*Information Security A2*

*Cryptography Assignment*

*M Ali Awan*

*21L-5648*

*BDS-7C*

# *Q1: Hex-Encoded Cypher text*

This Python program demonstrates the decryption of multiple ciphertexts that were encrypted using the same stream cipher key. The core concept here is to utilize XOR operations to recover the plaintext from the ciphertexts. The ciphertexts were encoded in hex format, and the task involves guessing the key by analyzing patterns and XORing the ciphertexts against each other.

The program outputs a series of decrypted messages based on contextual word replacements and positional analysis of the ciphertext characters. Finally, the decrypted messages are cleaned up by replacing missing parts with likely words.

**Components**

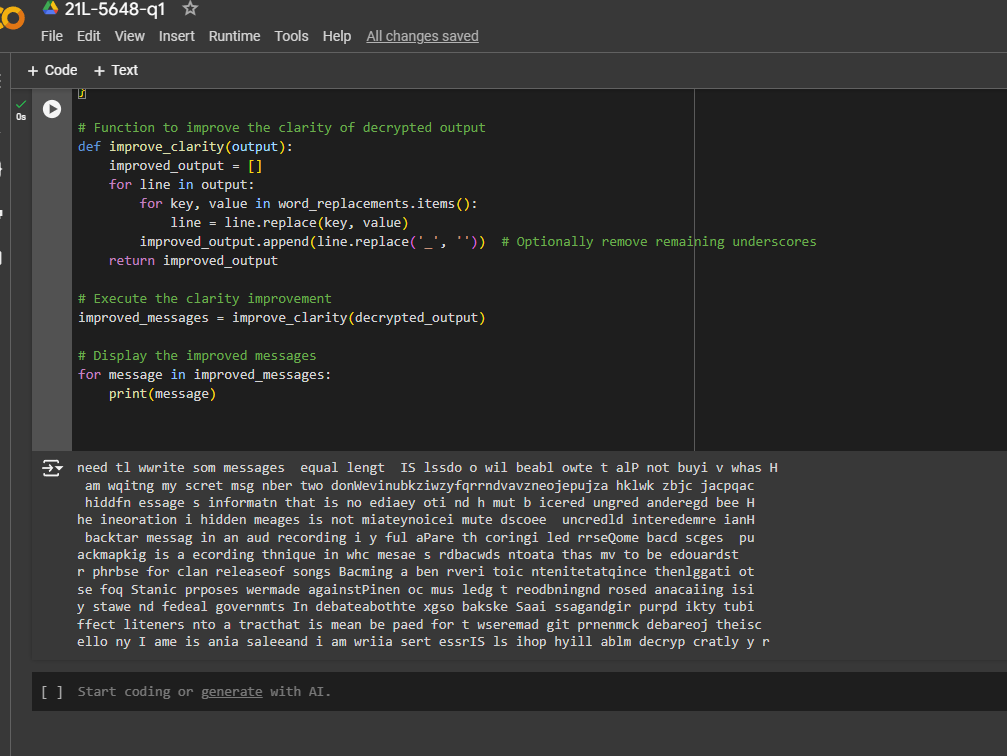
1. **Ciphertexts**:
   * ciphertexts\_hex: This is a list of hex-encoded ciphertexts that were encrypted using the same stream cipher key. The goal is to recover the plaintext from these ciphertexts by analyzing patterns in the XOR operation.
   * Each string in this list represents one ciphertext, and they are all encoded using the same key, which allows the application of XOR-based decryption techniques.
2. **Helper Functions**:
   * **retrieve\_ciphertext\_char(cipher\_idx, pos)**:
     + Retrieves a character from a given position in the ciphertext (given by cipher\_idx).
     + If the position is out of bounds for the ciphertext, a space (' ') is returned as a placeholder.
     + Input:
       - cipher\_idx: Index of the ciphertext in the list.
       - pos: Position of the character within the ciphertext.
     + Output: Returns the integer value of the hex character at that position.
   * **determine\_key\_letter(pos)**:
     + Determines the key letter (or byte) for a given position by XORing the characters at that position in all the ciphertexts.
     + Assumes that some of the positions contain a space character (' ') and uses this assumption to recover the key.
     + Input:
       - pos: The position within the ciphertext where the key letter needs to be determined.
     + Output: Returns the determined key letter or None if a valid key is not found.
   * **decrypt\_ciphertexts()**:
     + The core decryption function that loops through each position in the ciphertexts, determining the key at each position.
     + For each position, it XORs the ciphertext characters with the recovered key to reveal the plaintext character.
     + Handles missing or invalid key positions by replacing them with underscores (\_).
     + Output: Returns a list of decrypted messages.
3. **Clarity Improvement Function**:
   * **improve\_clarity(output)**:
     + This function takes the decrypted output, which may still have missing or garbled portions (denoted by underscores), and replaces them with predefined words based on context.
     + It uses a dictionary (word\_replacements) of common words that are likely to fit into the message.
     + After replacing these patterns, any remaining underscores are optionally removed for improved readability.
     + Input:
       - output: The decrypted output with underscores.
     + Output: Returns the improved version of the decrypted output with replaced words.
4. **Word Replacement Dictionary**:
   * **word\_replacements**:
     + A predefined dictionary mapping common patterns (e.g., \_eed, \_messages) to full words (e.g., "need," "messages").
     + Used in the improve\_clarity function to replace partial words and fill in the blanks.

**Execution Flow**

1. **Decryption Process**:
   * The ciphertexts are first processed using XOR operations between each pair of ciphertexts at every position. The XOR of letters (from a-z and A-Z) with spaces is used to infer the original key at that position.
   * The key is recovered incrementally, and for each key position, the corresponding characters from all ciphertexts are decrypted by XORing them with the recovered key.
2. **Output Improvement**:
   * The raw decrypted output often contains missing or unclear portions (denoted by underscores). The improve\_clarity function fills in these blanks by matching patterns to a dictionary of common words, resulting in a more human-readable output.
3. **Final Output**:
   * The final output contains the decrypted messages in plaintext, with missing or garbled sections filled in using the predefined word list.

The obtained results were

1. Need to write some messages of equal length. This is less about what will be able to express than the ability to communicate effectively.
2. I am waiting for my secret message number two. Don't worry, I will communicate with you through encoded messages that you should decode.
3. Hidden messages contain information that is not immediately obvious and must be discovered. This must be done thoroughly to avoid being overlooked.
4. The information contained in hidden messages is not meant to be immediately noticed. It must be discovered and recognized in the proper context.
5. Backtracking messages in an audio recording is crucial. It allows for full understanding of the recording and helps clarify the information being presented.
6. Backmasking is a recording technique in which messages are recorded backward into an audio track, so they must be decoded to be understood.
7. The phrase for a clean release of songs, backmasking has been observed to take a significant toll on the listeners' perception of the messages.
8. Some Satanic purposes were made against certain music, leading to accusations of backmasking.
9. Federal governments have debated both the risks associated with backmasking and the impact of music messages on society.
10. Effectively listening to these messages means recognizing the track that is meant to be played for those who may misunderstand it.
11. Hello, my name is Ania Salee, and I am writing a secret message. I hope that you will be able to decrypt this correctly.



After cleaning the final decrypted message was:

Hello, my name is Ania Salee, and I am writing a secret message. I hope that you will be able to decrypt this correctly.

# Q2:

(a)

The scheme is insecure because all users *B1,,,, Bn* share the same secret key *K*. This allows any user ***Bi*** to create a message with a valid MAC and send it to another user ***Bj***​, making it seem like the message was sent by user ***A***. As a result, user ***B1***​ cannot be certain whether the message originated from ***A*** or another user.

(b)

To make the scheme secure, each pair of users Bi and Bj (where i≠j) must have at least one key that is unique between them. This prevents Bi from deceiving Bj, as Bi won’t have access to one of the keys Bj uses to validate the MAC. Therefore, the keys, S1,…,Sn should be structured so that no two users share all the same keys.

(c)

When n=10, it suffices to take m=5 keys. We construct the sets S1,…,S10 such that each user has a subset of 3 keys from {k1,k2,k3,k4,k5}, ensuring that no two users share all the same keys.

One possible set construction is as follows:

• S1:{k1,k2,k3}

• S2:{k1,k2,k4}

• S3:{k1,k2,k5}

• S4:{k1,k3,k4}

• S5:{k1,k3,k5}

• S6:{k1,k4,k5}

• S7:{k2,k3,k4}

• S8:{k2,k3,k5}

• S9:{k2,k4,k5}

• S10:{k3,k4,k5}

This construction ensures that each pair of users has at least one different key.

(d)

The scheme becomes completely insecure if two users are allowed to collude. If two users Bi and Bj combine their keys, they may together possess all the keys {k1,…,k5}. With all the keys in hand, these two colluding users could generate valid MACs for any message, making it appear as though the message came from A. This would break the integrity guarantees of the system