Each of the 11 bytes has a value of 0x0b (which is 11 in decimal), following the same PKCS#7 padding convention.

```
[09/28/24]seed@VM:~/Desktop$ cat f1_dec.txt
12345
```

```
    [09/28/24]seed@VM:~/Desktop$ hexdump -C f1_dec.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[09/28/24]seed@VM:~/Desktop$
```

Task 5: Error Propagation Corrupted Cipher Text

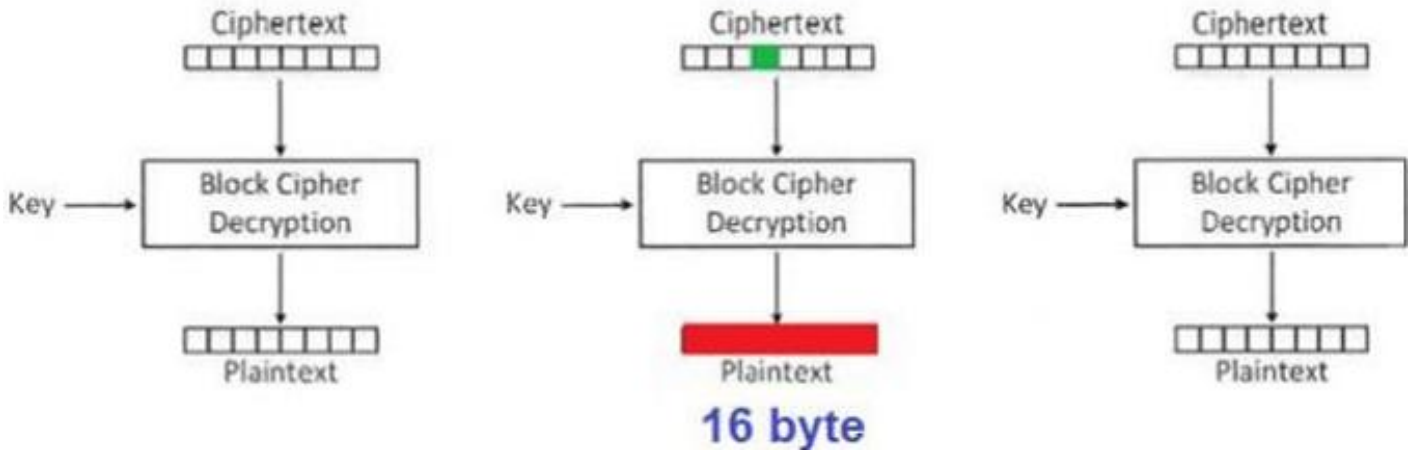


Steps:

Explanation	Command
Create a text file that is at least 1000 bytes long	echo -n "This is a sample text file for testing encryption and error propagation. " > testfile.txt for i in {1..20}; do cat testfile.txt >> testfile2.txt; done
Encrypt the file using the AES-128 cipher	KEY="00112233445566778889aabbccddeeff" IV="0102030405060708090a0b0c0d0e0f10" openssl enc -aes-128-cbc -e -in testfile.txt -out encrypted_file.bin -K $KEY -iv $IV
Create a Hex Dump of the Encrypted File	xxd encrypted_file.bin > encrypted_file.hex
Open the Hex Dump in a Text Editor	nano encrypted_file.hex
Corrupt a Byte	press **Ctrl + O** to save and **Ctrl + X** to exit
Convert the Hex Dump Back to Binary	xxd -r encrypted_file.hex > corrupted_encrypted_file.bin
Decrypt the Corrupted Ciphertext	openssl enc -aes-128-cbc -d -in corrupted_encrypted_file.bin -out decrypted_file.txt -K $KEY -iv $IV
Review the Decrypted Output	cat decrypted_file.txt

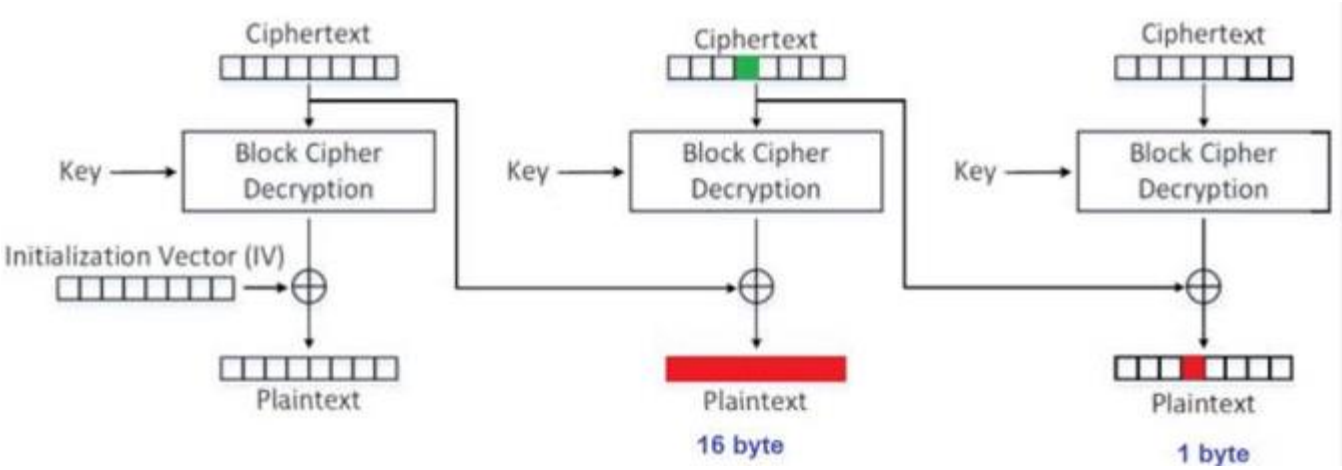
AES Modes: Impact of Bit Corruption

- In **ECB** mode, a ciphertext error affects only one block during decryption, with each block processed independently. Every byte in the decrypted block (16 bytes) is influenced by the corrupted byte, but error propagation does not extend beyond the affected block, which spans from byte 1 to byte 16.



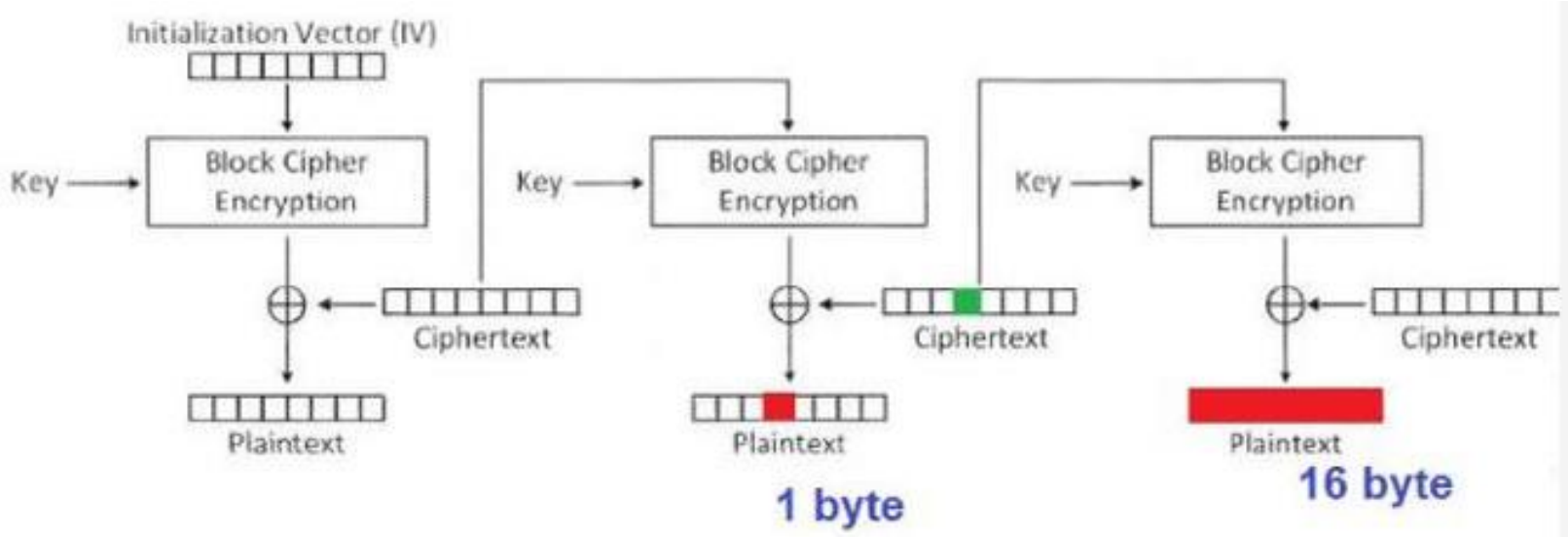
AES Modes: Impact of Bit Corruption (Cont.)

- In **CBC** mode, a single bit error in a ciphertext block affects the corresponding byte in the next plaintext block due to the XOR operation. This means that while the current plaintext block is altered, the first byte in the affected block will also be impacted by the previous ciphertext block, leading to significant changes in the resulting plaintext.



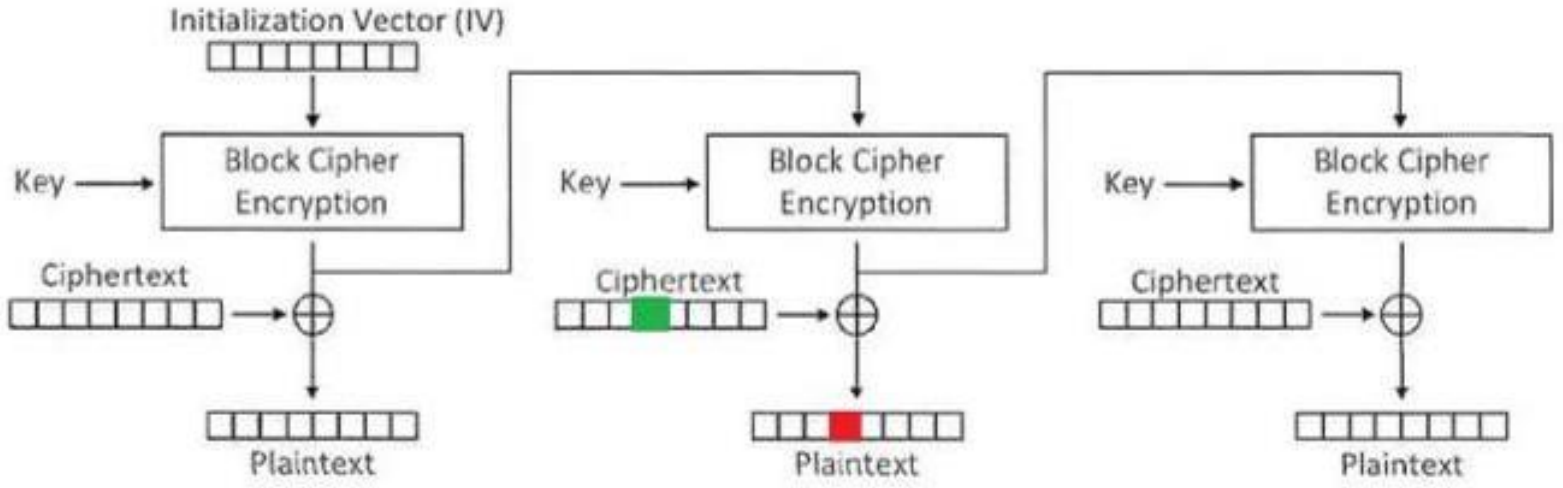
AES Modes: Impact of Bit Corruption (Cont.)

- In **CFB** mode, a single byte error in the ciphertext can significantly affect the subsequent plaintext. The error alters the corresponding bit in the current plaintext, leading to substantial changes in the decrypted output. This demonstrates a notable error propagation effect in CFB mode.



AES Modes: Impact of Bit Corruption (Cont.)

- In **OFB** mode, error propagation is minimal because each ciphertext block is simply XORed with the corresponding plaintext block during decryption. This means that only one byte will be affected by an error, without influencing any previous or subsequent blocks. Thus, OFB mode is the least impacted by error propagation.



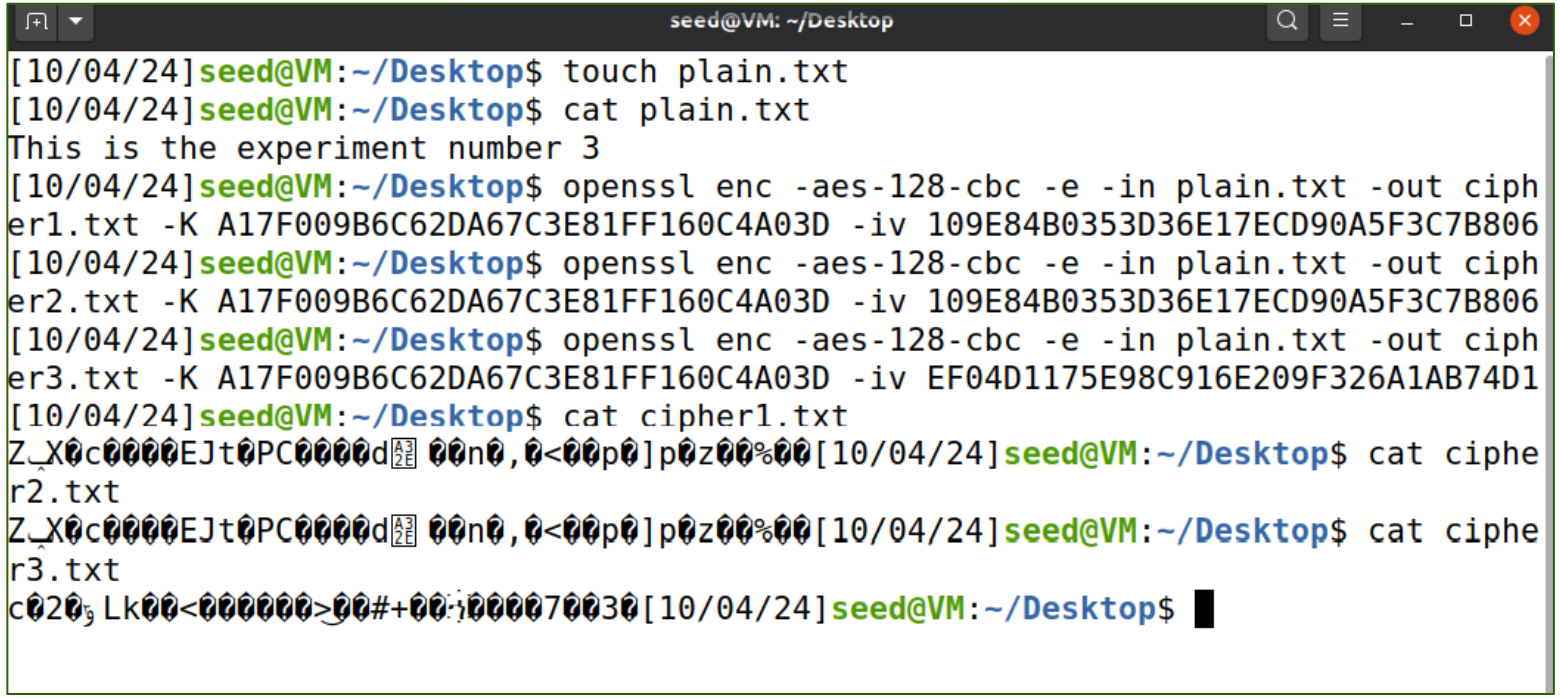
Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1 IV Experiment

- **Experiment Steps:**
  - Encrypt the same plaintext using:
    a) The same IV twice (under the same key).
    b) Two different IVs (under the same key).

Command
openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt -K A17F009B6C62DA67C3E81FF160C4A03D -iv 109E84B0353D36E17ECD90A5F3C7B806
openssl enc -aes-128-cbc -e -in plain.txt -out cipher2.txt -K A17F009B6C62DA67C3E81FF160C4A03D -iv 109E84B0353D36E17ECD90A5F3C7B806
openssl enc -aes-128-cbc -e -in plain.txt -out cipher3.txt -K A17F009B6C62DA67C3E81FF160C4A03D -iv EF04D1175E98C916E209F326A1AB74D1
cat cipher1.txt cat cipher2.txt cat cipher3.txt

Task 6.1 IV Experiment (Cont.)



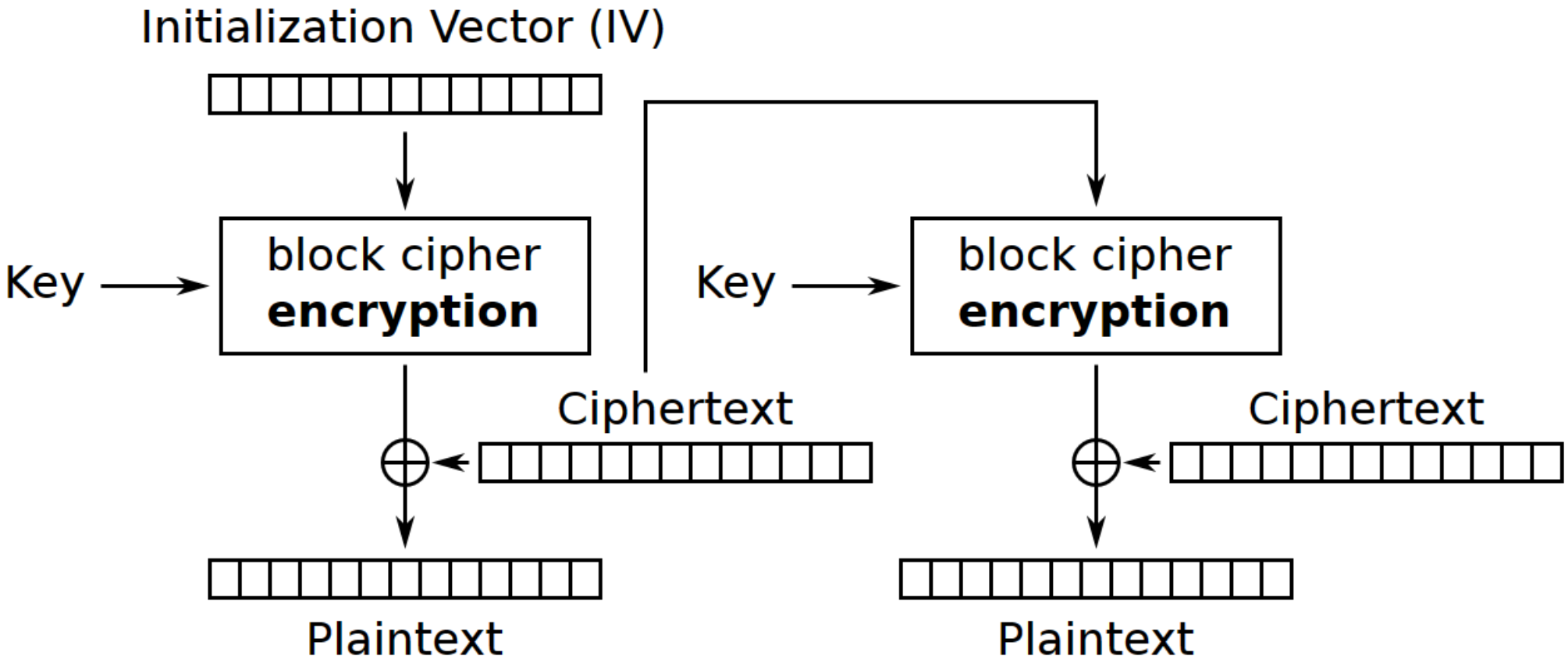
- **Observation:**
  - With two different IVs: The ciphertexts will be completely different, even though the plaintext and key are the same.
  - With the same IV twice: The ciphertexts will be identical, exposing a potential vulnerability.

Task 6.2 Common Mistake: Use the Same IV

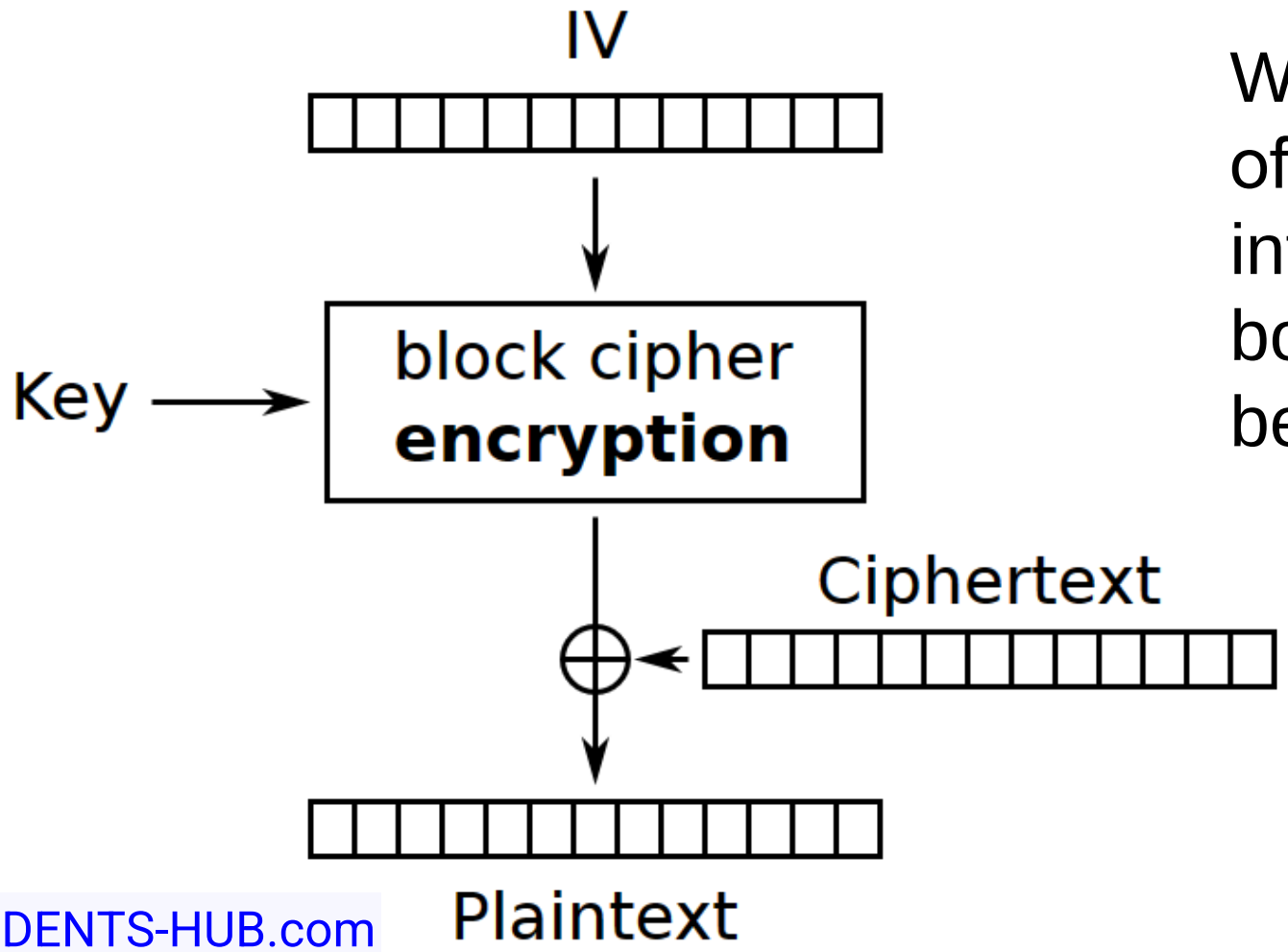
```
Plaintext  (P1): This is a known message!
Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

Plaintext  (P2): (unknown to you)
Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
```

- Vulnerable to ***known-plaintext attack.***

Cipher Block Chaining (CBC) mode decryption

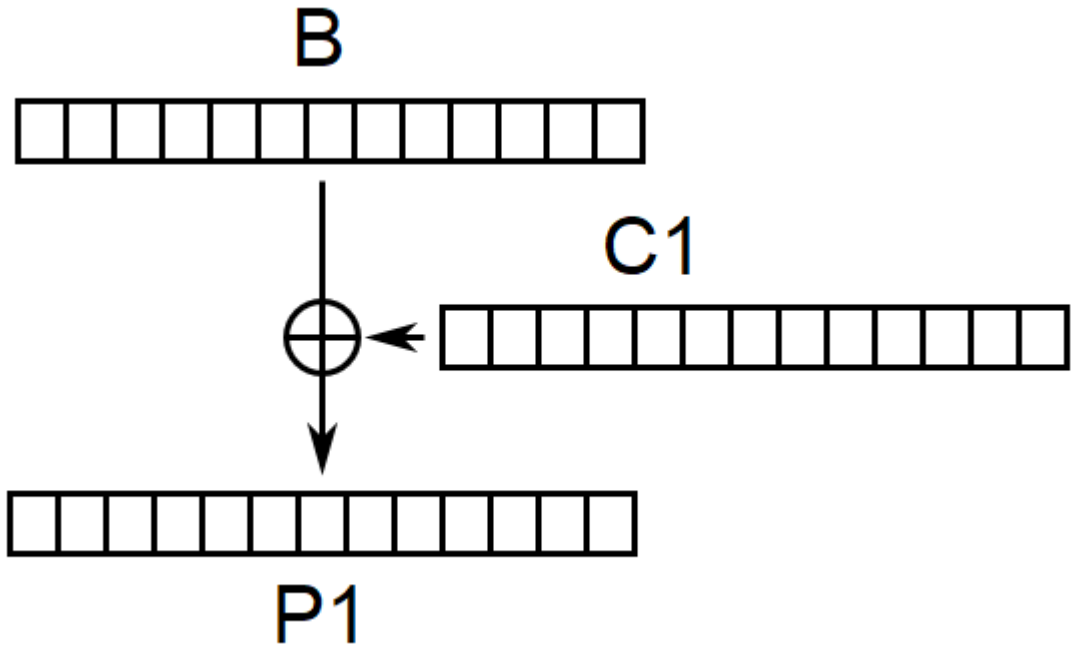


known-plaintext attack



We can abstract the output of “block cipher encryption” into **B** since we know that both IV and Key will always be the same.

known-plaintext attack (Cont.)



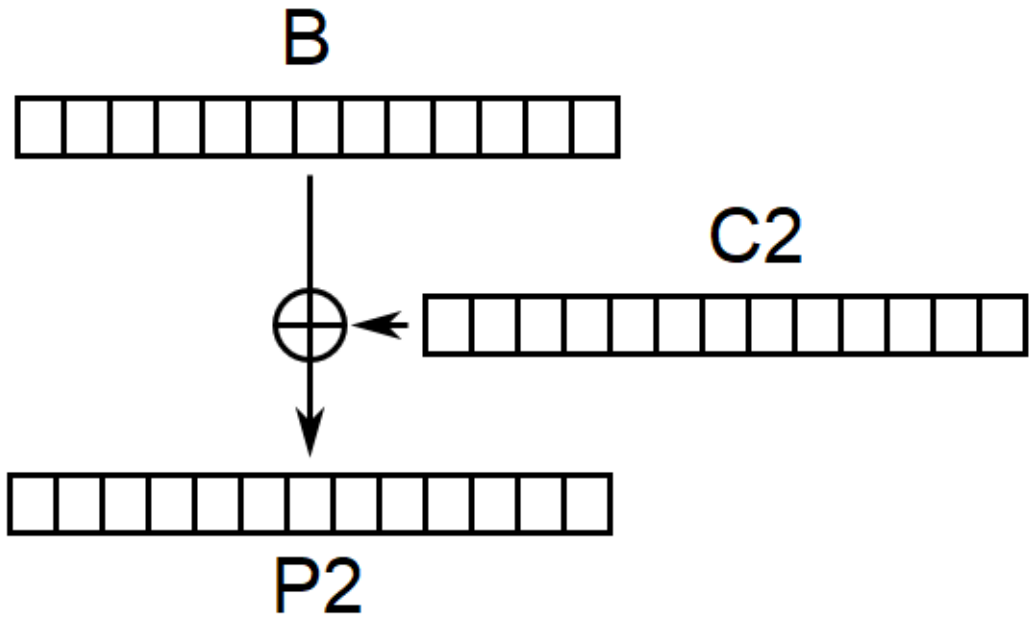
The equation for decrypting $\boldsymbol{P_1}$ is:

$$P_1 = C_1 \oplus B$$

We know the value of $\boldsymbol{P_1}$ and $\boldsymbol{C_1}$ so we can calculate the value of $\boldsymbol{B}$.

$$B = C_1 \oplus P_1$$

known-plaintext attack (Cont.)



The equation for decrypting $\boldsymbol{P_2}$ is:

$$P_2 = C_2 \oplus B$$

We know the value of $\boldsymbol{C_2}$ and $\boldsymbol{B}$ so we can calculate the value of $\boldsymbol{P_2}$

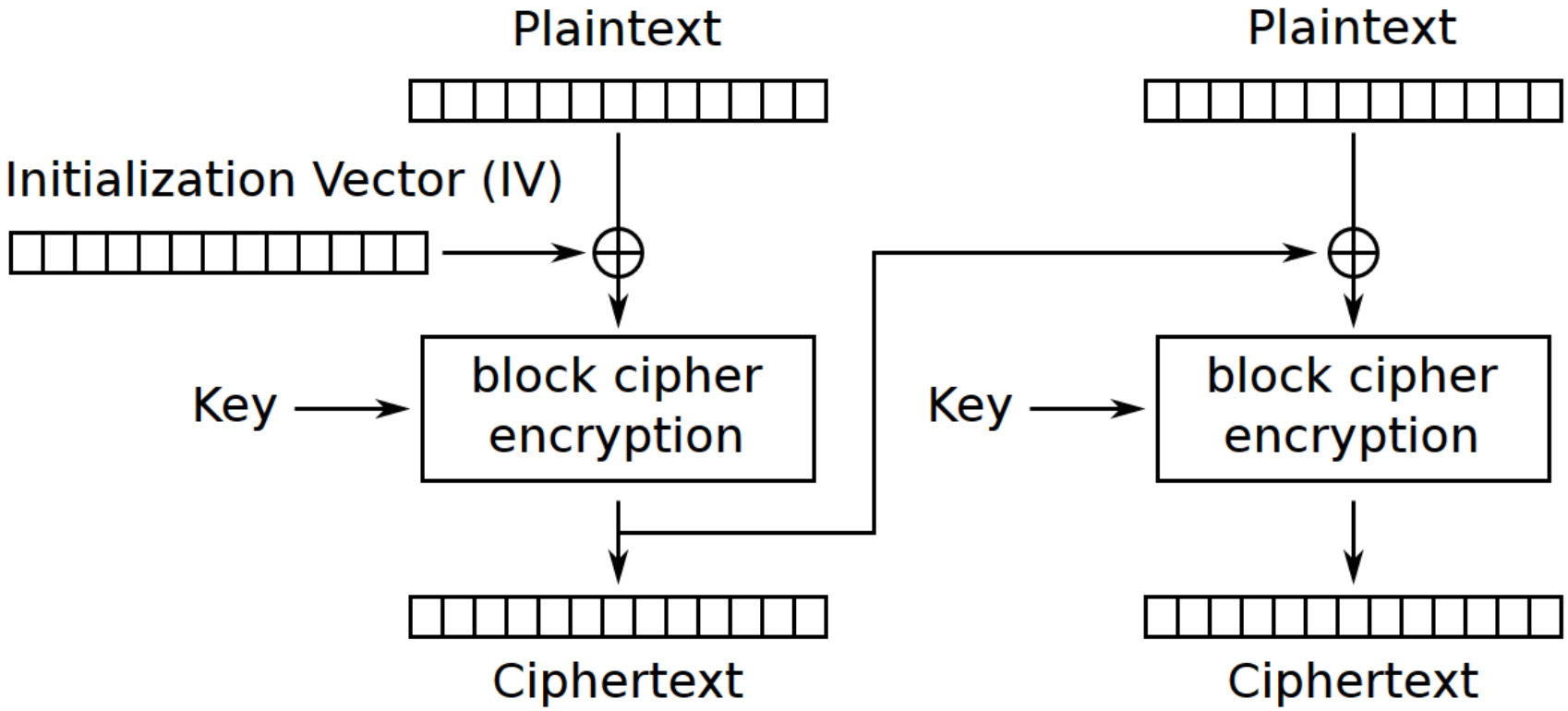
Task 6.3 Common Mistake: Use a Predictable IV

```
$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 54601f27c6605da997865f62765117ce
The IV used    : d27d724f59a84d9b61c0f2883efa7bbc

Next IV        : d34c739f59a84d9b61c0f2883efa7bbc
```

```
Your plaintext : 11223344aabbccdd
Your ciphertext: 05291d3169b2921f08fe34449ddc3611

Next IV        : cd9f1ee659a84d9b61c0f2883efa7bbc
Your plaintext : <your input>
```

- Vulnerable to ***chosen-plaintext attack.***

Cipher Block Chaining (CBC) mode encryption



Chosen-plaintext attack

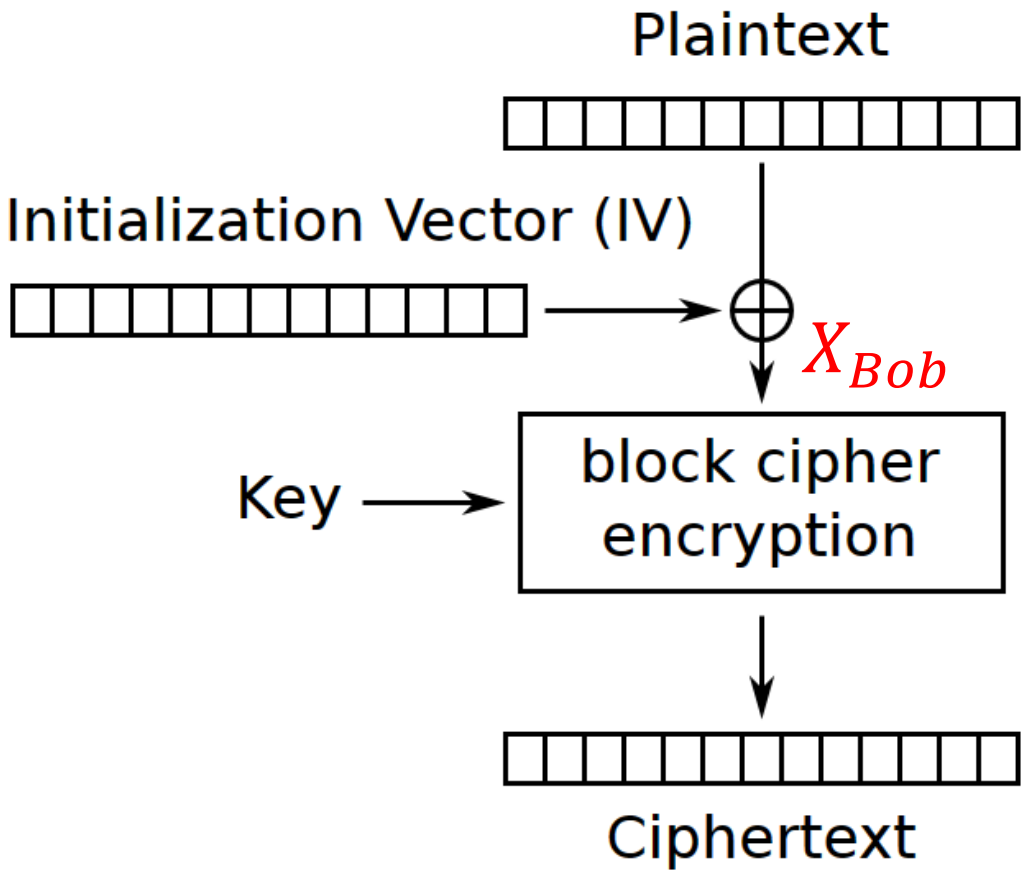
Bob's turn

List of known variables:

$$P_{Bob} = \text{"Yes" or "No"}$$

Assume that $P_{Bob} = \text{"Yes"}$

$$C_{Bob} \quad IV_{Bob} \quad IV_{You}$$



$$X_{Bob} = P_{Bob} \oplus IV_{Bob}$$

Chosen-plaintext attack (Cont.)

Your turn

List of known variables:

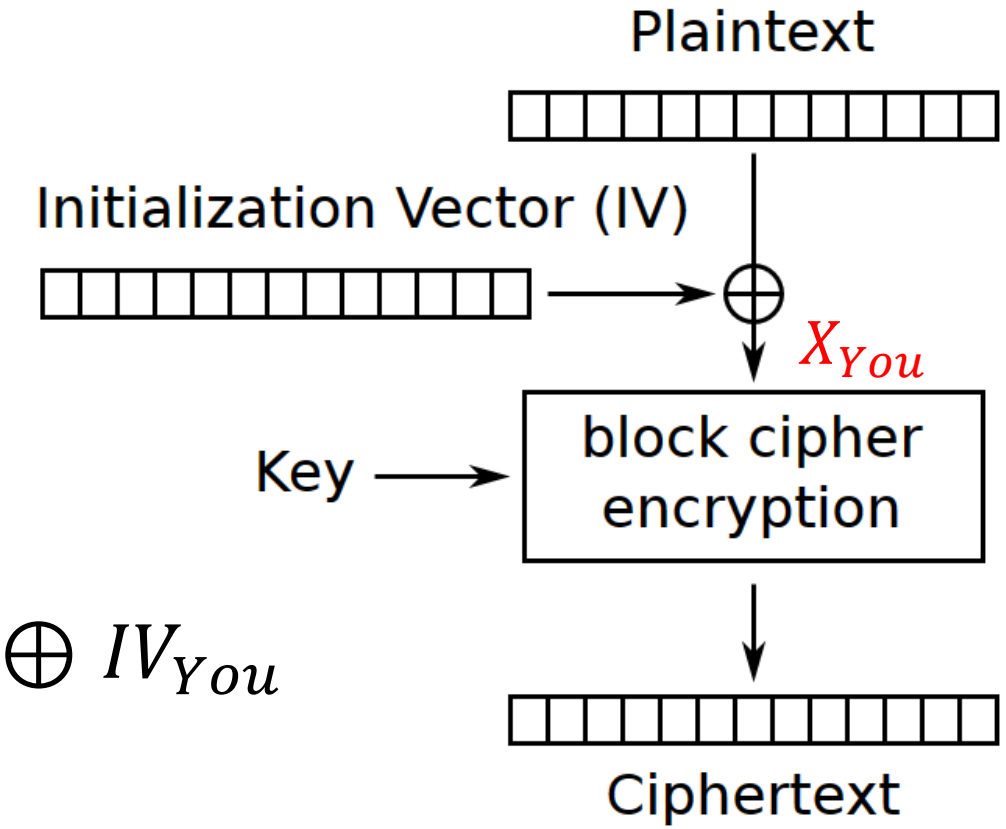
$$P_{You} = P_{Bob} \oplus IV_{Bob} \oplus IV_{You}$$

$$X_{You} = P_{You} \oplus IV_{You}$$

$$X_{You} = (P_{Bob} \oplus IV_{Bob} \oplus IV_{You}) \oplus IV_{You}$$

$$X_{You} = P_{Bob} \oplus IV_{Bob}$$

$X_{You} = X_{Bob}$ You reproduced Bob's turn.

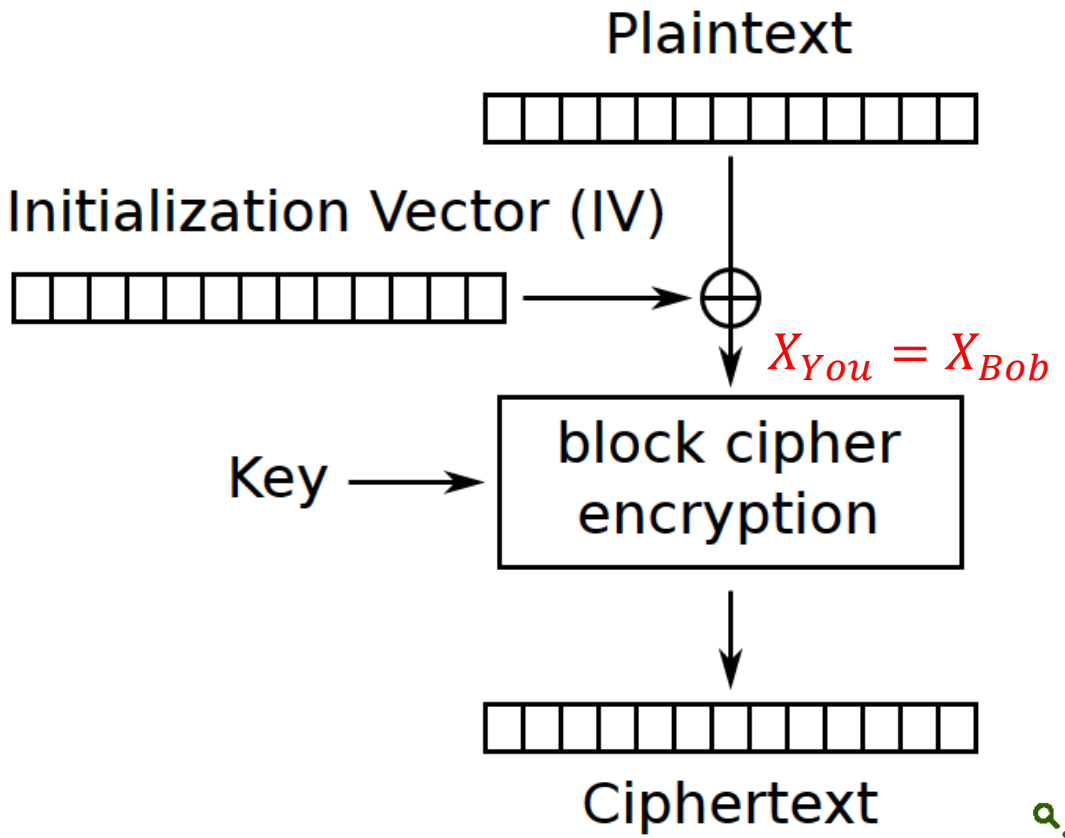


Chosen-plaintext attack (Cont.)

Note that:

$$X_{You} = X_{Bob}$$

Which means that you have reproduced Bob's turn, so if your ciphertext is the same as Bob's then we know the plaintext is "Yes", else the plaintext is "No"



Task 7: Programming with Crypto Library

جامعة بيرزيت BIRZEIT UNIVERSITY

Task 7 Programming with Crypto Library

- **Objective:**
  - Write a program to find the AES-128-CBC encryption key used for the plaintext and ciphertext below.

- **Plaintext:** "This is a top secret." (21 characters)
- **Ciphertext:** 764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
- **IV:** aabbccddeeff00998877665544332211

- **Key Details :**
  - The key is an English word shorter than 16 characters, padded with # to make it 16 characters.
  - Use an English word list for possible keys.

- **Note:**
  - You must implement the solution using the crypto library, not OpenSSL commands directly.

References

- **SEED LABS .**
- **Slides from Eng. Mohamad Balawi.**