Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS4320 - Applied Cryptography

Homework #1 (Symmetric Crypto Systems: Implementation and Analysis)

Prepared by:

Lina Ahmad 1220619 Section 1

Leen Alqazaqi 1220380 Section 1

Khadija Absi 1220458 Section 2

Instructor : Ahmad Shawahneh & Mohammad Hussin

Submission Date: May 16, 2025

Table of Contents

Table of figures

Task 1: Stream Cipher Cryptanalysis

1.1 Objective

In this task we aim to see how dangerous it is to reuse the same key stream. we are given ten ciphertexts that were decrypted using the same key, our goal is to fully or partially recover the key and use it to decrypt a target cipher text.

1.2 Approach and Methodology

our methodology steps:

1. **input** : we read all cipher texts from the file and converted them to bytes

2. **XORing pairs** : we want to use the property C1 $\oplus$ C2 = P1 $\oplus$ P2 (since the key is the same) so we did a pairwise xor and realized that wherever there is an alphabetical letter then there is a space in that place in one of the plaintexts

3. **Space guessing and voting** : we used the property above to guess where the space are. we also used for this a voting mechanism, the one with the most votes for that place is likely to be the place where the space actually is

4. **plaintext recovery**: any letter xored with the space will result to the toggled case of that letter so we can guess multiple parts of that text

5. **keystream recovery**: after recovering enough parts of the plaintext we can recover parts of the keystream (not full in our case) by simply XORing the cipher text with the corresponding recovered plaintext

6. **decryption of the target** : Finally, we use the recovered keystream to XOR the target ciphertext and reveal as much of the plaintext as possible.

1.3 Assumptions and Educated Guesses

ASCII letters are assumed in plaintexts.

Space (0x20) is commonly found in English plaintext and heavily relied on for guessing and recovering keystream in our task

When we XOR and get an alphabetic character, we assume a space was involved.

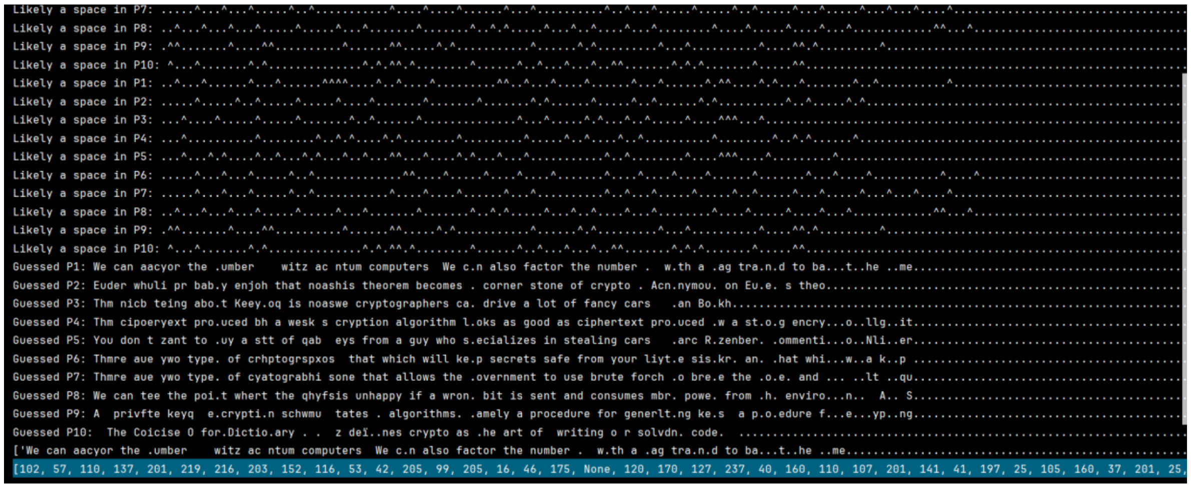
The key is reused across all ciphertexts, enabling this attack.

1.4 Recovered Keystream

[102, 57, 110, 137, 201, 219, 216, 203, 152, 116, 53, 42, 205, 99, 205, 16, 46, 175, None, 120, 170, 127, 237, 40, 160, 110, 107, 201, 141, 41, 197, 25, 105, 160, 37, 201, 25, 248, 170, 64, 26, 156, 109, 112, 143, 128, 192, 102, 199, 99, 254, 240, 18, 49, 72, 205, None, 232, 2, 208, 91, 169, 135, 119, 51, 93, 174, 252, 236, 213, 156, 67, 58, 107, 38, 139, 96, 191, 78, 240, 60, 154, 97, 16, 149, 187, None, 154, 49, 97, 237, 199, None, 10, 163, 53, 34, 207, 210, None, 210, None, 140, 87, 55, 110, 219, 168, 194, None, None, None, 124, None, None, 97, 226, 161, None, None, 69, 2, None]

this the recovered keystream and it is recovered as a list to preserve the position of the bytes and handle the unknown parts

this is a screenshot below from the terminal showing how we guessed the key :



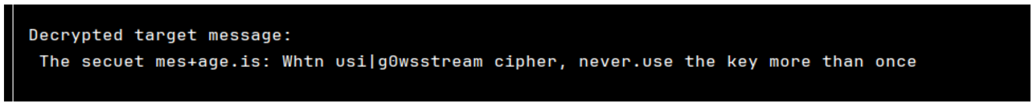
```
Likely a space in P7: .....^...^...^.....^..^...........^....^....^......^...^..........^..^.....^..^...^.....^.....^..^.....^...^.....^...^...^....^.........................................
Likely a space in P8: ..^...^...^...^.....^.....^...^.......^.......^..^.^.....^...^..^....^..^....^...^........^....^.....^....^...^............^^...^........................................
Likely a space in P9: .^^.......^....^^..........^......^^.....^.^...........^......^.^.......^.^.........^...^..........^....^^.^.........^...........................................
Likely a space in P10: ^...^.......^.^..............^.^.^^.^........^......^..^...^...^..^^...^...^..^^.......^.^.^.......^.....^^.....................................................
Likely a space in P1: ..^...^.......^...^......^^^^....^..^....^..........^^..^...^...^.......^......^...^.......^.^^....^.^...^.......^..^...........^....................................
Likely a space in P2: .....^.....^..^.....^.....^....^.......^.......^.......^.^.....^........^....^..^.......^.^..........^..^.....^.^..................................................
Likely a space in P3: ...^....^......^.....^.......^..^......^...............^....^.....^.^...^.....^.^...^..^.....^....^^^...^...................................................
Likely a space in P4: ...^..........^........^..^.^....^.^.........^.........^.....^..^....^...^..^....^..^..........^........^..^.^......^.............................................
Likely a space in P5: ...^...^.^....^..^...^.^...^..^...^^...^....^.^...^...^...........^..........^..^........^....^^^....^.........^...................................................
Likely a space in P6: .....^...^...^.....^..^.............^^....^.....^....^....^......^.......^....^....^....^......^.......^...^....^..........^....^......................................
Likely a space in P7: .....^...^...^.....^..^...........^....^....^......^...^..........^..^.....^..^...^.....^.....^..^.....^...^.....^...^...^....^.........................................
Likely a space in P8: ..^...^...^...^.....^.....^...^.......^.......^..^.^.....^...^..^....^..^....^...^........^....^.....^....^...^............^^...^........................................
Likely a space in P9: .^^.......^....^^..........^......^^.....^.^...........^......^.^.......^.^.........^...^..........^....^^.^.........^...........................................
Likely a space in P10: ^...^.......^.^..............^.^.^^.^........^......^..^...^...^..^^...^...^..^^.......^.^.^.......^.....^^.....................................................
Guessed P1: We can aacyor the .umber    witz ac ntum computers  We c.n also factor the number .  w.th a .ag tra.n.d to ba...t..he ..me...........................................
Guessed P2: Euder whuli pr bab.y enjoh that noashis theorem becomes . corner stone of crypto . Acn.nymou. on Eu.e. s theo..............................................
Guessed P3: Thm nicb teing abo.t Keey.oq is noaswe cryptographers ca. drive a lot of fancy cars   .an Bo.kh...................................................................
Guessed P4: Thm cipoeryext pro.uced bh a wesk s cryption algorithm l.oks as good as ciphertext pro.uced .w a st.o.g encry...o..llg..it...........................................
Guessed P5: You don t zant to .uy a stt of qab  eys from a guy who s.ecializes in stealing cars   .arc R.zenber. .ommenti...o..Nli..er.........................................
Guessed P6: Thmre aue ywo type. of crhptogrspxos  that which will ke.p secrets safe from your liyt.e sis.kr. an. .hat whi...w..a k..p .........................................
Guessed P7: Thmre aue ywo type. of cyatograbhi sone that allows the .overnment to use brute forch .o bre.e the .o.e. and ... ..lt ..qu.........................................
Guessed P8: We can tee the poi.t whert the qhyfsis unhappy if a wron. bit is sent and consumes mbr. powe. from .h. enviro...n..  A.. S........................................
Guessed P9: A  privfte keyq  e.crypti.n schwmu  tates . algorithms. .amely a procedure for generlt.ng ke.s  a p.o.edure f...e...yp..ng........................................
Guessed P10:  The Coicise O for.Dictio.ary . .  z deï..nes crypto as .he art of  writing o r solvdn. code.  ...............................................................
['We can aacyor the .umber    witz ac ntum computers  We c.n also factor the number .  w.th a .ag tra.n.d to ba...t..he ..me..................................................
[102, 57, 110, 137, 201, 219, 216, 203, 152, 116, 53, 42, 205, 99, 205, 16, 46, 175, None, 120, 170, 127, 237, 40, 160, 110, 107, 201, 141, 41, 197, 25, 105, 160, 37, 201, 25,
```

Figure 1:How we got the key stream

But how to guess the missing parts of the keystream?

it can be easily guessed if we guess the plaintext and as we will see in the next section we can guess the full plain text. after having the full plain text and we already have the cipher text we can xor them and get the full key stream!

1.5 Final Decrypted Message



```
Decrypted target message:
 The secuet mes+age.is: Whtn usi|g0wsstream cipher, never.use the key more than once
```

Figure 2:Final Decrypted message

as expected, the message is not full since we did not recover the full key. however, it is so obvious. **The secret message is: When using a stream cipher, never use the key more than once.**

Task 2: Implementing the Data Encryption Standard (DES)

2.1 Overview of Implementation

In this task, we implemented the full DES encryption and decryption process from scratch, without relying on any cryptographic libraries. The implementation was modular and followed the exact steps of the DES algorithm as taught in lectures.

The core of the implementation included the following parts:

- **Initial Permutation (IP)** and **Final Permutation (FP)**: These were implemented using fixed permutation tables and it was applied at the beginning and end of the encryption/decryption (on the full text not a part of it).
- **Expansion Permutation (E-box)**: The 32-bit right half of the block was expanded to 48 bits using the expansion table specified in our course slides.

- **S-Box Substitution**: the most important process since it is the non-linear part. Eight S-boxes were implemented, each taking a 6-bit input and producing a 4-bit output. This involved extracting row and column index from the 6-bit blocks and using the lookup tables (8 tables each for a block).
- **P-Box Permutation**: The output of the S-box substitution was permuted again using the P-box table.
- **Key Schedule**: The 64-bit key was processed using PC-1 to generate a 56-bit key by removing the parity parts of the key, then split into left and right halves. These halves were shifted according to the shift schedule, combined, and passed through PC-2 to generate 16 round keys. each key is for a round
- **DES Rounds**: The main Feistel loop was implemented to perform 16 rounds of encryption using the components above. Decryption was achieved by reversing the key schedule and reusing the same function.

right half : IP → split the halves → expansion e box → xor with the key for the round → enter the s box → permutation p box → xor with the left half and this is the right half

left half : right half of the previous round

repeat this for 16 round and then combine left and right halves

All intermediate steps such as binary/hex conversion and bitwise XOR … were handled manually. The encryption and decryption were verified by returning the original plaintext from the ciphertext.(we encrypted and decrypted the plaintext and got the original text at the end).

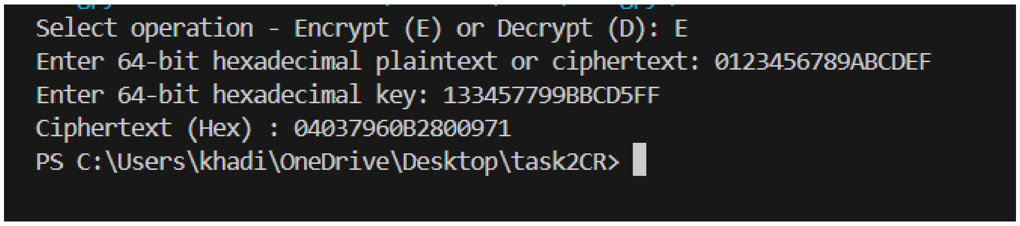
2.2 Sample Input/Output for Encryption and Decryption

<u>Input</u>:

- Operation: Encrypt
- Plaintext (64-bit): 0123456789ABCDEF
- Key (64-bit): 133457799BBCD5FF

<u>Output</u>:

- Ciphertext: 04037960B2800971



```
Select operation - Encrypt (E) or Decrypt (D): E
Enter 64-bit hexadecimal plaintext or ciphertext: 0123456789ABCDEF
Enter 64-bit hexadecimal key: 133457799BBCD5FF
Ciphertext (Hex) : 04037960B2800971
PS C:\Users\khadi\OneDrive\Desktop\task2CR>
```

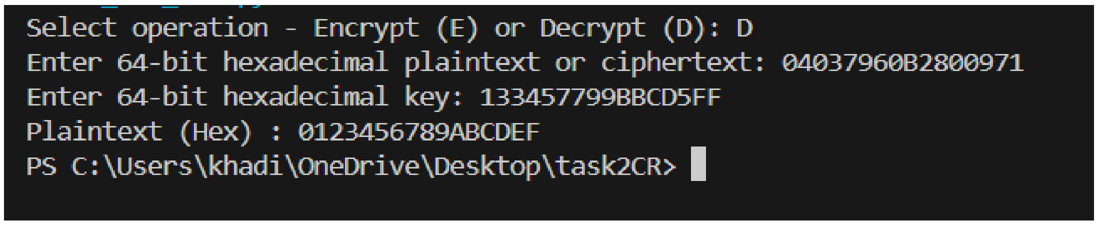
Figure 3:Sample of input and output for encrypt operation

<u>Input</u>:

- Operation: Decrypt
- Ciphertext: 04037960B2800971
- Key (64-bit): 133457799BBCD5FF

<u>Output</u>:

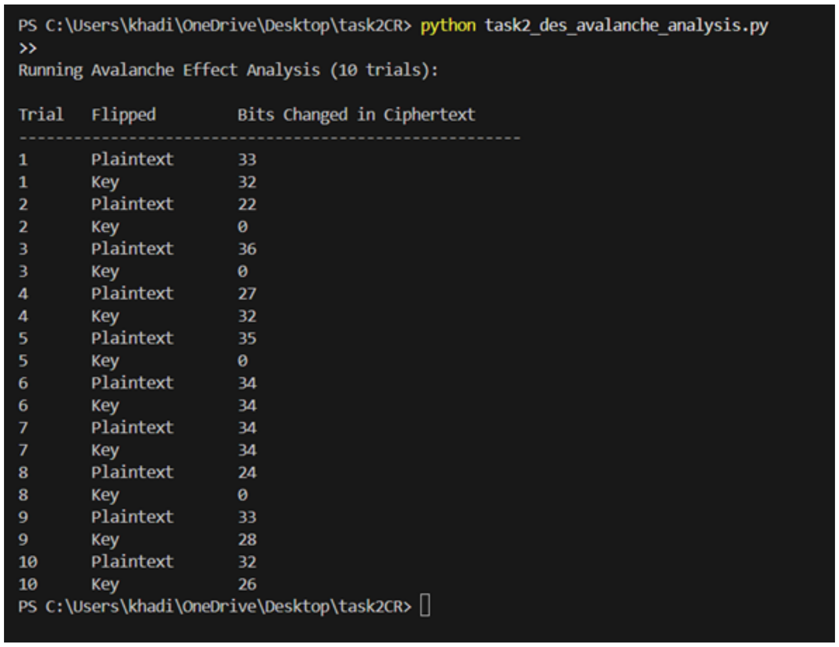
- Plaintext: 0123456789ABCDEF



```
Select operation - Encrypt (E) or Decrypt (D): D
Enter 64-bit hexadecimal plaintext or ciphertext: 04037960B2800971
Enter 64-bit hexadecimal key: 133457799BBCD5FF
Plaintext (Hex) : 0123456789ABCDEF
PS C:\Users\khadi\OneDrive\Desktop\task2CR>
```

Figure 4:Sample of input and output for decrypt operation

2.3 Avalanche Effect Results and Interpretation



```
PS C:\Users\khadi\OneDrive\Desktop\task2CR> python task2_des_avalanche_analysis.py
>>
Running Avalanche Effect Analysis (10 trials):

Trial   Flipped         Bits Changed in Ciphertext
-------------------------------------------------------
1       Plaintext       33
1       Key             32
2       Plaintext       22
2       Key             0
3       Plaintext       36
3       Key             0
4       Plaintext       27
4       Key             32
5       Plaintext       35
5       Key             0
6       Plaintext       34
6       Key             34
7       Plaintext       34
7       Key             34
8       Plaintext       24
8       Key             0
9       Plaintext       33
9       Key             28
10      Plaintext       32
10      Key             26
PS C:\Users\khadi\OneDrive\Desktop\task2CR> 
```

Figure 5:Avalanche Effect Results

<u>Summary</u>:

The results demonstrate a strong avalanche effect. On average, flipping just one bit in the plaintext or the key changes about half of the output bits in the ciphertext. This behavior is consistent with the desirable properties of a secure block cipher like DES, showing high sensitivity to input changes.

2.4 Assumptions Made

- The key is 64-bit
- The input is in hexadecimal only the code don’t accept binary input
- The encrypt and decrypt is DES functions

Task 3: Meet-in-the-Middle Attack on Triple DES

3.1 Attack Strategy and Methodology

The implemented attack is a Meet-in-the-Middle (MITM) attack on a 3DES encryption scheme, where the ciphertext is computed using the structure **C = Enc_DES(K1, Dec_DES(K2, Enc_DES(K1, P)))**. This form of 3DES applies encryption with **K1**, followed by decryption with **K2**, and then another encryption with **K1**. The attack begins by choosing a known plaintext **P** and retrieving it's corresponding ciphertext **C** from the server. In the **forward phase**, the plaintext is encrypted using all 4096 possible **K1** values to get an intermediate result, which is then decrypted with all 4096 possible **K2** values. Each output is stored in a table as potential results of the middle operation **(Dec_DES(K2, Enc_DES(K1, P))**). In the **backward phase**, the ciphertext is decrypted using each possible **K1** value, and the resulting value is compared against the table to find a match. A match indicates a valid **(K1, K2)** pair that satisfies the full 3DES structure. It is important to note that for a match to be valid, **K1** used in the forward phase must match **K1** used in the backward phase. This meet-in-the-middle strategy significantly reduces the attack complexity compared to a full brute-force search.

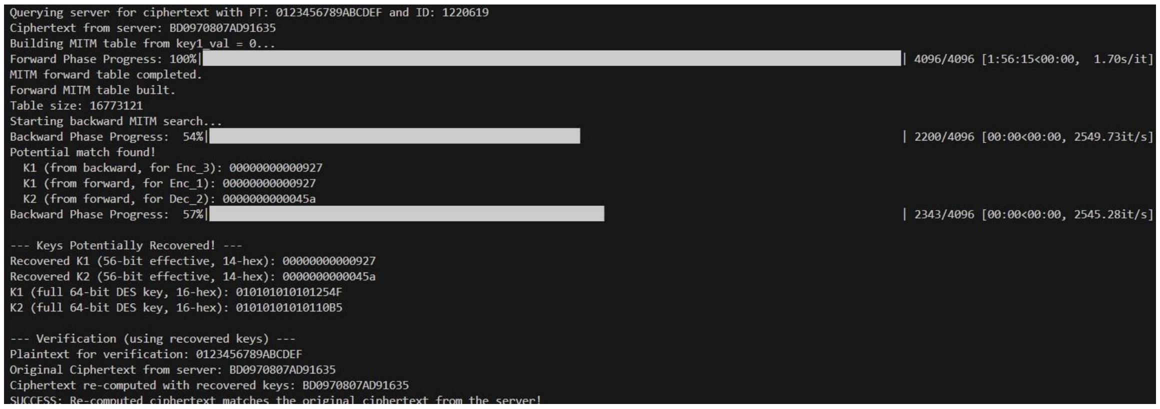
3.2 Proof of Correctness

After running the Meet-in-the-Middle attack, the program successfully recovered the two original 12-bit DES keys used in the 3DES encryption process. The keys K1 and K2 were found by identifying a match between the forward and backward intermediate computations. The recovered keys were initially in 56-bit format (14-digit hexadecimal) and were then expanded to full 64-bit DES keys (16-digit hexadecimal) by adding parity bits. The final recovered values for K1 and K2 are confirmed to correctly produce the given ciphertext from the known plaintext when used in the 3DES encryption sequence verifying the success of the attack .

3.3 Number of Queries and Operations

The Meet-in-the-Middle attack implementation performs a total of approximately 16 million DES operations. Specifically, during the forward phase, the code iterates over 4,096 possible K1 values. For each value of K1, it performs one DES encryption followed by 4,096 DES decryptions using all possible K2 keys, totaling 4096×4096= 16,777,2164096 encryption-decryption operations. This phase builds a lookup table for all intermediate results. In the backward phase, the program iterates over another 4,096 possible K1 keys, performing one DES decryption for each, and compares the result against entries in the precomputed table. Since the server is queried only once at the beginning to obtain the ciphertext for a known plaintext (using **query_server function**), the total number of external queries to the server is just **one**, while the total number of internal DES computations is approximately **16.78 million**. This reflects the typical time-memory tradeoff in MITM attacks, reducing the time to find a key pair by storing a large precomputed table.

3.4 Final Recovered Keys



```
Querying server for ciphertext with PT: 0123456789ABCDEF and ID: 1220619
Ciphertext from server: BD0970807AD91635
Building MITM table from key1_val = 0...
Forward Phase Progress: 100%|████████████████████████████████████████| 4096/4096 [1:56:15<00:00,  1.70s/it]
MITM forward table completed.
Forward MITM table built.
Table size: 16773121
Starting backward MITM search...
Backward Phase Progress:  54%|█████████████████████                   | 2200/4096 [00:00<00:00, 2549.73it/s]
Potential match found!
  K1 (from backward, for Enc_3): 00000000000927
  K1 (from forward, for Enc_1): 00000000000927
  K2 (from forward, for Dec_2): 0000000000045a
Backward Phase Progress:  57%|██████████████████████                  | 2343/4096 [00:00<00:00, 2545.28it/s]

--- Keys Potentially Recovered! ---
Recovered K1 (56-bit effective, 14-hex): 00000000000927
Recovered K2 (56-bit effective, 14-hex): 0000000000045a
K1 (full 64-bit DES key, 16-hex): 010101010101254F
K2 (full 64-bit DES key, 16-hex): 01010101010110B5

--- Verification (using recovered keys) ---
Plaintext for verification: 0123456789ABCDEF
Original Ciphertext from server: BD0970807AD91635
Ciphertext re-computed with recovered keys: BD0970807AD91635
SUCCESS: Re-computed ciphertext matches the original ciphertext from the server!
```

Figure 6: Final Recovered Keys

The image shows the successful execution of a Meet-in-the-Middle (MITM) attack on a 3DES encryption scheme.

Issues and Limitations Encountered

- In task 1 only parts of the keystream were found and the other parts were missing . This was not a big problem since the plaintext was easy to guess and we were able to recover the full key from this knowledge.
- In task 3, the code performs over 16 million DES operations and stores a large lookup table mapping all possible intermediate values, which can cause the program to slow down or consume significant memory. Also, to mitigate crashes or data loss during long executions, a check-pointing mechanism was implemented, but it adds I/O overhead and can slow performance further. Another key limitation of the implementation is the lack of parallelism. Utilizing multithreading or multiprocessing could significantly reduce execution time by distributing the workload across multiple CPU cores.

Team Member Contributions

All of the team members worked on all of the tasks equally, each member wrote a function and the others continued. Same for the report.

Conclusion

In this project, the team successfully tackled three significant tasks in applied cryptography. The first task involved cryptanalyzing a stream cipher, demonstrating the vulnerabilities of key reuse by partially and then fully recovering a keystream to decrypt a target message. The second task focused on the practical implementation of the Data Encryption Standard (DES) from scratch, which included all core components such as initial and final permutations, expansion, S-box substitution, P-box permutation, and the key schedule, ultimately verifying it's avalanche effect. Finally, the team implemented a Meet-in-the-Middle attack on a specific variant of Triple DES, successfully recovering the keys and highlighting the computational trade-offs involved in such an attack. While encountering some limitations, such as partial initial keystream recovery and the computational intensity of the Triple DES attack, the project provided a comprehensive, hands-on experience in both implementing symmetric crypto systems and analyzing their vulnerabilities.