Experiment 8: Applications of Cryptography

Due date: 22.05.2022 @23.59

Res. Asst. Esin Ece Aydın
aydinesi16@itu.edu.tr

"It is better to create than to learn! Creating is the essence of life."
Gaius Julius Caesar

1 Introduction

Secure communication is one of the essential needs of today. People need security and privacy in their communications. Cryptography provides many techniques and methods for secure communication. Although cryptography was utilized especially in the military field in the past, it is possible to come across its applications in almost all technological devices today.

In this experiment, three important cryptography applications in the history of cryptography will be implemented as digital designs. First, the implementation of Caesar Cipher, named after the Roman Republic general and dictator Julius Caesar, will be implemented. Caesar Cipher was one of the first examples of cryptography in history. Julius Caesar used this encryption technique to communicate with his generals during the war [1].

The second technique will be the implementation of the Vigenère Cipher used by the Confederate Army. Unlike the Caesar Cipher, the Vigenère Cipher encrypts messages using a key. Both parties, the sender and the receiver, must have the same key for conducting the communication [2].

Finally, the implementation of the Enigma machine used by the German Army in the World War 2 and a sample communication environment will be realized. The Enigma machine was used to encrypt and decrypt secret messages. According to experts, the defeat of the German Army was accelerated by 2 years thanks to the work of the Ultra group, which deciphered the Enigma and other encryption machines [3].

You can use any operator/code block in this experiment. You must simulate all parts and modules.

2 Part 1: Helper Modules

In this part you will design 4 helper modules. Detailed information is given below.

- **CharDecoder Module:** Transform A-Z chars to binary decoded version.

  Ex: 'A' → h0000001, 'Z' → h2000000, 'T' → h0080000

  - **char (8-bit input)**: ASCII code of the character.
  - **decodedChar (26-bit output)**: Decoded output of the character.

- **CharEncoder Module:** Transform binary decoded version of the char to ASCII code for A-Z char.

  Ex: h0000020 → 'F', h0000400 → 'K', h0020000 → 'R'

  - **decodedChar (26-bit input)**: Decoded input of the character.
  - **char (8-bit output)**: ASCII code of the character.

- **CircularRightShift Module:** Right Shift Module with shift count.

  - **data (26-bit input)**: Input data.
  - **shiftAmount (5-bit input)**: Shift amount.
  - **out (26-bit output)**: Data shifted to the right 'shiftAmount' times.

- **CircularLeftShift Module:** Left Shift Module with shift count.

  - **data (26-bit input)**: Input Data.
  - **shiftAmount (5-bit input)**: Shift amount.
  - **out (26-bit output)**: Data shifted to the left 'shiftAmount' times.

3 Part 2: Caesar Cipher

In this part, you are required to implement Caesar Cipher Encryption and Decryption modules. Then you'll create a Caesar Module to show encrypted and decrypted messages. Figure 1 shows the encryption technique of Caesar Cipher when the shift count is 3. Example inputs and outputs of the encryption module are 'A' → 'D', 'F' → 'I', and 'X' → 'A'. For the decryption module, 'E' → 'B', 'H' → 'E', 'J' → 'G', etc. Required modules and their information are given below.

- **CaesarEncryption Module:** Encrypt the char using Caesar Cipher technique.
  - **plainChar (8-bit input)**: ASCII code of the input character.
  - **shiftCount (5-bit input)**: Shift amount of the cipher.
  - **chipherChar (8-bit output)**: The character which is encrypted using Caesar Cipher with shiftCount.

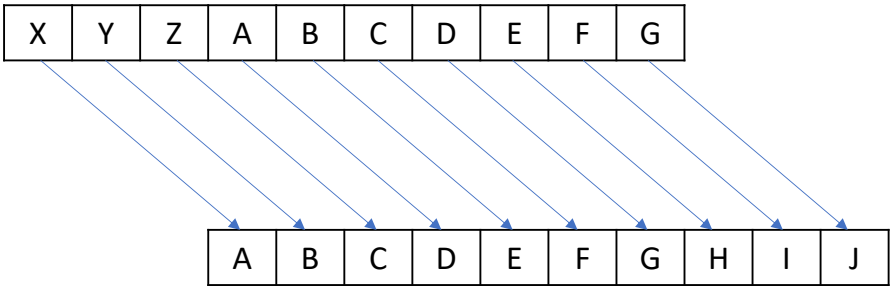

Figure 1: Caesar Encryption

- **CaesarDecryption Module:** Decrypt the encrypted char using Caesar Cipher technique.
  - **chipherChar (8-bit input)**: Encrypted char input.
  - **shiftCount (5-bit input)**: Shift amount of the cipher.
  - **decryptedChar (8-bit output)**: The character which is decrypted using Caesar Cipher with shiftCount.

- **CaesarEnvironment Module:** Encryption and decryption processes of a plain char. Figure 2 shows the block scheme of the CaesarEnvironment Module.
  - **plainChar (8-bit input)**: ASCII code of the input character.
  - **shiftCount (5-bit input)**: Shift amount of the cipher.
  - **chipherChar (8-bit output)**: The character which is encrypted using Caesar Cipher with shiftCount.
  - **decryptedChar (8-bit output)**: The character which is decrypted using Caesar Cipher with shiftCount.

Hint: You can use helper modules for encryption and decryption operations.

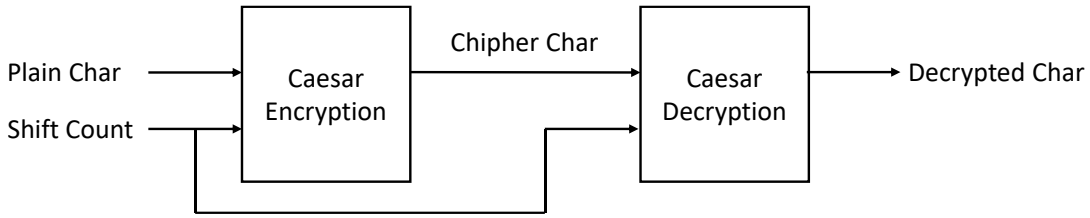

Figure 2: Caesar Encryption and Decryption Process

4 Part 3: Vigenère Cipher

In this part, you are requested to implement the Vigenère Cipher Encryption and Decryption modules. Then, you will create a Vigenère Module to show encrypted and decrypted messages. Vigenère Cipher uses the same key message for encryption and decryption processes. Equation 1 and Equation 2 show encryption and decryption calculations of the Vigenère Cipher, respectively. Table 1 shows an example for the encryption and decryption calculations of the Vigenère Cipher whose key message is "KADIROZLEM" and the plain text is "ISTANBULTECHNICAL".

$$C_i = (P_i + K_i) \quad mod \quad 26 \tag{1}$$

where C_i is the cipher char, P_i is the plain char, and K_i is the key char.

$$D_i = (C_i - K_i) \quad mod \quad 26 \tag{2}$$

where D_i is the decrypted char, C_i is the cipher char, and K_i is the key char.

Table 1: Example Calculation of Vigenère Cipher

i	**Plain Char**	**Key Char**	P_i	K_i	C_i	**Cipher Char**	D_i	**Decryted Char**
0	I	K	8	10	18	S	8	I
1	S	A	18	0	18	S	18	S
2	T	D	19	3	22	W	19	T
3	A	I	0	8	8	I	0	A
4	N	R	13	17	4	E	13	N
5	B	O	1	14	15	P	1	B
6	U	Z	20	25	19	T	20	U
7	L	L	11	11	22	W	11	L
8	T	E	19	4	23	X	19	T
9	E	M	4	12	16	Q	4	E
10	C	K	2	10	12	M	2	C
11	H	A	7	0	7	H	7	H
12	N	D	13	3	16	Q	13	N
13	I	I	8	8	16	Q	8	I
14	C	R	2	17	19	T	2	C
...	...	...	...	...	...	...	...	...

Required modules and their information are given below for Vigenère Cipher.

- **VigenereEncryption Module:** Encrypt the char using Vigenère Cipher technique.
  - **plainChar (8-bit input)**: ASCII code of the input character.
  - **keyInput (80-bit input)**: Key message of the cipher (10 characters).

- **load (1-bit input)**: Load the keyInput to key register at the rising edge of the load signal.
- **clock (1-bit input)**: Shift key register only one character at the rising edge of the clock signal (when load = 0).
- **chipherChar (8-bit output)**: The character which is encrypted using Vigenère Cipher.

- **VigenereDecryption Module:** Decrypt the encrypted char using Vigenère Cipher technique.
  - **chipherChar (8-bit input)**: Encrypted char input.
  - **keyInput (80-bit input)**: Key message of the cipher (10 characters).
  - **load (1-bit input)**: Load the keyInput to key register at the rising edge of the load signal.
  - **clock (1-bit input)**: Shift key register only one character at the rising edge of the clock signal (when load = 0).
  - **decryptedChar (8-bit output)**: The character which is decrypted using Vigenère Cipher.

- **VigenereEnvironment Module:** Encryption and decryption processes of the plain char. Figure 3 shows the block scheme of the VigenereEnvironment Module.
  - **plainChar (8-bit input)**: ASCII code of the input character.
  - **keyInput (80-bit input)**: Key message of the cipher.
  - **load (1-bit input)**: Load signal of modules.
  - **clock (1-bit input)**: Clock signal of modules.
  - **chipherChar (8-bit output)**: The character which is encrypted using Vigenère Cipher.
  - **decryptedChar (8-bit output)**: The character which is decrypted using Vigenère Cipher.

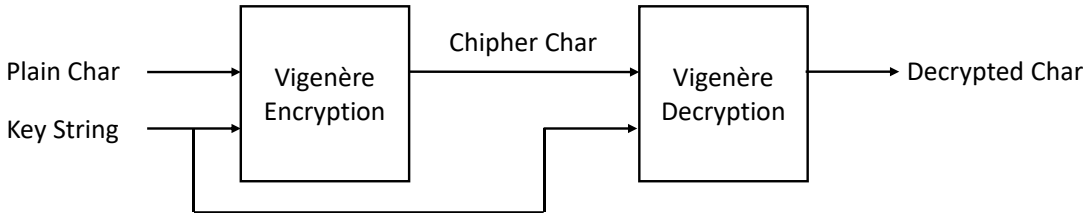

Figure 3: Vigenère Encryption and Decryption Process

5 Part 4: Enigma Machine

In this part, you are request to implement the Enigma machine. The Enigma machine can both encrypt and decrypt a message. For secure communication, receiver and transmitter must have Enigma machines which are configured and initiated identically. The rotors in the Enigma machine rotate every time a new input character is given and this rotation changes the configuration of the machine entirely. This way, a character is decrypted differently every time it is given as an input. Figure 4 shows an example character generation. Plugboard, Rotor1, Rotor2, and Rotor3 get 26-bit inputs and change the orders of the bits for output.

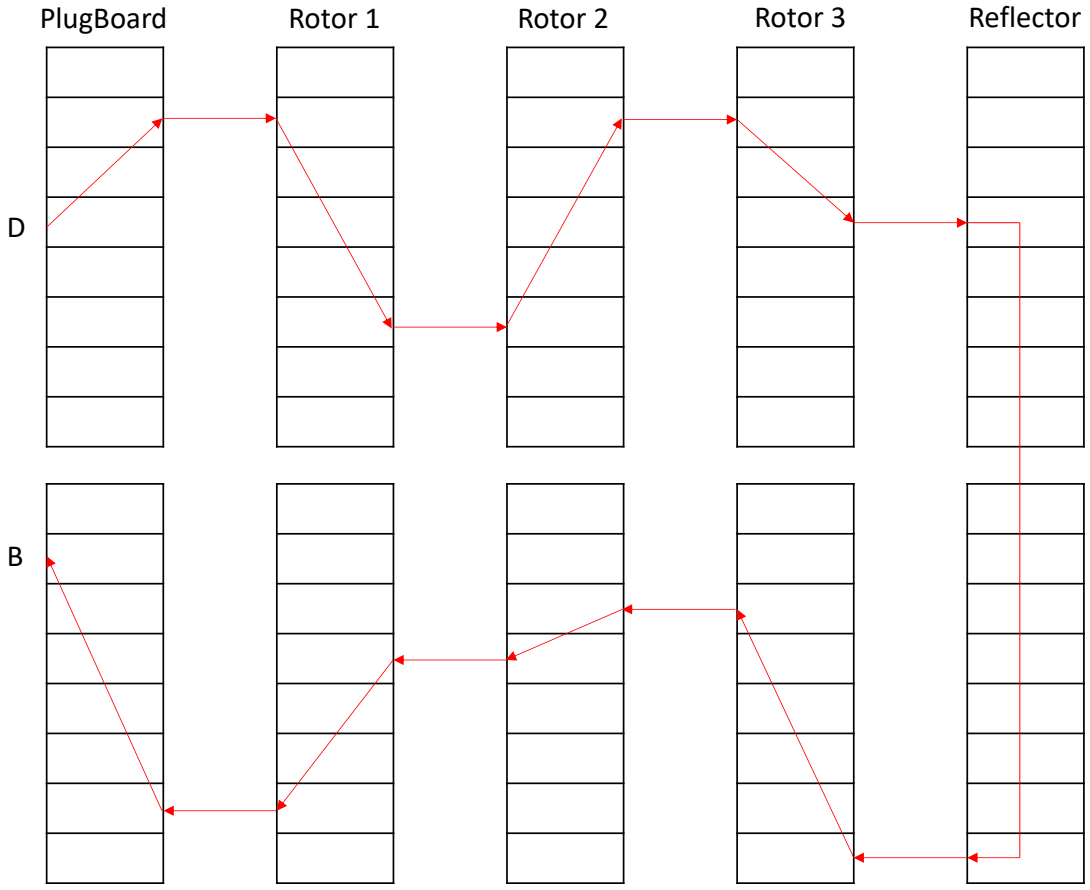

Figure 4: Enigma Machine Character Generation

Rotors rotate according to the clock signal. Rotor1 (fast rotor) rotates one bit at each clock cycle, Rotor2 (normal rotor) rotates one bit for every 26 clock cycles, and Rotor3 (slow rotor) rotates one bit for every 676 clock cycles. Figure 5 shows an example for rotation operations. Green area shows the inner connection for changing order of the

connections. You must use circular shift modules for rotation operations. Circular Right Modules will be used for input connections (forward and backward inputs) and Circular Left Modules will be used for output connections (forward and backward outputs). You must use position counter for the shifting operation (0-25). Also, each rotor can be set to a specific starting position between 0-25 at the rising edge of the load signal. Rotor rotates at the rising edge of the clock signal. Rotor1 and Rotor2 have clock outputs for next rotor module. Clock outputs generate a clock signal in 26 clock signal. In other words, Output Clock signal frequency is (1/26) * Input Clock Signal Frequency.

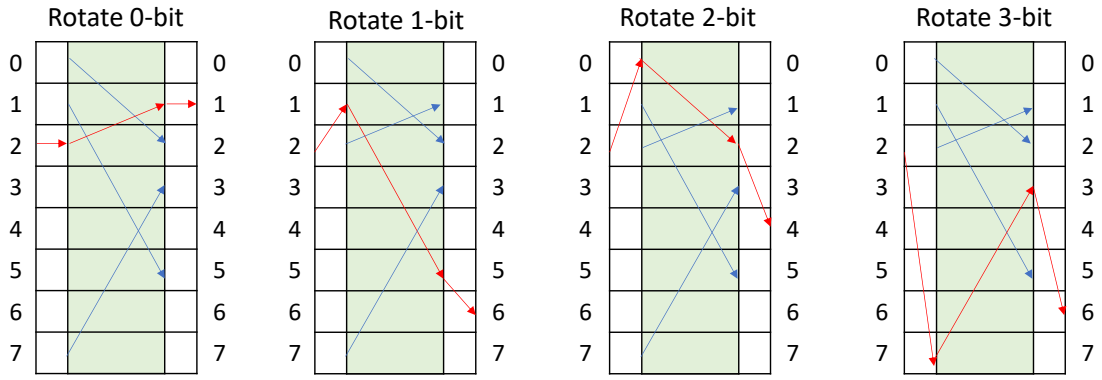

Figure 5: Example Enigma Rotor Rotate Operations

The Enigma machine has 8-bit char input/output, therefore you should use the char decoder/encoder for plugboard connection. After creating the Enigma machine, you will create EnigmaCommunication module which conducts a full encryption/decryption process with 2 Enigma machines. Figure 6 shows the block scheme of the Enigma Communication module.

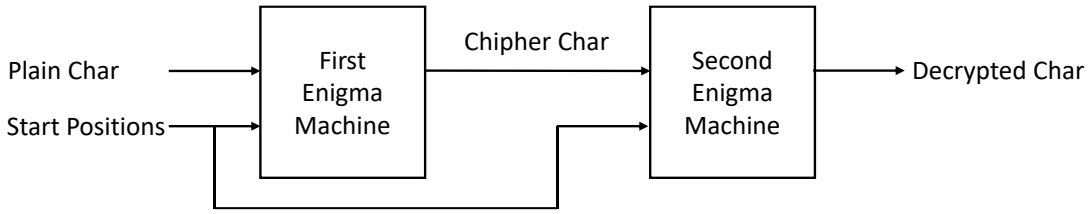

Figure 6: Enigma Communication Module Block Scheme

Detailed information for the Enigma machine's modules is given below.

- **PlugBoard Module: Provide connection between CharDecoder, CharEncoder and Rotor1.** **Table 2** shows the connection of input and output bits.

charInput bits are connected to forwardOutput bits and backwardInput bits are connected to charOutput bits.

Example: backwardInput[6]=charOutput[3], forwardOutput[19]=charInput[15]...

- **charInput (26-bit input)**: Forward direction input of the character.
- **backwardInput (26-bit input)**: Backward direction input of the character.
- **forwardOutput (26-bit output)**: Forward direction output of the character.
- **backwardOutput (26-bit output)**: Backward direction output of the character.

- **Rotor1 Module: Provide connection between Plugboard and Rotor2.** Table 3 shows the inner connection of input and output bits. You also need circular shift modules to generate rotating steps of the rotor. forwardInput bits are connected to forwardOutput bits and backwardInput bits are connectedd to backwardOutput bits.
  - **forwardInput (26-bit input)**: Forward direction input of the character.
  - **backwardInput (26-bit input)**: Backward direction input of the character.
  - **startPosition (5-bit input)**: It is used to determine the starting position of the rotor. It will be loaded to position counter at the positive edge of the load signal.
  - **load (1-bit input)**: It is used to load the starting position data.
  - **clockIn (1-bit input)**: Increase the position counter at the rising edge of the clockIn signal (when load=0).
  - **clockOut (1-bit input)**: Provide clock signal for Rotor2. 26 clockIn signals generate 1 clockOut signal.
  - **forwardOutput (26-bit output)**: Forward direction output of the character.
  - **backwardOutput (26-bit output)**: Backward direction output of the character.

- **Rotor2 Module: Provide connection between Rotor1 and Rotor3.** Table 4 shows the inner connection of input and output bits. You also need circular shift modules to generate rotating steps of the rotor. forwardInput bits are connected to forwardOutput bits and backwardInput bits are connected to backwardOutput bits.
  - **forwardInput (26-bit input)**: Forward direction input of the character.
  - **backwardInput (26-bit input)**: Backward direction input of the character.
  - **startPosition (5-bit input)**: It is used to determine the starting position of the rotor. It will be loaded to position counter at the positive edge of the load signal.

- **load (1-bit input)**: It is used to load the starting position data.
- **clockIn (1-bit input)**: Increase the position counter at the rising edge of the clockIn signal (when load=0).
- **clockOut (1-bit input)**: Provide clock signal for Rotor3. 26 clockIn signals generate 1 clockOut signal.
- **forwardOutput (26-bit output)**: Forward direction output of the character.
- **backwardOutput (26-bit output)**: Backward direction output of the character.

- **Rotor3 Module: Provide connection between Rotor2 and Reflector.** Table 5 shows the inner connection of input and output bits. You also need circular shift modules to generate rotating steps of the rotor. forwardInput bits are connected to forwardOutput bits and backwardInput bits are connected to backwardOutput bits.
  - **forwardInput (26-bit input)**: Forward direction input of the character.
  - **backwardInput (26-bit input)**: Backward direction input of the character.
  - **startPosition (5-bit input)**: It is used to determine the starting position of the rotor. It will be loaded to position counter at the positive edge of the load signal.
  - **load (1-bit input)**: It is used to load the starting position data.
  - **clockIn (1-bit input)**: Increase the position counter at the rising edge of the clockIn signal (when load=0).
  - **forwardOutput (26-bit output)**: Forward direction output of the character.
  - **backwardOutput (26-bit output)**: Backward direction output of the character.

- **Reflector Module: Turn the forward direction of the character to the backward direction.** Table 6 shows the connection of input and output bits. Reflector module connects to the Rotor3.
  - **inputConnection (26-bit input)**: Forward direction input of the character.
  - **outputConnection (26-bit output)**: Backward direction output of the character.

- **EnigmaMachine Module: Encrypt and decrypt the characters.**
  - **char (8-bit input)**: Connect to the charDecoder. Output of the decoder is connected to the plugboard charInput.
  - **startPosition1 (5-bit input)**: StartPosition information for Rotor1.

  - **startPosition2 (5-bit input)**: StartPosition information for Rotor2.
  - **startPosition3 (5-bit input)**: StartPosition information for Rotor3.
  - **load (1-bit input)**: Provides load signal to load start position data to rotors. It is directly connected to the rotors' load input.
  - **clock (1-bit input)**: Provides clock signal to Rotor1. Other clocks signal for Rotor2 and Rotor3 generated inside of the Rotor1 and Rotor2.
  - **outChar (8-bit output)**: Output of the Enigma machine. This data comes from char encoder (connected to charOutput of plugBoard).

- **EnigmaCommunication Module: Generates the communication environment between two Enigma machines. Block scheme of the environment is shown in Figure 6.**
  - **plainChar (8-bit input)**: The character to be transferred securely.
  - **startPosition1 (5-bit input)**: StartPosition information for Rotor1.
  - **startPosition2 (5-bit input)**: StartPosition information for Rotor2.
  - **startPosition3 (5-bit input)**: StartPosition information for Rotor3.
  - **load (1-bit input)**: Provides load signal to rotors.
  - **clock (1-bit input)**: Provides clock signal to Rotor1.
  - **chipherChar (8-bit output)**: Encrypted character which the is output of the first Enigma machine, it will be the input of the second Enigma machine.
  - **decryptedChar (8-bit output)**: Decrypted character which is the output of the second Enigma machine.

Table 2: Plugboard Bit Connections

forwardOutput	**charInput**	**forwardOutput**	**charInput**
backwardInput	**charOutput**	**backwardInput**	**charOutput**
0	4	13	22
1	10	14	24
2	12	15	7
3	5	16	23
4	11	17	20
5	6	18	18
6	3	19	15
7	16	20	0
8	21	21	8
9	25	22	1
10	13	23	17
11	19	24	2
12	14	25	9

Table 3: Rotor1 Bit Connections

forwardOutput	**forwardInput**	**forwardOutput**	**forwardInput**
backwardInput	**backwardOutput**	**backwardInput**	**backwardOutput**
0	7	13	3
1	12	14	14
2	21	15	13
3	17	16	11
4	0	17	8
5	2	18	4
6	22	19	10
7	20	20	6
8	23	21	5
9	18	22	19
10	9	23	16
11	25	24	24
12	15	25	1

Table 4: Rotor2 Bit Connections

forwardOutput	**forwardInput**	**forwardOutput**	**forwardInput**
backwardInput	**backwardOutput**	**backwardInput**	**backwardOutput**
0	19	13	9
1	4	14	14
2	7	15	22
3	6	16	24
4	12	17	18
5	17	18	15
6	8	19	13
7	5	20	3
8	2	21	10
9	0	22	21
10	1	23	16
11	20	24	11
12	25	25	23

Table 5: Rotor3 Bit Connections

forwardOutput	**forwardInput**	**forwardOutput**	**forwardInput**
backwardInput	**backwardOutput**	**backwardInput**	**backwardOutput**
0	19	13	13
1	0	14	25
2	6	15	7
3	1	16	24
4	15	17	8
5	2	18	23
6	18	19	9
7	3	20	22
8	16	21	11
9	4	22	17
10	20	23	10
11	5	24	14
12	21	25	12

Table 6: Reflector Bit Connections

outputConnection	inputConnection	outputConnection	inputConnection
0	24	13	10
1	17	14	12
2	20	15	8
3	7	16	4
4	16	17	1
5	18	18	5
6	11	19	25
7	3	20	2
8	15	21	22
9	23	22	21
10	13	23	9
11	6	24	0
12	14	25	19

6 Report

- You can use any **software tool** for your circuit designs. You may attach them to the report as figures by properly referencing them in the text.
- Please use the table attributes of Latex. You can check out online Latex table generators (for example: https://www.tablesgenerator.com).
- Your report should contain information about the results of your simulations. If your implementations are not fully correct, discuss what the source of the errors might be in your report.
- For further details about the report, please check Ninova e-learning system.

7 Submission

- Please add comments to the code to clarify your intends.
- Please don't send any document via e-mail to one of the assistants.
- Your reports must be written in Latex format. Latex report template is available on Ninova. You can use any Latex editor of your choice. If you upload your report without Latex file, you directly get 0 as your report grade. You should upload both .pdf and .tex files of your report.
- You should submit 2 separate ".v" files for your Verilog codes: one containing the modules, one containing the simulation codes.

- Please make sure that you have written your full name and student ID number to every document you are submitting.
- Please only write the names of members who have contributed to the experiment.
- Note that late submissions are not accepted, be aware of the deadline.
- Please do not hesitate to contact me (aydinesi16@itu.edu.tr)
  Don't forget to have fun and stay healthy.

References

[1] Wikipedia contributors, "Caesar cipher — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Caesar_cipher&oldid=1019328856, 2021. [Online; accessed 14-May-2021]. 1

[2] G. J. Simmons, "Vigenère cipher. Encyclopedia Britannica.." https://www.britannica.com/topic/Vigenere-cipher, 2017. [Online; accessed 14-May-2021]. 1

[3] B. Copeland, ""Ultra". Encyclopedia Britannica.." https://www.britannica.com/topic/Ultra-Allied-intelligence-project, 2019. [Online; accessed 14-May-2021]. 1