**INFORMATION SECURITY**

**LAB ASSIGNMENT 1**

**Submitted by:-**

MIAN SAIFULLAH ALAM

**Submitted to:-**

MS AMBREEN GUL

**Registration**

SP24-BSE-005

**Date:-**

12 - OCTOBER - 2025

# **COMSATS UNIVERSITY ATTOCK CAMPUS**

**Question**

**1. Write python code for your designed stream cipher approach for encryption decryption, you can use approach from more than one already developed ciphers as given in lab practice exercises.**

**2. Design and implement an adversarial attack approach for your proposed stream cipher approach.**

**Instructions**

Prepare a clear report .doc file with code explanation and output screenshots.

**Answer**

from typing import Tuple  
  
def arc4\_crypt(data: bytes, key: bytes) -> bytes:  
  
 box = list(range(256))  
 j = 0  
 key\_len = len(key)  
 for i in range(256):  
 j = (j + box[i] + key[i % key\_len]) & 0xFF  
 box[i], box[j] = box[j], box[i]  
  
 i = 0  
 j = 0  
 out = bytearray()  
 for byte in data:  
 i = (i + 1) & 0xFF  
 j = (j + box[i]) & 0xFF  
 box[i], box[j] = box[j], box[i]  
 k = box[(box[i] + box[j]) & 0xFF]  
 out.append(byte ^ k)  
  
 return bytes(out)  
  
def vernam\_xor(data: bytes, key: bytes) -> bytes:  
  
 out = bytearray()  
 key\_len = len(key)  
 for i, b in enumerate(data):  
 out.append(b ^ key[i % key\_len])  
 return bytes(out)  
  
  
def xor\_bytes(a: bytes, b: bytes) -> bytes:  
 return bytes(x ^ y for x, y in zip(a, b))  
  
def keystream\_reuse\_attack(cipher1: bytes, cipher2: bytes, known\_plaintext1: bytes = None) -> dict:  
  
 result = {}  
 cxor = xor\_bytes(cipher1, cipher2)  
 result['c1\_xor\_c2'] = cxor  
 if known\_plaintext1:  
 L = min(len(known\_plaintext1), len(cxor))  
 recovered = xor\_bytes(cxor[:L], known\_plaintext1[:L])  
 result['recovered\_p2\_prefix'] = recovered  
 else:  
 result['recovered\_p2\_prefix'] = None  
 return result  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 key\_text = "SuperSecretKey!!"  
 key\_bytes = key\_text.encode('utf-8')  
 plaintext1 = "Dive Dive Dive".encode('utf-8')  
  
 ciphertext1 = arc4\_crypt(plaintext1, key\_bytes)  
 decrypted1 = arc4\_crypt(ciphertext1, key\_bytes)  
 print("ARC4 Example")  
 print("Plaintext1:", plaintext1.decode())  
 print("Ciphertext1 (hex):", ciphertext1.hex())  
 print("Decrypted1:", decrypted1.decode())  
 print("-" \* 50)  
  
 key2\_text = "thisismykey12345"  
 key2 = key2\_text.encode('utf-8')  
 plaintext2 = "HELLO STUDENTS".encode('utf-8')  
 cipher2 = vernam\_xor(plaintext2, key2)  
 plain2 = vernam\_xor(cipher2, key2)  
 print("Vernam-style Example")  
 print("Plaintext2:", plaintext2.decode())  
 print("Cipher2 (hex):", cipher2.hex())  
 print("Decrypted2:", plain2.decode())  
 print("-" \* 50)  
  
 pA = b"Hello Alice! This is secret A."  
 pB = b"Hello Bob! This is secret B, longer."  
 cA = arc4\_crypt(pA, key\_bytes)  
 cB = arc4\_crypt(pB, key\_bytes)  
 attack = keystream\_reuse\_attack(cA, cB, known\_plaintext1=b"Hello Alice!")  
 print("Keystream reuse attack demo (ARC4 with same key used twice)")  
 print("cA ^ cB (hex):", attack['c1\_xor\_c2'].hex())  
 print("If attacker knows prefix of pA = 'Hello Alice!'")  
 print("Recovered prefix of pB:", attack['recovered\_p2\_prefix'].decode())  
 print("-" \* 50)  
  
 v1 = vernam\_xor(b"SECRET MESSAGE ONE", key2)  
 v2 = vernam\_xor(b"OTHER MESSAGE TWO", key2)  
 attack\_v = keystream\_reuse\_attack(v1, v2, known\_plaintext1=b"SECRET ")  
 print("Vernam reuse attack demo (repeating-key XOR)")  
 print("v1 ^ v2 (hex):", attack\_v['c1\_xor\_c2'].hex())  
 print("Recovered prefix of message2 (using known prefix of message1 'SECRET '):",  
 attack\_v['recovered\_p2\_prefix'].decode())

**EXPLANATION**

**arc4\_crypt (ARC4/RC4 function)**This function implements the RC4 stream cipher: it first builds a 256-byte state array and scrambles it with the key (Key-Scheduling Algorithm), then it repeatedly updates two indices and swaps positions in the state to produce a keystream byte (Pseudo-Random Generation Algorithm). Each input byte is XORed with a keystream byte to produce output, so the same function both encrypts and decrypts because XOR is reversible.

**vernam\_xor (Vernam / repeating-key XOR):**  
This function applies a simple repeating-key XOR: it walks through the data and XORs each byte with a byte from the key, repeating the key if the data is longer; running the same function again with the same key reverses the operation and returns the original text.

**xor\_bytes (helper):**  
This small helper takes two byte strings, pairs their bytes, XORs each pair, and returns the result; it is used to compute values like c1 ^ c2 which reveal relationships between plaintexts when the same keystream has been reused.

**keystream\_reuse\_attack (attack demo):**  
This routine demonstrates the core weakness of reusing a keystream: it computes c1 ^ c2 (which equals p1 ^ p2 if the keystream is the same), and if an attacker knows some bytes of p1 they can XOR those with c1 ^ c2 to recover the same bytes of p2; this shows why reusing keys/nonces is dangerous.

**ARC4 example in main:**  
The demo creates a key and plaintext, encrypts using arc4\_crypt, then decrypts by calling the same function again to show encryption is reversible; printed hex values let you verify that encryption changed the data and decryption restored it.

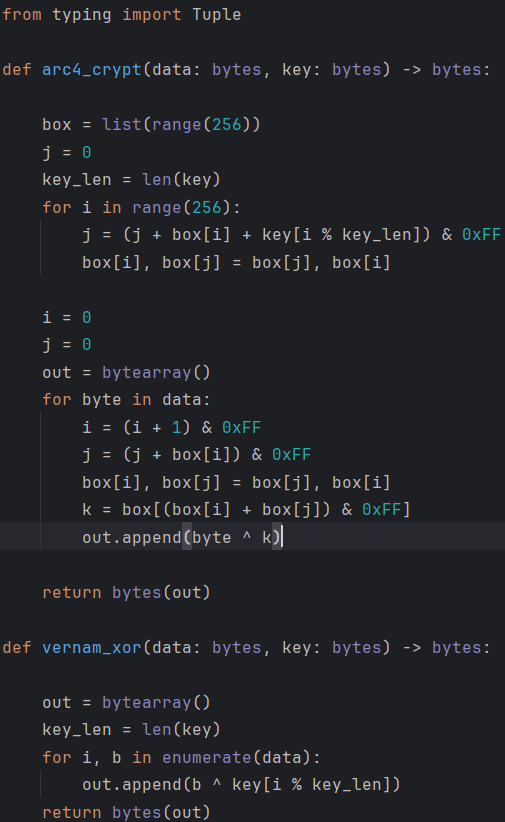
**Vernam example in main:**  
The demo shows repeating-key XOR by encrypting a short message and decrypting it with the same function, illustrating again that XOR is symmetric and that a repeating key is easy to experiment with but insecure if reused across messages.

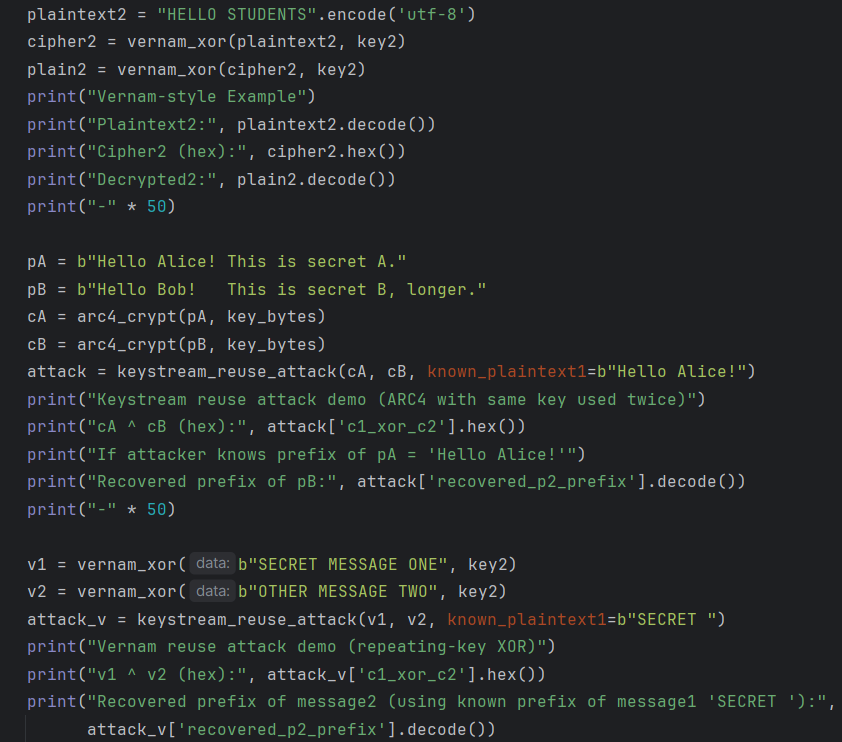
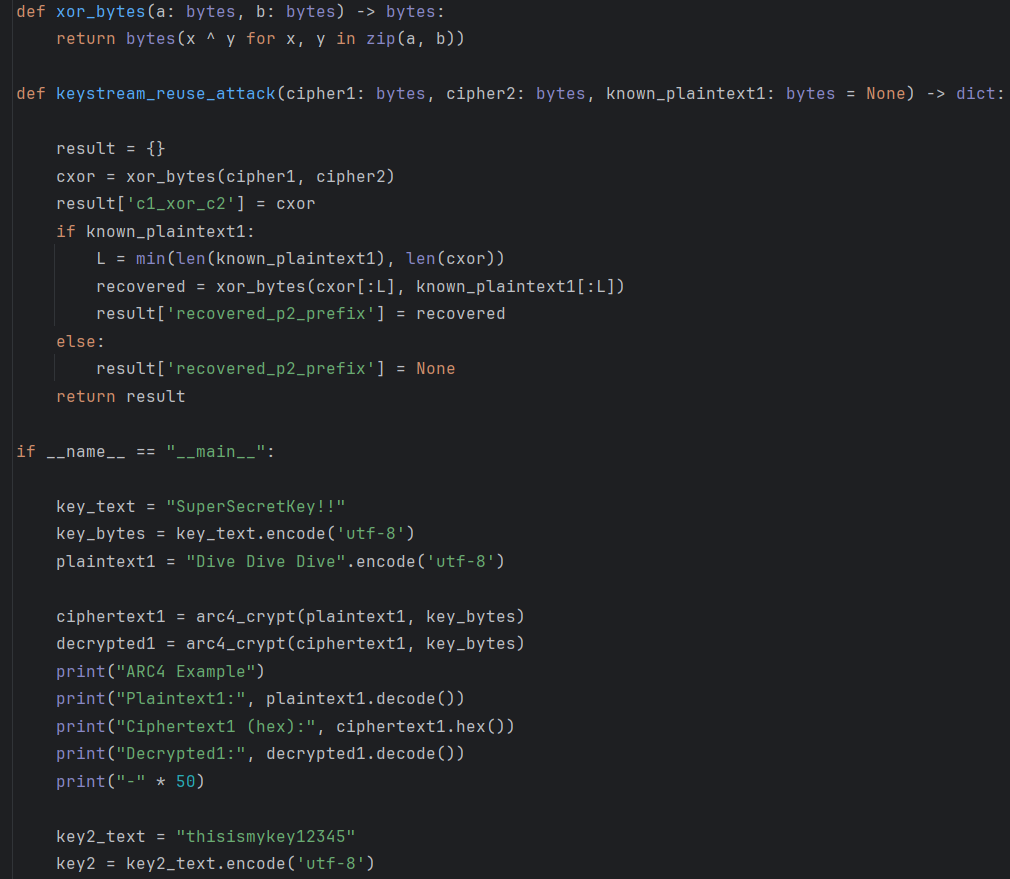
**ARC4 keystream-reuse attack demo in main:**  
This example intentionally encrypts two different messages with the same ARC4 key to produce the same keystream; the code then computes cA ^ cB and, using a known prefix of the first message, recovers the corresponding prefix of the second message — demonstrating the two-time-pad vulnerability.

**Vernam reuse attack demo in main:**  
This shows the same idea for repeating-key XOR: two messages encrypted with the same short key leak information via v1 ^ v2, and a known substring of one message can be used to reveal the corresponding substring of the other.

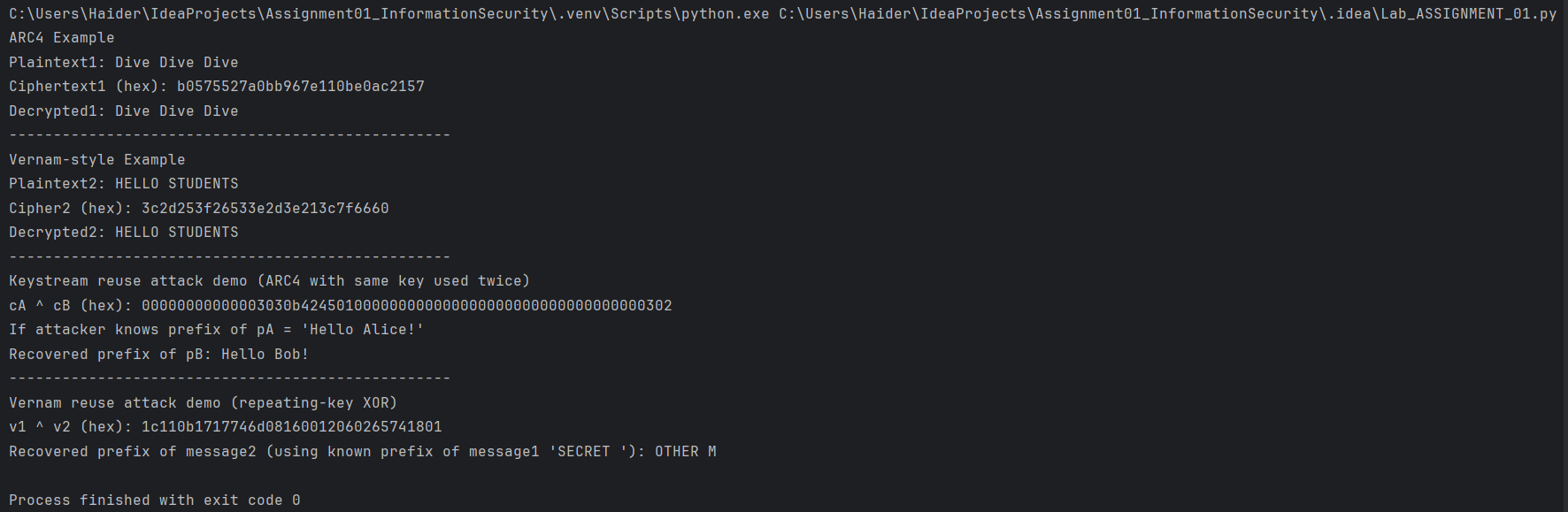
* The code shows two simple stream-cipher methods (ARC4 and repeating-key XOR), demonstrates encryption and decryption (XOR is reversible), and proves that reusing the same keystream/key for different messages lets an attacker recover secret data by XORing the two ciphertexts.

**CODE**





**Output**

****