University of Gothenburg
Department of Computer Science December 17, 2019

Cryptography: Assignment 4

Aladdin Persson (gushijal@student.gu.se),
Personnummer: 199707103934

Assignment 2, Version 1.0

Contents

1 Introduction

In this project we wish to create our own and unique translation based block cipher. Today there are different standarization processes such as AES and many others that has gone through robust testing in order to make sure it is secure and efficient. In this we wish to develop an insight on how this process can work rather than develop a new robust and secure block cipher.

In this project we generate a toy block cipher that is most likely not able to pass robust testing and is only used as a way to better understand block ciphers and how they can be implemented.

2 Assingment 4

2.1 Prerequisites and notation

A block cipher works by having an encryption algorithm **E** and a decryption algorithm **D** that acts on a **n** length bit-string. Simply explained the encryption algorithm takes as input a message **msg** and returns a ciphertext **ct**. The fundamental property that is required to hold for it to be a block cipher is that **D** should be the inverse of the encryption algorithm. Then if we decrypt the cipher text we get back our original message.

In this project we will look at a specific variant of a block cipher called translation based (TB) block cipher. This variant works by having the message and passing the message through a 'round' function and performing this round function iteratively a specific number of times which we denote as the number of **rounds**. More specifically to our project we work on 18-bit length string which first is split into three 6-bit strings that are independently passed through a substitution box (S-box). How these are specifically constructed will be explained in detail in another section. After it passes the S-box is goes through a linear mapping layer **P** that will take as input the entire 18-bit length string, and then lastly after the **P-layer** it is xor'd with a key k. This process is then repeated a number of **rounds** amount of times. Before the message is input to the round function the message is initially xor'd with a key k and this is then input to the round function.

The key k which is xor'd at the end of the round function will be a new one for each time the round function is applied. This is done by the use of a key scheduler function that takes as input the previous key and generates a new key for the consecutive round. In this project the key scheduler is implemented in octal number system and is based on a substitution. This means that we need to first convert the 18-bit string to an octal number and then perform the key scheduler and substitute on the octal representation. More specifically the key scheduler function $\phi(x)$ is defined as follows

x	0_8	1_8	2_8	3_8	4_8	5_8	6_8	7_8
$\phi(x)$	3_8	5_8	7_8	1_8	2_8	0_8	6_8	4_8

.

As mentioned the key scheduler works in the octal number system and what

this means is that each number is written using 8 as its base. For example $112_8 = 1 \cdot 8^2 + 1 \cdot 8^1 + 2 \cdot 8^0 = 73$ in our number system 10. We will be working on bit strings and therefore we need a way to convert from binary to octal and vice-versa, this is covered in the first question in the subsection *Questions*.

Notation that is used in this report is that we let $||$ denote the *concatenation* operator and we let $\oplus$ denote the *xor* operation.

2.2 Substitution and linear mapping

We take as input a 18-bit string and we have a total of three S-boxes meaning each of them works on a 6-bit string. The S-box is however defined using the octal number system and each 6-bit string is a 2 digit number in the octal number system. We therefore convert each 6-bit into a two digit octal and then perform a substitution. The S-box works as follows, it takes the first octal digit that is between 0 and 7 and looks up the first corresponding row. Then it looks at the second digit that is also between 0-7 and uses this to look up the corresponding column. For that specific row i and column j there is another two digit octal number that it will be substituted to.

	0	1	2	3	4	5	6	7
0	67	64	14	35	60	24	17	54
1	01	42	47	15	41	23	63	52
2	04	56	55	31	11	37	07	27
3	46	70	05	76	22	43	71	77
4	57	36	33	40	26	50	13	21
5	53	51	10	32	25	44	00	75
6	65	74	06	03	12	02	34	20
7	66	72	16	45	30	62	61	73

Table 1: S-box substitution mapping

It is also possible to write this in more compressed notation, such as for example writing the key scheduler in the notation $\phi(x) = 35712064$, denoting that the digit 0 is mapped to 3 and 1 to 5 etc. This can be similarly done to the S-box but the table seemed to be a more intuitive as a description of the S-box. With having the S-box predefined and everything precomputed it lends to quick computations as all that is needed is a table lookup.

The linear mapping layer $\mathbf{P}$ is defined as

$$\lambda(x) = \lambda \times x$$

where λ is a matrix, x a vector that is 18-bit in length and the $\times$ signifies matrix multiplication where the additions are instead the $\oplus$ operator. Since it takes in the entire 18-bit length the matrix is 18x18 in size. Written in compressed octal notation where it is written as such by taking columnwise and taking the three first digits and mapping it to its corresponding octal number. This results in an $6x18$ matrix instead and can be a bit more convenient to write. Written in matrix form as octal it is

$$\lambda = \begin{pmatrix} 4 & 6 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 4 & 2 & 3 & 0 & 0 & 2 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 & 4 & 2 & 3 & 6 & 0 & 0 & 2 & 2 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 2 & 1 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 4 & 5 & 2 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 6 & 0 & 2 & 0 & 5 & 2 & 1 \end{pmatrix} \quad (1)$$

2.3 Questions

Question 1: Is there a way to quickly change from octal to binary representation and viceversa?

It's possible to utilize the fact that $8 = 2^3$ and hence that each three digits in binary has a value between 0-7 in octal representation. For example 000_2 is 0_8, 001_2 is 1_8 and 011_2 is 3_8, etc. Vice-versa each number in octal has a three digit representation in binary.

Question 2: What is the 12-th round-key k_{12} obtained by the key-scheduler from the key $k = (341765)_8$. Do you think that the key scheduler is good? Why?

Running the key scheduler function iteratively we obtain the result 064521. The key scheduler works on a substitution that is a linear mapping and there-

fore if one obtains two connected keys and knowing that it works on a linear mapping it is possible to obtain how the substitution works. Even if this was not the case it is only 8! possible alternatives which is quite low. Based on this the key scheduler doesn't seem to be the best choice.

Question 3: How can you verify that your S is in fact a correct substitution box?

The substitution box needs to have a substitution for each possible 6-bit integer, in octal this is represented by values in the range of 0-77 and is a total of 64 numbers. Our S-box contains a mapping for each possible of these values, i.e it has a substitution value and is in fact unique in this fact for each possible input value. One possible way of verifying is by checking that each possible value gives a substitution value but this is evident from the table.

Question 4: Why does the linear map have to be invertible?

When performing the decryption algorithm where it is by definition that it is the inverse of the encryption the operations needs to be invertible in such a way that we are able to obtain the original message back. If it is not the case that the linear mapping is not invertible, then it is impossible to obtain the original message therefore not satisfying the fundamental property that decryption is the inverse of encryption.

Question 5: How confident are you about the security of your S and P? Are they secure? Can you prove it?

I believe it is possible to come up with attacks against the security of S and P and then give a corresponding calculation given those attacks on how long it may take to break it. The problem with cryptography is that we can only do these calculations on attacks already known or ones that is possible to come up. We cannot know for certain that it is secure since we do not know every attack against it. We can say that looking at our substitution box, assuming that the attacker does not obtain several pairs and can figure out the substitution mapping used, then by brute force attack there are 64! possible combinations. This is very many possibilities and so by bruteforce attack we are secure. The linear mapping would also be very difficult to break using a bruteforce attack since the 18x18 matrix which involves 324 entries with a naive bruteforce would

be 2^{324} possible matrices. Not all of these are possible as not all invertible and so this could be reduced further.

Question 6: Which is a good number of round? Do you think that it is more secure to have r greater than 50 or is r greater than 5 is enough?

Looking at for example AES which involves 10 rounds or DES which involves 16 rounds there is the question of where goes the limit where a brute-force attack can break it within a given timeframe and when is the rounds too many by checking the speed of execution. More rounds is preferred, but there is a trade-off between speed. In this implementation of the block cipher we have taken an arbitrary 512 rounds because it operates on very small amount of bits and is still quick enough in computation.

2.4 Test vectors

In the code that is included in this project there is a function that is called *test_vectors* that is intended to perform testing on the key scheduler function as well a few test cases to check that the basic functionality of encryption and decryption is working.

k	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
000000	000000	333333	111111	555555	000000	333333	111111	555555	000000
111111	111111	555555	000000	333333	111111	555555	000000	333333	111111
012345	012345	021753	104573	140235	017325	071453	102543	120735	014375
120735	120735	014375	041253	107523	170435	012345	021753	104573	140235
104573	104573	140235	017325	071453	102543	120735	014375	041253	107523
041253	041253	107523	170435	012345	021753	104573	140235	017325	071453
017325	017325	071453	102543	120735	014375	041253	107523	170435	012345

Figure 1: Test cases for key scheduler

msg	k	ct
111111111111111111	123456	101011010100100010
000000000000000000	654321	110001010010010011
011111111111111110	564321	110101001110000111
101111111111111101	777777	100001110100010110

Table 2: Four different test cases for a specific message, key and outputted ciphertext

Given a specific message and specific key it is also possible to track the cipher text for a specific round.

msg	k	ct0	ct1	ct2	...	ct511	ct
000000	000000	000000	355051	424250	326606	314027	111374

Table 3: Specific message and key and the returning cipher text for specific rounds