LAB ASSIGNMENT: 01

SUBMITTED BY:

Aliza Faisal SP24-BSE-048

SUBMITTED TO: MA'AM AMBREEN GUL

DATE: 12TH OCTOBER, 2025

COURSE: Information Security

Salsa20 Stream Cipher Algorithm

Q1. 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.

Salsa20 takes your secret key and creates an endless stream of random numbers that you use to hide your messages by mixing them together using simple math operations. It employs a unique ARX (Add-Rotate-XOR) based design that provides robust security while maintaining exceptional performance across various platforms. The cipher operates on a 4×4 matrix of 32-bit words, implementing a cryptographic pseudorandom function that generates keystream blocks through iterative round transformations.

Code Explanation:

1. Import Libraries

```
from Crypto.Cipher import Salsa20
from Crypto.Random import get_random_bytes
```

- What it does: Gets the Salsa20 code tools from the library.

2. Create Key and Nonce

```
key = get_random_bytes(32)    # 256-bit password
nonce = get_random_bytes(8)   # Unique number for each message
```

What it does:

- Key = Secret password (32 characters long)
- Nonce = Unique number that changes for every message

3. Our Message

```
user_text = input("Enter message: ")  # Gets string
message = user_text.encode()          # Convert to bytes for encryption
```

- **What it does:** This is the secret message we want to protect.

4. Encryption (Hiding Message)

cipher = Salsa20.new(key=key, nonce=nonce)

secret_code = cipher.encrypt(message)

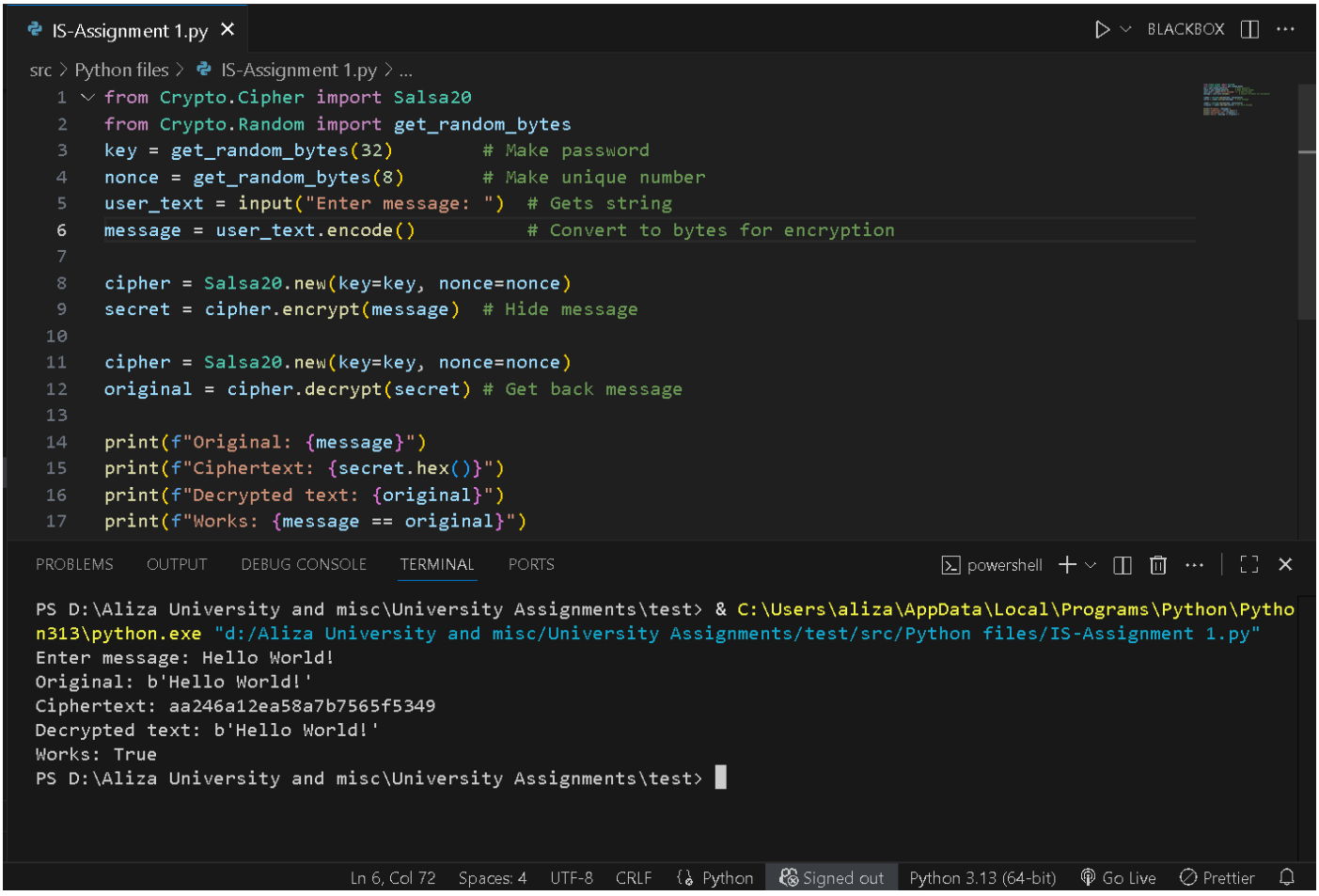
- **What it does:** Uses the key and nonce to turn our message into secret code that looks like random numbers.

5. Decryption (Getting Message Back)

cipher = Salsa20.new(key=key, nonce=nonce)

original_message = cipher.decrypt(secret_code)

- **What it does:** Uses the same key and nonce to turn the secret code back into the original message.



How Salsa20 Works (Simple):

1. **Creates Random Stream**: Salsa20 uses the key and nonce to generate a long stream of random-looking numbers.
2. **XOR Operation**: It mixes our message with these random numbers using XOR (like a light switch):
    - Same input twice = back to original

    - Message ⊕ Random = Secret Code
    - Secret Code ⊕ Same Random = Original Message
3. **Security**: Without the exact same key and nonce, nobody can read the secret code.

Q2. Design and implement an adversarial attack approach for your proposed stream cipher approach.

Nonce Reuse Attack

What Happens in the Code:

We make a big security mistake by using the same nonce twice with the same key to encrypt two different messages.

The Problem:

- Same key + same nonce = same random stream
- **Message 1:** "Pay $100 to Alice"
- **Message 2:** "Pay $500 to Alice"
- Both get encrypted with the same random stream

The Attack:
A hacker can take both secret codes and mix them together using XOR. This cancels out the random stream and reveals the relationship between the two original messages.

The Result:
Without knowing the key, the hacker can see that both payments are going to "Alice" and only the amount is different!

Why This Works:

The Math:

- Same key + same nonce = same random stream
- Encrypted1 = Message1 ⊕ RandomStream
- Encrypted2 = Message2 ⊕ RandomStream
- Encrypted1 ⊕ Encrypted2 = Message1 ⊕ Message2

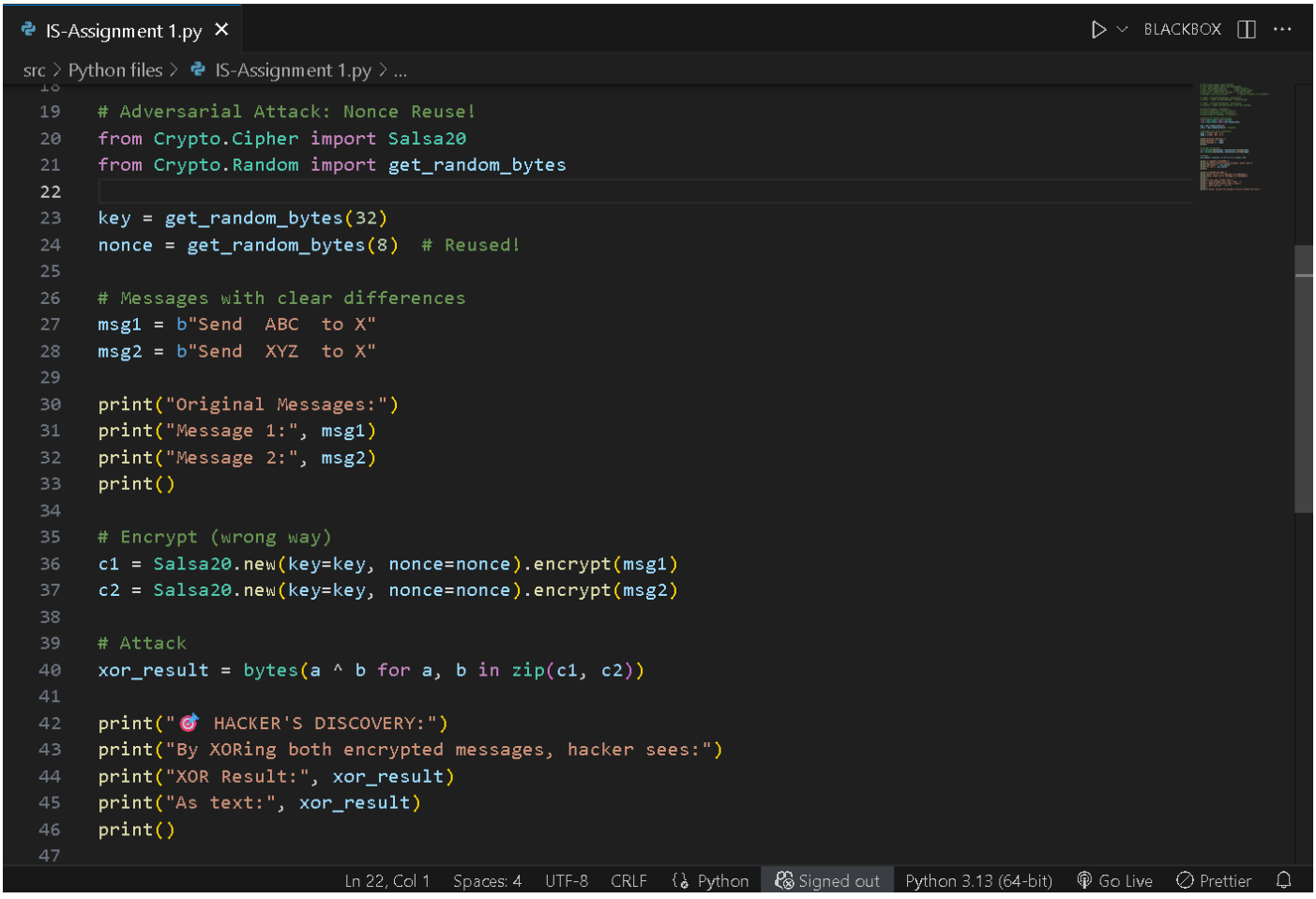
The random stream cancels out, leaving only the relationship between the two original messages!

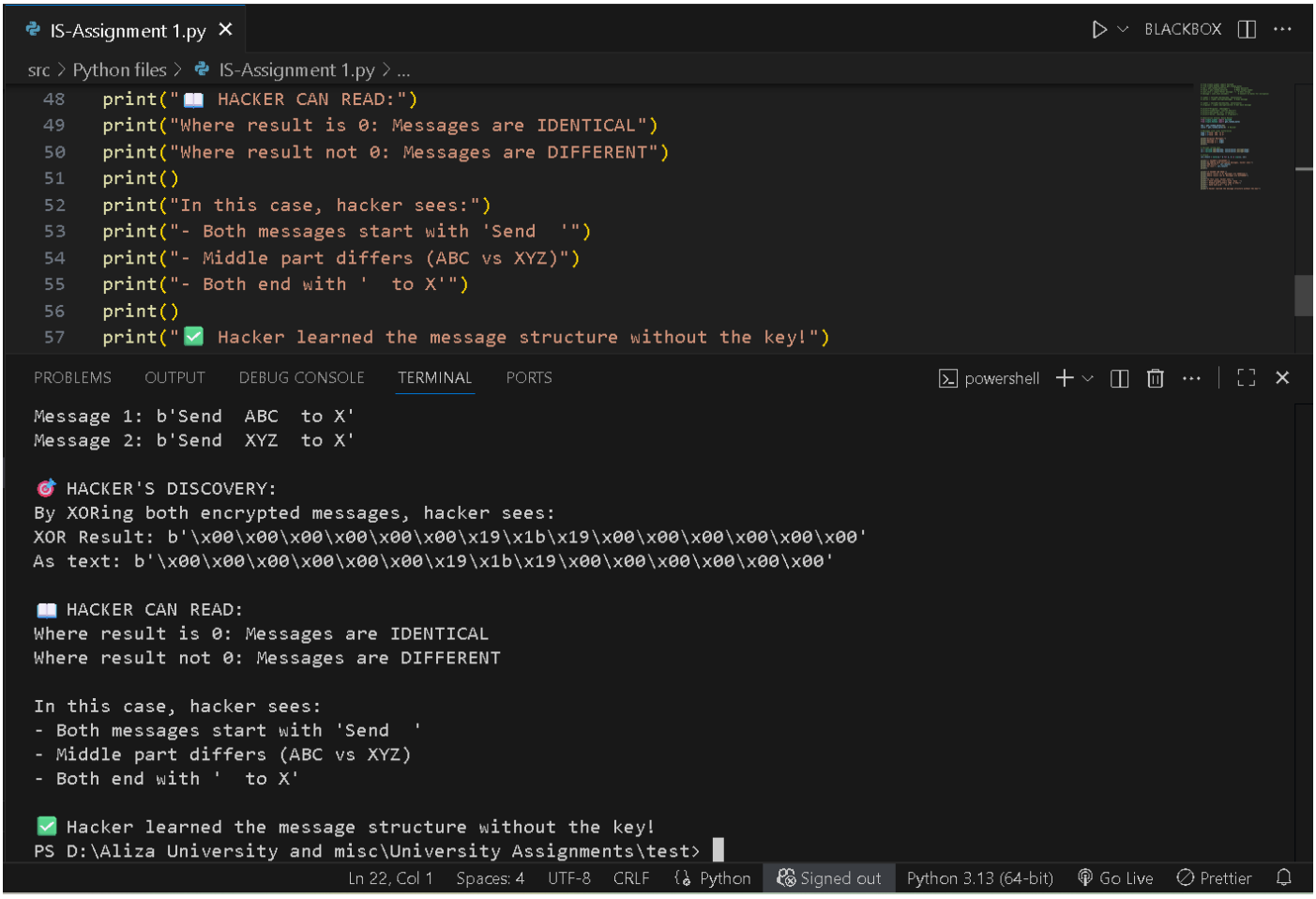
Why This is Bad:

- Hacker learns sensitive information without breaking the encryption
- Can guess one message to figure out the other
- Completely breaks the security

The Fix:

Always use a different nonce for every message with the same key.




IS-Assignment 1.py
BLACKBOX
src > Python files > IS-Assignment 1.py > ...
48 print("📖 HACKER CAN READ:")
49 print("Where result is 0: Messages are IDENTICAL")
50 print("Where result not 0: Messages are DIFFERENT")
51 print()
52 print("In this case, hacker sees:")
53 print("- Both messages start with 'Send '")
54 print("- Middle part differs (ABC vs XYZ)")
55 print("- Both end with ' to X'")
56 print()
57 print("✅ Hacker learned the message structure without the key!")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
Message 1: b'Send ABC to X'
Message 2: b'Send XYZ to X'
🎯 HACKER'S DISCOVERY:
By XORing both encrypted messages, hacker sees:
XOR Result: b'\x00\x00\x00\x00\x00\x00\x19\x1b\x19\x00\x00\x00\x00\x00\x00'
As text: b'\x00\x00\x00\x00\x00\x00\x19\x1b\x19\x00\x00\x00\x00\x00\x00'
📖 HACKER CAN READ:
Where result is 0: Messages are IDENTICAL
Where result not 0: Messages are DIFFERENT
In this case, hacker sees:
- Both messages start with 'Send '
- Middle part differs (ABC vs XYZ)
- Both end with ' to X'
✅ Hacker learned the message structure without the key!
PS D:\Aliza University and misc\University Assignments\test>
Ln 22, Col 1
Spaces: 4
UTF-8
CRLF
Python
Signed out
Python 3.13 (64-bit)
Go Live
Prettier