CHALMERS UNIVERSITY OF TECHNOLOGY
Dept. of Computing Sciences

Programming Assignment, Cryptography course

The programming assignment requires to *designing and implementing a translation-based block cipher*.
The group has to provide:

- **A code implementation**
- **A report** in which the block cipher design is explained in detail so that other people can re-implement the cipher. The report has to contain:
  - **All the design** choices made by the group, *i.e.* functions, permutations, implementations, etc.
  - **Answers** to some specific questions contained in this document
  - **Test vectors** that allows anyone to verify the correct implementation of the block cipher.

The learning objective of the assignment is to let the group understand how complex is the process required in the standardization process for a cryptographic primitive.

0. Introduction

Cryptography seems to be a very interesting and *"cool world"* since it provides cryptographic primitives that can be used to guarantee secure communication. It might seem that everyone can create his own cryptographic primitives and, for example, *home-brew* its own blockcipher!
This should **not** be the case.
In fact, there are different standardization processes, like AES[1], NESSIE[2] or the current CAESAR[3], in order to define *which* block cipher is more secure, efficient and should be used.
The standardization process is in fact a competition, like a crypto-Hunger Game! Different research teams design their own primitives, register them in the contest and then the goal is to *break* the ciphers submitted by each research team.
Only one design will remain and will be recommended by the standardisation process to be used.

Designing cryptographic primitives is **hard**. But **why?**
In this assignment we will go through the process of *designing a toy block cipher*.

---

[1]https://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process
[2]https://en.wikipedia.org/wiki/NESSIE
[3]https://competitions.cr.yp.to/caesar.html

1. Translation Based Block Ciphers

A block cipher is defined by an *encryption* algorithm E and a decryption algorithm D acting on a fixed length bit-string $\{0,1\}^{\mathsf{n}}$ of length n.
E takes as input a message msg and a key k and encrypts it into a ciphertext ct.
D takes a ciphertext ct and a key k and outputs a message msg.
The main property we require is that, whenever we fix a key k, the decryption algorithm D is the inverse of the encryption E with the same key. Formally $\mathsf{D}(\mathsf{k}, \mathsf{E}(\mathsf{k}, \mathsf{msg})) = \mathsf{msg}$.

Notation: Let $\|$ denote the *concatenation* operator. If a and b are bit-strings, $a\|b$ is the concatenation of the two strings. Let $\oplus$ be the *xor* operation.

In this assignment, we consider the *translation based (TB)* block-ciphers (or known as *substitution-permutation*) as depicted in Figure 1.

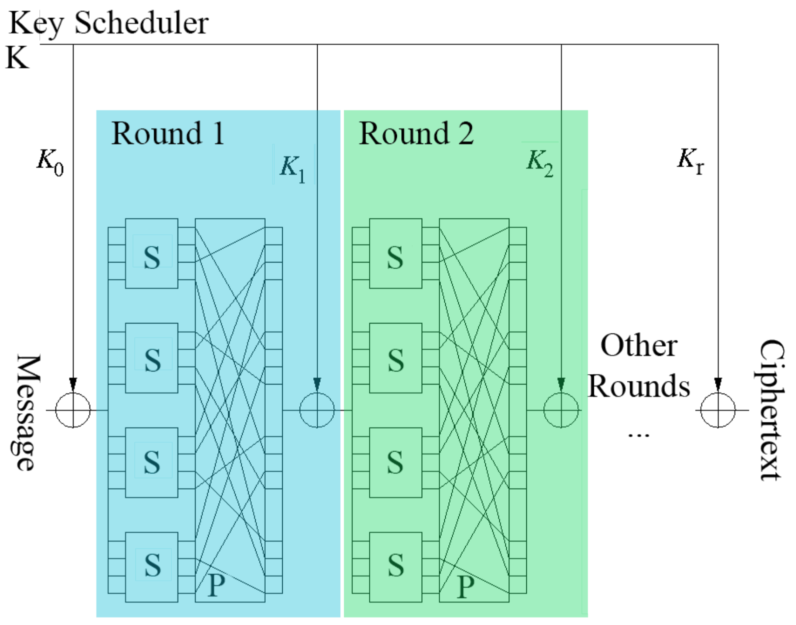

Figure 1: Encryption algorithm of a TB block cipher.

A TB block-cipher is an **iterated block-cipher design**. This means that the block-cipher is defined with a **round function** R that is iterated for a specific **number of round** r. In this way, it is just necessary to define the round function R and the number of rounds r in order to completely describe an iterated block-cipher. We define the ***i*-th partial ciphertext** ct_i the output of the i-th round function.

A mandatory algorithm used in block-ciphers is the **key-scheduler** H. This algorithm takes a key k and returns a finite list of keys $\{\mathsf{k}_0, \mathsf{k}_1, \dots, \mathsf{k}_{\mathsf{r}}\}$ called round-keys. It is important to notice that the number of keys is $\mathsf{r}+1$ where r refers to the number of rounds and the *"+1"* key, the k_0 key, is xored with the original message.
Therefore, the message msg is xored with the round-key k_0, then the i-th round will use the i-th round-key k_i.

What defines an iterated block-cipher into a translation-based is the specific composition of the round. A TB round is defined using a permutation map S on short bit-substrings and then a linear map P over the whole message space.

This means that the TB block-cipher is defined over the message space of bit-string $\{0,1\}^{\mathsf{n}}$ of length n. In the beginning of the round, the partial ciphertext ct_{i-1} is subdivided into m sub-strings of dimension l, namely $\mathsf{ct}_{i-1}[1]\|\ldots\|\mathsf{ct}_i[l]$. In parallel, every sub-string is mapped via the permutation S which is a permutation over $\{0,1\}^l$ (*"short bit-substring"* since $l < \mathsf{n}$) obtaining what we call $\mathsf{S}(\mathsf{ct}_{i-1}) = \mathsf{S}(\mathsf{ct}_{i-1}[1])\|\mathsf{S}(\mathsf{ct}_{i-1}[2])\|\ldots\|\mathsf{S}(\mathsf{ct}_{i-1}[\mathsf{m}])$. Finally, we concatenate the whole message and apply the linear map P defined over $\{0,1\}^{\mathsf{n}}$ and xor the round key k_i obtaining $\mathsf{P}(\mathsf{S}(\mathsf{ct}_{i-1})) \oplus \mathsf{k}_i$ which is the new partial ciphertext ct_i.

Therefore, by having the key scheduler H, a specific number of rounds r, a design for the round function R using a permutation map S and a linear map P, it is possible to completely define a TB block-cipher.
Note: it is still necessary to fix the message and key length, n long bit-string, and the dimension l of the permutation map S. Observe, and/or convince yourself, that it is implicit that the length of the messages is equal to the dimension of the permutation map multiplied by the number of them, *i.e.* $\mathsf{n} = l \cdot \mathsf{m}$.

By combining all these notions, it is possible to formally define a TB-block cipher:

Definition 1. *Let the message space M, the ciphertext space C and the key space K be the set of bit-strings of length $\{0,1\}^n$. Let l and m be positive integers such that $n = l \cdot m$. Consider a permutation $S : \{0,1\}^l \to \{0,1\}^l$ a bit-string of length l and a linear transformation $P : \{0,1\}^n \to \{0,1\}^n$. Let r be a positive integer indicating the number of rounds. Consider the key scheduler $H : K \to K^{r+1}$ that takes as input a key k and return a set of round-keys $\{k_0, k_1, \ldots, k_r\}$ with index i. A* translation-based block-cipher (E, D) *is defined by the length of the bit-string used l, m, n, the number of rounds r, the specific permutation/linear map S, P and a key scheduler H. A sketch representation of the cipher construction can be seen in Figure 1.*

- ***Before the rounds:***
  The message msg is xor-ed with the key k_0. The result is the partial ciphertext ct_0.

- ***For any round** $i \in \{1, \ldots, \mathbf{r}\}$:*
  The round function $R(i, ct_{i-1}, k_i)$ is the i-th round function that takes as input the $i-1$-th partial ciphertext and the i-th round key k_i. The partial ciphertext ct_i is splitted into m substrings of length l, $ct_{i-1} = ct_{i-1}[1]\|ct_{i-1}[2]\|\ldots\|ct_{i-1}[m]$ and on every sub-string, the permutation S is computed obtaining $S(ct_{i-1}[1])\|S(ct_{i-1}[2])\|\ldots\|S(ct_{i-1}[m])$ that we can denote with $S(ct_{i-1})$. Afterwards, the result is mapped via the linear map P and then the round key k_i is xor-ed with the result obtaining $P(S(ct_{i-1})) \oplus k_i$. The result is the partial ciphertext ct_i.

- ***After the rounds:***
  The final ciphertext ct is the final partial ciphertext ct_r.

In the *"wild"* there are a lot of attacks and design principles for block ciphers. We hardly recommend to just give a look to Avanzi's book "A Salad of Block Ciphers"[4] that contains a state-of-the-art discussion and explanation on *how* to design a block-cipher, *how* to prove the security, *which* are the faulty design choice and a lot of different *implementation* of real block ciphers.

[4]https://eprint.iacr.org/2016/1171.pdf

1. Project Specification

In order to easily represent the boxes S and P, we will consider the octal[5] numerical system and represent numbers with a subscript 8. For example, 123_8 is in fact the number $83 = 1 \cdot 8^2 + 2 \cdot 8^1 + 3 \cdot 8^0$ or, in binary, $(001\ 010\ 011)_2$.
Notice that the binary representation has the most-significant bit on the left!

$$(100)_2 = 8 \qquad (010)_2 = 2 \qquad (001)_2 = 1$$

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

For the sake of simplicity, your goal is to design a 18-bit translation-based block cipher with a substitution box S of 6-bits. The message, ciphertext and key space is $\{0,1\}^{18}$. The key-scheduler $\mathsf{H} : \mathsf{K} \to \mathsf{K}^{\mathsf{r}+1}$ is given and it is recursively defined using the octal permutation ϕ as defined in the Table 1.

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

Table 1: ϕ permutation used in the key-scheduler H.

The first round-key k_0 is equal to the input key k. Then, given i-th round-key k_i, the $(i+1)$-th round-key k_{i+1} is obtained by converting the bit-string into the octal representation, applying octal-wise the permutation ϕ and reverse the string before re-joining the pieces. Formally,

$$\mathsf{k}_i \to \Big(\mathsf{k}_i[1] \cdots \mathsf{k}_i[6]\Big)_8 \longrightarrow \Big(\phi\big(\mathsf{k}_i[6]\big) \cdots \phi\big(\mathsf{k}_i[1]\big)\Big)_8 = \mathsf{k}_{i+1} \tag{1}$$

For example, if the round-key is $\mathsf{k}_i = (012345)_8$, then the next round-key k_{i+1} is

$$\Big(012345\Big)_8 \to \Big(021753\Big)_8$$

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

As stated in Definition 1, we have defined $l, \mathsf{m}, \mathsf{n}$ and H.

It is now your choice to define/design the substitution box $\mathsf{S} : \{0,1\}^6 \to \{0,1\}^6$ the linear map $\mathsf{P} : \{0,1\}^{18} \to \{0,1\}^{18}$ and the number of rounds r.

We highly suggest you to define the S in octal notation, similarly to the permutation ϕ.

Question 3: How can you verify that your S is in fact a correct substitution box?
(Hint: What does it mean "substituting"?)

To simplify the notation in our case, permutations can be written as a octal string by just writing the ordered outputs of the permutation.
For example, let us consider the general description for S in Table 2 where every octal input x will be sent to two octal numbers NN_x. By just considering the S line, we can

x	00	01	$\dots$	07	10	11	$\cdot$	77
$\mathsf{S}(x)$	NN_{00}	NN_{01}	$\dots$	NN_{07}	NN_{10}	NN_{11}	$\cdot$	NN_{77}

$$\mathsf{S} = NN_{00} \| NN_{01} \| \dots \| NN_{07} \| NN_{10} \| NN_{11} \| \cdot \| NN_{77}$$

Table 2: S permutation notation trick.

just represent the S map with the octal string obtained by concatenating the outputs, which is $NN_{00} \| NN_{01} \| \dots \| NN_{07} \| NN_{10} \| NN_{11} \| \cdot \| NN_{77}$. As a concrete example, the permutation used for the key scheduler ϕ can be denoted as $\phi = 35712064$.

In order to design the linear map P, let us start with an example and consider a small map $\lambda : \{0,1\}^3 \to \{0,1\}^3$ defined as the multiplication of x with a specific bit-matrix. x is an octal number, therefore it is represented with 3 bits $x = (x_1\, x_2\, x_3)_2$.

$$\lambda(x) = \lambda \times x = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

If we compute the row-column multiplication, we obtain

$$\lambda(x) = \quad x_1 \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \oplus x_2 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \oplus x_3 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad = \begin{pmatrix} x_1 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \\ x_2 \oplus x_3 \end{pmatrix}$$

Now, if we consider the input $x = (101)_2$, we have that $\lambda(x) = (001)_2$.

In order to compress the notation, we can represent the matrix in octal-notation too!
We can take the bit-string and read them as *"up-to-down"* columns and change them to octal notation. In our example, we obtain that λ can be represented as

$$\lambda(x) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times x = \begin{pmatrix} (6)_8 & (3)_8 & (7)_8 \end{pmatrix} \times x$$

A longer bit string will have a longer octal representation. We highly suggest for your project to use the octal-notation. You final matrix will therefore will therefore be represented as a $6 \cdot 18 = 108$ octal-numbers written in a 6 rows and 18 column table that can be compressed into a single octal-string that is the concatenation of the rows. As an example, let us consider the matrix η and the procedure to obtain a compressed representation.

$$\eta = \left(\begin{array}{cccccc} 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right) = \begin{pmatrix} 7 & 2 & 7 & 7 & 0 & 7 \\ 4 & 7 & 2 & 6 & 1 & 7 \end{pmatrix} = 727707\,472617$$

The only requirement for the matrix λ is to be invertible[6]. In our example, the inverse map λ^{-1} is $\lambda^{-1} = 375$.

Question 4: Why does the linear map have to be invertible?
(Hint: We are now describing the "encryption" algorithm. What about the "decryption"?)

[5]https://en.wikipedia.org/wiki/Octal
[6]https://en.wikipedia.org/wiki/Invertible_matrix

If everything went well, you should now have a substitution box S and an invertible linear matrix to be used into the linear map P.

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

The last missing piece is a number of rounds r. Pick whatever number you prefer!

Question 6: Which is a *good* number of round? Do you think that it is more secure to have $\mathsf{r} > 50$ or $\mathsf{r} > 5$ is enough?

Congratulations! You now have designed a TB block cipher!
The next step is to explain *what* you have chosen, *why* these specific functions and *how* you might allow other to re-implement your cipher.

2. General Specification Report

In order to better understand the requirement, please take a look at the structure of the specification for the PRESENT-Toy[7] or LED[8] block ciphers. Other examples can be easily be found on-line.

The design-report is the **blueprint** of your cipher. It has to be easy to read and understand, it has to contain all the information needed to re-implement it. For this reason, the report should be well-structured and should contains:

1. *Notation used*: the designers notation used in their report;
2. *S and P*: it is mandatory to clearly state and define S and P. It is mandatory to explain *how* designers have chosen these functions;
3. *Security/Efficiency discussion*: in this section, designers explain and motivate their choices, either using mathematical proofs or empirical tests. In the next section;
4. *Test Vectors*: in order to check if the implementation is correct.

Regarding the last point, test vectors are really important to verify the correctness of an implementation. There is *no standard way* to provide test-vectors. you could provide either test vector for a complete encryption:

$$\text{input message } \mathsf{msg} + \text{ input key } \mathsf{k} \longrightarrow \text{ output ciphertext } \mathsf{ct}$$

or the partial output of every round:

$$\text{input message } \mathsf{msg} + \text{ input key } \mathsf{k} \longrightarrow \text{ round outputs } \left(\mathsf{ct}_0, \mathsf{ct}_1, \dots, \mathsf{ct}_\mathsf{r}\right)$$

By providing a good list of vectors $(\mathsf{msg}, \mathsf{k}, \mathsf{ct})$, or $(\mathsf{msg}, \mathsf{k}, (\mathsf{ct}_0, \mathsf{ct}_1, \dots, \mathsf{ct}_\mathsf{r}))$, anyone can test the correct implementation.

To better explain the test-vector importance, let us consider the following example, the key-scheduler H you have to use in your project and some test-vector to verify the correct implementation of H.

[7] https://pdfs.semanticscholar.org/7082/7ac50fa184893fe68314d741b8b85baba36e.pdf
[8] https://eprint.iacr.org/2012/600.pdf

Let us consider the key $\mathsf{k} = 012345$.
By the definition of H, we have that $\mathsf{k}_0 = \mathsf{k} = 012345$. Afterwards, $\mathsf{k}_1 = 534602$ and it is obtained by applying Equation 1 and related instruction in the same section.

A list of test-vector that you can use to check the correct implementation of the key scheduler H is, in octal notation, contained in Table 3.

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

Table 3: Key-scheduler H test-vectors.

As you might expect, in order to provide the test vectors, it is **necessary** to have a *code implementation* of your cipher. Therefore, you will have to include it.
It is **mandatory** that the code is commented, clean and easy to use.

3. Your Specification Report!

Of course it is quite hard to write the third section about *"Security"*.
That is why **your report must contain**, as separate sections:

1. **Notation used**: you will need a section where you have to explain the *octal* notation, how your block cipher is designed, *i.e.* translation-based;
2. **S and P**: state and describe your functions
3. **Answer to the assignment questions**
4. **Test Vectors Table**: 3-4 vectors in the format $(\mathsf{msg}, \mathsf{k}, \mathsf{ct})$ and the partial evaluations, for all the r rounds, of the message $\mathsf{msg} = (00\ 00\ 00)_8$ with key $\mathsf{k} = (00\ 00\ 00)_8$.

The test-vector table will therefore have the format represented in Table 4.

msg	k	ct
mmmmmm	*kkkkkk*	*cccccc*
mmmmmm	*kkkkkk*	*cccccc*
mmmmmm	*kkkkkk*	*cccccc*
mmmmmm	*kkkkkk*	*cccccc*

msg	k	ct_0	ct_1	ct_2	...	ct_r	ct
000000	000000	*cccccc*	*cccccc*	*cccccc*	...	*cccccc*	*cccccc*

Table 4: Format for the final report test-vector.

4. Submission Check-list and Correction Criteria

You code implementation **must be**:

- ☐ properly commented and the code has to be explained
- ☐ properly structured and written
- ☐ easy to understand
- ☐ easy to use, should include a few lines in the report on *"how to use the code"*
- ☐ the key-scheduler H has to be implemented.
  Use the test-vectors of Table 3 to verify your implementation!

We **do not** impose any particular limitation on the programming language used and it should not be required to use specific additional packages.
We *highly recommend* (but not require) to use a pretty common language such as C, Java, Python, Haskell, Magma. If you think that your language is too *"exotic"*, just contact the TA responsible for the PA and ask the permission to use the *"exotic"* language.

You report **must be structured as**:

- ☐ **Title page** with
  - *name, email, personnummer* of the members of your group
  - *name* of your block-cipher
  - *date*
  - *version* of your work[9]
- ☐ **Notation section** in which you explain all the preliminaries sufficient to understand your design. Some of these point are:
  - *octal* notation
  - how a *translation-based* block-cipher is defined
- ☐ **S and P** definition which has to be *clear and easy* to understand.
  Some **mandatory** checks and explanations you have to provide are:
  - ☐ state the number of rounds r you chose
  - ☐ S has to be a permutation in compressed octal notation (128 octal number)
  - ☐ P has to be invertible written in compressed octal notation (108 octal number)
- ☐ **Answer to the questions**
- ☐ **Test vectors**:
  - ☐ *3-4 vectors* in the format $(\mathsf{msg}, \mathsf{k}, \mathsf{ct})$
  - ☐ *partial evaluations*, for all the r rounds, of the message $\mathsf{msg} = (00\ 00\ 00)_8$ with key $\mathsf{k} = (00\ 00\ 00)_8$

[9] Just to differentiate in case of re-submission.

Your final submission *zip* **must contain**:

- ☐ the **"implementation"** folder with your implementation code
- ☐ *if necessary*, a **"readme.txt"** file where you explain *how to use* your code
- ☐ the report **"report_GroupNumber.pdf"**

The correction process will be made progressively and in the following order:

- ☐ *reading and verifying* the content (except the test vectors) of the report.
  If there is a missing check-box, the assignment will be rejected.
- ☐ *looking and executing* the code implementation.
  If there are issues or not-clear parts, the assignment will be rejected.
- ☐ *verifying the test vectors*.
  If there are issues, the assignment will be rejected.

For any doubt or question, you can always contact the TA responsible of the assignment.

Hint: just keep it simple and discuss a lot with your partner. Work as a group!

Useful links for curious groups:

- At the time of the creation of this assignment, there is a NIST contest on lightweight[10] block ciphers!
- In this wikipedia page[11] it is possible to see a list of different symmetric key ciphers and their security.

[10] https://csrc.nist.gov/projects/lightweight-cryptography
[11] https://en.wikipedia.org/wiki/Cipher_security_summary