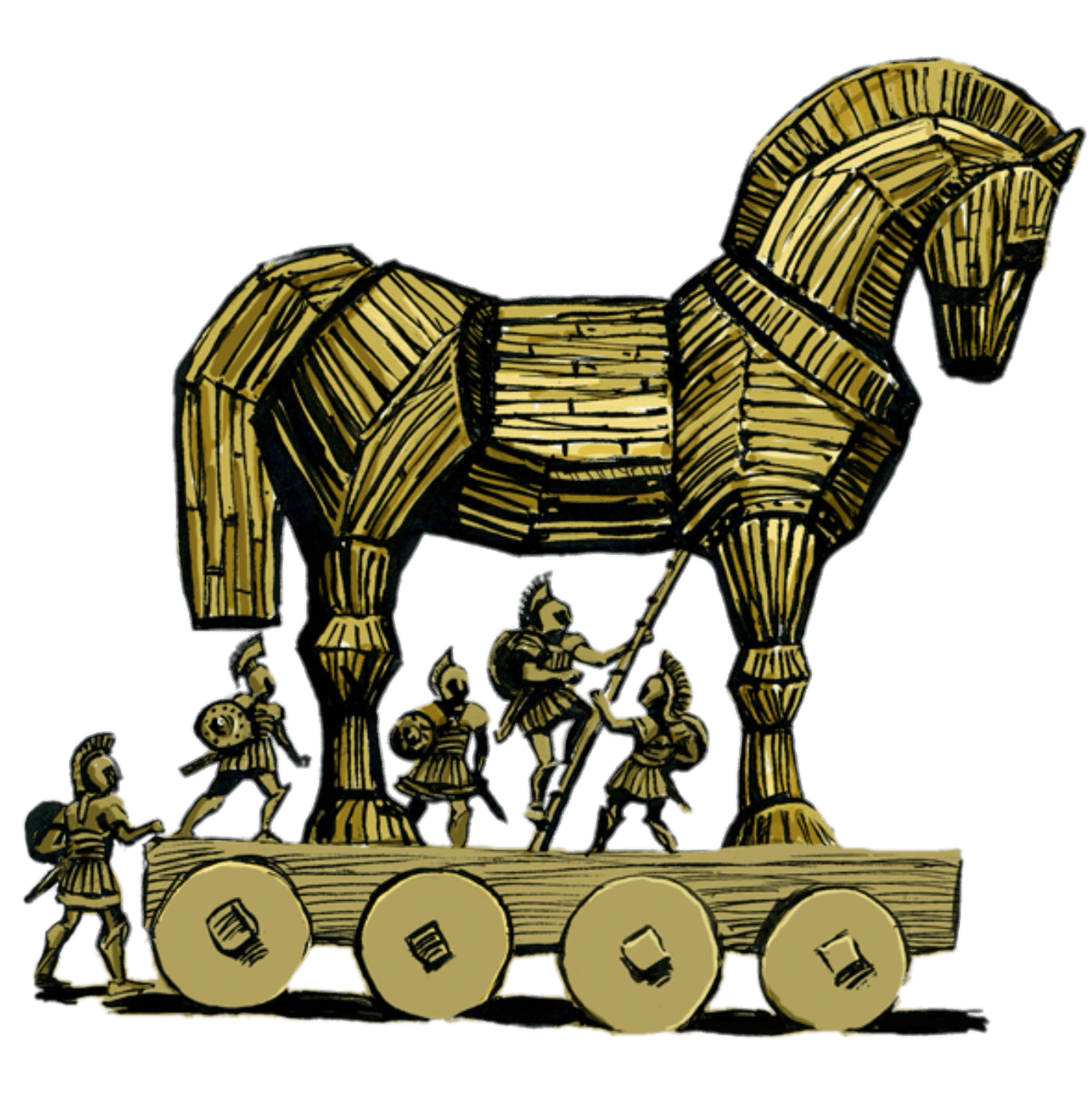
**HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH-QUALITY TRAINING**





**COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE**



**Lecturer: Nguyen Dang Quang**

**Topic: Cipher & Decipher in C and Assembly language**

*HCMC June 15, 2020  
Last update: June 16, 2020*

# STUDENT LIST

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **Role** | **EMAIL** | **STUDENT’S ID** |
| Trịnh Minh Anh | Team Leader | peterburgs.vn@gmail.com | 17110003 |
| Lê Đức Thịnh | Member | thinhle2199@gmail.com | 17110076 |

# EVALUATION & SCORE

**Overall score:**

# PREFACE

The present report is the outcome of the courses Information Security, Computer Architecture & Assembly Language. It is our pleasure to share knowledge and experience with everyone. The objectives of this project are to create an algorithm in which we could apply the knowledge to process data in a text, and then cipher/decipher it.

With the topic for the final project " **Cipher & Decipher in C and Assembly language,** " we would like to explain all the required fields as simple as possible. Despite the language barrier, we would try our best to make things clear.

Note that nothing is entirely perfect, this report might have some mistakes, and we would appreciate all the evaluation and feedback.

# TABLE OF CONTENTS

[STUDENT LIST i](#_Toc43191828)

[EVALUATION & SCORE ii](#_Toc43191829)

[PREFACE iii](#_Toc43191830)

[TABLE OF CONTENTS 1](#_Toc43191831)

[IMAGE LIST 2](#_Toc43191832)

[TABLE LIST 3](#_Toc43191833)

[CONTENT 4](#_Toc43191834)

[I. PROJECT OVERVIEW 4](#_Toc43191835)

[A. DEFINITION 4](#_Toc43191836)

[B. SYSTEM CONFIGURATION 4](#_Toc43191837)

[II. TASK DISTRIBUTION 5](#_Toc43191838)

[III. IMPLEMENTATION 7](#_Toc43191839)

[1. CIPHERING 7](#_Toc43191840)

[1.1. Stack frame layout 7](#_Toc43191841)

[BIBLIOGRAPHY 8](#_Toc43191842)

# IMAGE LIST

*All images present in this report are listed here.*

[Figure 1. Stack frame 7](file:///C:\Users\stare\Desktop\Cypher_Report.docx#_Toc43192151)

# TABLE LIST

*All tables present in this report are listed here.*

[Table 1. Task distribution 5](#_Toc43192332)

[Table 2. Project distribution by member 6](#_Toc43192333)

# CONTENT

*This section contains all content of the report.*

# I. PROJECT OVERVIEW

*This chapter is an overview of the entire project; what you need to mind most here is the definition, which illustrates the purposes of the application.*

## A. DEFINITION

We call our solution is “**Cypher**”. Cypher has 2 main features:

* Cipher: receive the input as a text file and a key file from the user to right rotate by a certain number of bits. The output is in hexadecimal.
* Decipher: receive the output of Cipher function then left rotate to recover the initial data.

The encoding scheme is explained below:

16-bit word

16-bit word

16-bit word

Plaintext (16)

ROR

ROR

ROR

Rotating keys (8)

a

b word

c

8 keys word

The decoding scheme is contrary to the encoding scheme.

## B. SYSTEM CONFIGURATION

In the development period, we push the project into GitHub so that everyone could access and program remotely. After that, in the testing period, we carry out the project and run on only one computer to receive solid results.

* Memory: 7.7 GiB.
* Processor: Intel® Core™ i5-8300H CPU @ 2.30GHz × 8.
* OS: Ubuntu.
* GNOME: 3.34.1.
* OS type: 64-bit.
* OS version: 19.10.
* Virtualization: VMware.
* Text editor: Visual Studio Code.

# II. TASK DISTRIBUTION

*This chapter illustrates the tasks for each member.*

Table 1. Task distribution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | Due date | Taken by | Requirement/purpose | % Done | Result |
| Research for documentations and tutorials. | May 19 | All | Clarify the vision, technical requirements, and other factors. | 100% | - Definition  - Reasons to choose this topic  - Input and output data  - Main features of the application |
| Cipher plain text. | May 21 | Thinh | Input: plain text, key.  Output: ciphered text. | 100% | - cipher.c: C program to call external library, read input, write output.  - encode.asm: assembly program to cipher.  -encodedText.txt: include encoded text. |
| Decipher ciphered text. | May 21 | Anh | Input: ciphered text, key.  Output: original text. | 100% | - decipher.c: C program to call external library, read input, write output.  - decode.asm: assembly program to decipher.  newDummy.txt: contain the text deciphered from encoded text. |
| Merge program into consistent workflow | May 21 | Thinh | Input: cipher & decipher program.  Output: working program | 100% | The program works correctly according to vision of the topic. |
| Testing | May 21 | Anh | Test as much cases as possible.  Must have one test case includes large text file. | 100% | Find minor bugs.  Large text file (>1.000.000 line of text) run too slow. |
| Optimize code | May 21 | Thinh | Add comment, remove redundant code. | 100% | Program run faster (at least 10% faster than before). |
| Add timer | May 21 | Anh | Timer must be precise up to 1/1.000 | 100% | Timer can be precise up to 1/1.000.000 |
| Manage repository | May 21 | Thinh | Create, commit, and update changes | 100% | Project on GitHub has been managed. |
| Write report | June 15 | Anh | Write a comprehensive report based on requirements given. | 100% | Final report. |

Overall, we have a final meeting to determine the percentages that each member has distributed to the project.

Table 2. Project distribution by member

|  |  |  |  |
| --- | --- | --- | --- |
| # | Student Name | Main tasks | % Distribution |
| 1 | Trịnh Minh Anh | Decipher.  Idea to solve the problem.  Testing.  Optimize code.  Report. | 50% |
| 2 | Lê Đức Thịnh | Cipher.  Merge 2 program.  Optimize code.  Implement idea and make algorithm.  Manage repository. | 50% |

# III. IMPLEMENTATION

*This chapter describes how we implement the application.*

## CIPHERING

The idea of the “Cypher” application is that we take the plain text and 8 keys from two files .txt using the C programming language. After having the plain text and the keys, we push them through a function called “encode”. “Encode” function is written in assembly which returns an encoded string. In other to write a function in assembly, we need to recall the stack frame knowledge from the course.

### 1.1. Stack frame layout and access argument addresses

The figure beneath shows a simple layout of the stack frame of a function. The stack grows from a high memory address to a low memory address. When a function is called, a stack frame will be created. The argument on the rightmost of the function will be pushed to the stack first so that it is laid at the highest memory address. After pushing all arguments to the stack, the program will push the return address.

Figure 1. Stack frame

[%esp]

%ebp

Ret address

Arg 1

Arg 2

High mem

Low mem

[%esp+12]

[%esp+8]

The function needs to be able to access the argument value and then processes that value, which requires access to the argument addresses. The argument addresses can only be accessed by a register called ***ebp***. As you can see in figure 1, the ebp register is placed right below the return address so that when the function wants to access the arg 1 it just needs to plus 8 to the ebp (each element in the stack is 4 bytes). **The only responsibility of the function is to push the ebp right below the return address.**

encode:

push ebp

mov ebp, esp

### 1.2. Strategy

We will use the top-down approach to explain the strategy of the application, which means we will explain what we did in the C program first and then move to the assembly function.

As we explained above, the C program is responsible for reading the keys so that we will use an object with the data type FILE in stdio.h library to read text from a file called key.txt.

//Read Keys from file

FILE \*reader;

reader = fopen("key.txt", "r");

char keys[9];

while (!feof(reader))

{

fgets(keys, 9, reader);

}

fclose(reader);

The keys value will be stored in the character array called keys.

The next step is reading the plain text from a file called dummy.txt. To avoid lagging, we just read each line of the file, encode it and write it to a file called encodedText.txt.

while (!feof(reader))

{

fgets(pltext, 500, reader);

fflush(stdin);

STR\_LEN = StringLength(pltext);

unsigned char \*result = "";

//Padding last line

if (feof(reader))

{

pltext[StringLength(pltext)] = ' ';

STR\_LEN++;

}

result = encode(pltext, STR\_LEN, keyPtr);

FILE \*reader1 = fopen("encodedText.txt", "ab+");

for (int i = 0; i < STR\_LEN; i++)

{

fprintf(reader1, "%02x", \*(result + i));

}

if (pltext[0] != '\0' && !feof(reader))

{

fprintf(reader1, "\n");

}

fclose(reader1);

}

The program loops through the file until it reaches the ‘\0’ character, which means the end of file (eof). The encode function takes three parameters. The first one is pltext which is a pointer pointing to the first character of the line, the second one is STR\_LEN storing the length of the line and the last one is keyPtr which is responsible for pointing to the first key in keys array.

After processing the pltText with the keys, the encode function returns a string including hex characters and it is stored in the result variable.

Now let’s move the assembly function – “encode”.

section .data

keyIndex db 0

rotatingCount db 0

section .bss

resultArr resb 3000

section .text

global encode

encode:

push ebp

mov ebp, esp

Let’s take a look at the data section and bss section, we need keyIndex with the data type **byte**, to store the current index of the keys array argument, rotatingCount with the data type **byte,** to store the number of bits will be rotated. And of course, we need an array (resultArr) to store the encoded string and return it at the end of the function.

After pushing the ebp, we implement a loop to iterate through the pltText. The data block is 16-bit which means 2 characters will be encoded with a specific key value at a specific index. In each iteration, the key index will be increased by 1 and because the maximum value of the key index is 7, the key index will reset to 0 of it greater than 7.

; word index

xor esi, esi

; key index

xor edi, edi

\_loop:

; Get key value

mov eax, [ebp + 16]

mov cl, byte[eax + edi]

; Modulo key value to 16

xor eax, eax

mov al, cl

mov cl, 16

div cl

mov cl, ah

; Get 16-bits plain text

mov eax, [ebp + 8]

mov dx, word[eax + esi]

; Rotate right

ror dx, cl

; Copy value to new string

mov word[resultArr + esi], dx

inc edi

; Reset key index to 0 if it > 7

cmp edi, 8

jl \_skip\_the\_reset

xor edi, edi

\_skip\_the\_reset:

add esi, 2

cmp esi, [ebp + 12]

jl \_loop

# BIBLIOGRAPHY

*While implementing this project, we have referenced some documentation and get inspired of some website on the Internet.*

1. Divega. "Get Started with Entity Framework 6 - EF6." Microsoft.Com, 23 Oct. 2016, docs.microsoft.com/en-us/ef/ef6/get-started. Accessed 25 Nov. 2019.
2. Rick-Anderson. "Create a Web App with ASP.NET Core MVC." Microsoft.Com, 26 Oct. 2017, docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-3.0. Accessed 25 Nov. 2019.
3. JS Foundation - js.foundation. “JQuery API Documentation.” Jquery.Com, 2019, api.jquery.com/.
4. “SQL Tutorial.” *W3schools.Com*, 2019, www.w3schools.com/sql/.
5. ‌ "Using CSS Animations." MDN Web Docs, 18 Nov. 2019, developer.mozilla.org/en-US/docs/Web/CSS/CSS\_Animations/Using\_CSS\_animations.