University of Jordan

Embedded Systems Lab

Summer 2018/2019

Project: Secure Communication using PICRYPT Algorithm

Preferred Group Size	Grading	Project Due Date
(4) Four is the maximum allowed group size.	**(20)** Twenty marks is the project weight.	**(31st)** The deadline for the project is the week of July 31st, 2019.

Project Description

In this project you are required to implement a simple base-to-unit secure communication system. There will be two PIC16 microcontrollers: one represents the base station that transmits and logs commands, the other is a unit station that receives and displays the commands.

The first microcontroller will have a set of three messages pre-programmed in the device EEPROM. The user selects which message to send by pressing one of three push buttons where each is assigned for a message. The microcontroller reads the message in blocks of four bytes (32 bytes) from the EEPROM and applies a 32-bit key encryption before transmitting the message to the unit station. The base station keeps track of how many times each message has been selected. The user can request to see how many times each message has been transmitted on a set of 7-segment displays. The user can do so using two push buttons. The first button is used to go through the message IDs (1, 2, 3, or OFF). The second button is used as an ENTER key to display the log or turn all 7-segments off. Each display should show the message ID (number) and a two-digit counter for the number of transmissions. The details of the encryption algorithm will be provided later.

On the other side, a microcontroller receives and stores the message in a group of four bytes. Then whenever a message is received, the user is alerted that a message is ready to be decrypted (i.e. LED). The user should manually press a button for the microcontroller to decrypt the message and display its contents on an LCD screen.

Project Details

In this project, you will be using hardware timers for 7-segment multiplexing delay, an LCD, and the USART which will be introduced to you as we go on. Most of the codes that we will be using during the experiments can be reused in the project. The focus, however, will be more on the software components and specifically on the Encryption/Decryption Algorithm. Remember that the PIC we used in the lab is an 8-bit microcontroller, yet we are going to simulate some simple 32-bit operations.

Initially, we restrict the message size to **four bytes** and the messages that can be selected for transmission are:

- ATTK which denotes an attack order.
- RTRT which denotes a retreat order.
- MRCH which denotes a march order.

These messages are stored inside the microcontroller EEPROM. Even though we have not covered the EEPROM in the lab course, we expect you to learn how to use it as it is quite

straightforward. To initialize the EEPROM with the above messages, we refer you to use the "de" directive. Details on EEPROM initialization can be found in the MPASM assembler documentation. You can find the documentation in MPLAB by going to:

Help → Topics → MPASM Assembler → OK. Then you can find the "de" directive under the **Directives** topic where you can find some examples.

Even though for the base project the message size is fixed, if you are going to implement the bonus part, there are two ways to actually denote the end of a variable-size message. You can either terminate the message with a null (number 0 in this case) which works well for terminating ASCII messages but not numbers stored in the EEPROM. The other method is to start the message by a number which denotes its length in bytes. Remember to consult the datasheet for the maximum size of the EEPROM in your device. Reading and writing the EEPROM is easy. Consult Section 3 of the PIC16F877A datasheet. There are two example codes for reading and writing the data EEPROM which you can implement in your project. However, my advice is to modify the code supplied in the datasheet to use the "banksel" directive instead of the hardcoded switching. Also remember to switch back to the bank you have been using before using the EEPROM code.

PICRYPT

In this project, you are going to use a 32-bit symmetric key encryption/decryption algorithm that we came up with for this project. In symmetric key cryptography, the algorithm uses the SAME key for encrypting and decrypting the message. In real-life, cryptography standards use 128-bit, 256-bit, 512-bit and more (e.g. 4K-bits) as s key size. The larger the key size, the more complex it is to crack the key using brute force search techniques. For example, an 8-bit key means that it only takes 256 attempts to guess what the key is. In this project, the PICRYPT algorithm uses 32-bit keys due to memory limitations and only to serve as an example for cryptographic operations. Kindly note, our PICRYPT algorithm is just an algorithm for educational purposes and is not to be deployed in real systems. Our algorithm is based on similar operations that are used in many current encryption algorithms. These include xor-ciphering, bit-rotations, the use of multiple keys, prime numbers, and multiple rounds of encryption.

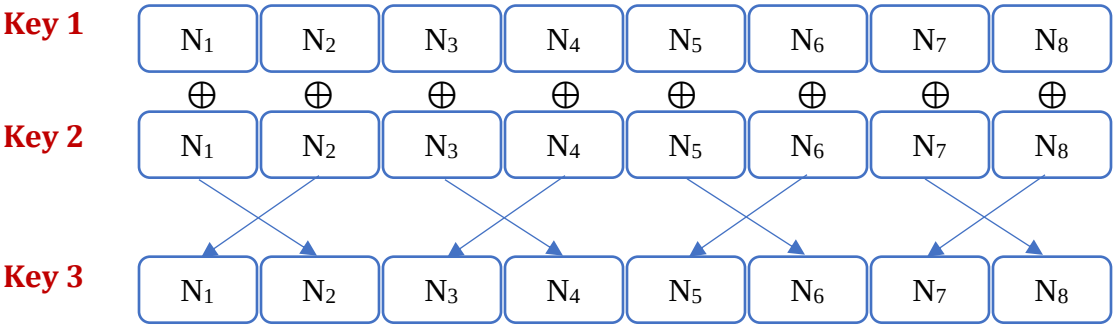
The PICRYPT algorithm uses **four keys** in total. Key_1 and Key_2 are two 32-bit prime numbers that are initially stored in the EEPROM. In your project, you can use any 32-bit prime numbers you want, and we might ask you to initialize these keys by our supplied numbers during the project discussion. You can start with these two keys in the beginning: ***0xd2a5664f*** and ***0xe0b15add***. Remember that these are 32-bit number that will not fit in one PIC register or memory location. You must reserve four bytes of memory or four registers to handle them. In order to avoid confusion, we are going to use this notation throughout this project when using 32-bit numbers: The Most Significant Byte is denoted by Byte 1 and the Least Significant Byte as Byte 4. So, when you want to give your variables

names to store these 32-bit keys, kindly use this notation. For example, Key_1_1 means the most significant byte of the first key. Key_2_4 means the least significant byte of the second key. Remember that each byte is composed from two nibbles. Basically, the notation is Key_x_y where x is the key number and y is the byte order or Key_x_Ny where in this case Ny is the nibble number as follows:

Byte 1		Byte 2		Byte 3		Byte 4	
N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8

The third and fourth keys are generated **in the code** from the first and second keys. The third key is generated as follows:

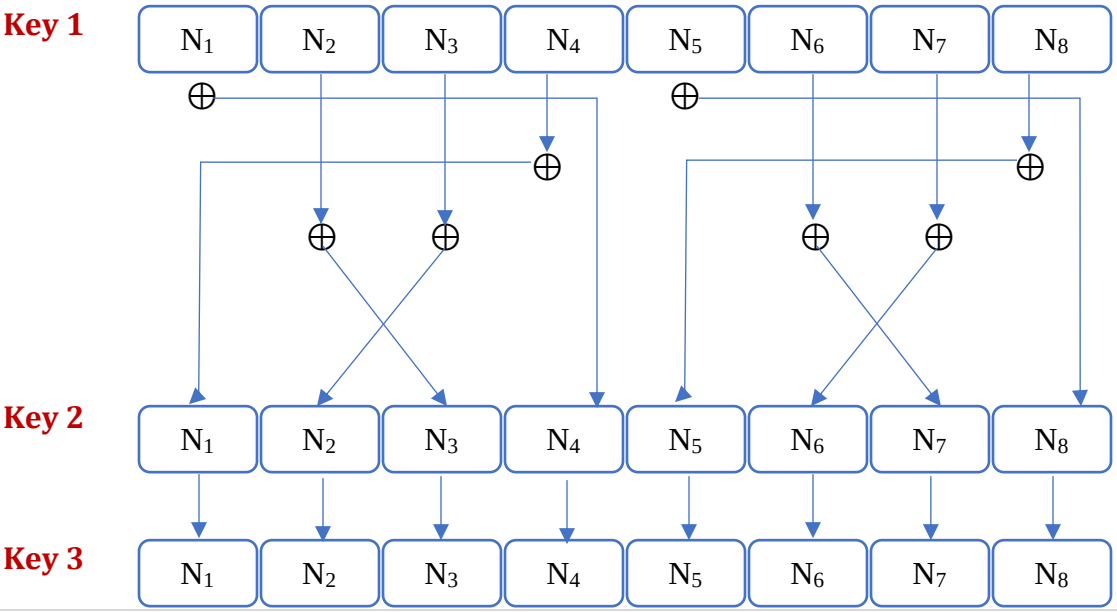
Most Significant Byte (Nibble) → Least Significant Byte (Nibble)



So basically, the first key and the second key are XORed with each other. The nibbles are swapped in each byte and stored in the third key.

Examples: $Key\_3\_N_1 = Key\_1\_N_2 \oplus Key\_2\_N_2$ $Key\_3\_N_2 = Key\_1\_N_1 \oplus Key\_2\_N_1$

The generation of key four is a bit more involved from a programming perspective:



Examples: $Key\_3\_N_1 = Key\_1\_N_4 \oplus Key\_2\_N_1$ $Key\_3\_N_2 = Key\_1\_N_3 \oplus Key\_2\_N_2$

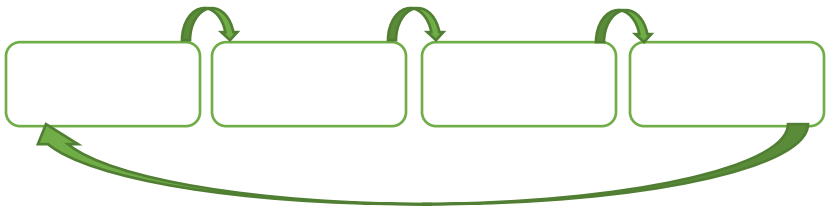
After the third and fourth keys are derived and stored, and the selected message is read from the EEPROM and stored in temporary variables, the PICRYPT Algorithm executes as follows:

PICRYPT Algorithm	
Encryption	**Decryption**
1. While (No_Rounds != 0) 2. { 3. R = Msg ⊕ Key_1 4. R = Rotate R to the left by 1 5. R = R ⊕ Key_2 6. R = Rotate R to the left by 1 7. R = R ⊕ Key_3 8. R = Rotate R to the left by 1 9. R = R ⊕ Key_4 10. R = Rotate R to the left by 1 11. No_Rounds = No_Rounds - 1 12.}	1. While (No_Rounds != 0) 2. { 3. R = Rotate Msg to the right by 1 4. R = R ⊕ Key_4 5. R = Rotate R to the right by 1 6. R = R ⊕ Key_3 7. R = Rotate R to the right by 1 8. R = R ⊕ Key_2 9. R = Rotate R to the right by 1 10. R = R ⊕ Key_1 11. No_Rounds = No_Rounds - 1 12.}
*Msg and R are simulated 32-bit numbers on PIC16Fxxx	

Notice that the Decryption algorithm is in the reverse order of operations of the encryption algorithm.

For this project, the number of encryption and decryption rounds is set to 5

Hint: Remember that we are dealing with the keys and operations on simulated 32-bit numbers, so you have to make sure that when you use the rotate operations, that bits from one byte are rotated into the other bytes as follows (Rotate right example):



Bonus (2 Marks)

Instead of having a message of size 4, in this bonus you are going to implement variable size messages (e.g. Attack, Retreat, March). You can either terminate these strings by 0 in the EEPROM or specify the length of the message at the beginning. You will read, encrypt, and

transmit the message in blocks of four. If the remaining message size is less than four, then you must pad the message by zeros (missing bytes are initialized by zeros). You must take care of all related issues to the number of blocks transmitted, their display, and solve any requirements that may arise such that the project works as specified before. However, if you implement variable-length messages, on the receiver end you can decrypt and display the messages automatically without the need for user intervention as before.

GRADING FOR CODE EFFICIENCY AND GENERALITY

It is of utmost importance to write codes which are minimum in size and execute quickly. We ask you to use subroutines, modular design and functional reuse whenever possible. In some cases, the use of indirect addressing (FSR and INDF) can be helpful in reducing code size and increasing speed. Your code must NOT be hard-coded to only work with the codes provided. It should be generic, work with any set of keys and any message. It is important to not forget to use functional comments.

During the discussion, we might ask you to show us on MPLAB how your code works with a different set of keys so be prepared for the generic case. Also, we might ask you to see the total program or certain subroutine size or timing. Marks can be assigned as part of a competition where the highest marks are given to whom has the most efficient code. For example, we might ask you to measure the time for the PICRYPT algorithm/encryption part. Suppose we assign 2 marks for this part, then the group with the fastest subroutine will get full mark, the second highest 75% of the grade and so on. If two groups have suspiciously similar codes, timing, or size, both groups will get a zero.

Signup Sheet

The project can be done by two students within the allotted time frame. However, you can form groups up to four students. Kindly divide yourself into groups and signup your names in the following Google sheet Embedded Lab Summer Groups before July 8th, 2019.

Important Notes

- Start as early as possible on your project, though the project description sounds simple, there is inherent complexity in both hardware and software aspects, so do not underestimate the time it needs, you will have many problems along the way which you will have to resolve!

- Never think of buying a model or commissioning someone to do it for you, not only will you get a zero in the project, but also your act will be considered as a direct violation to JU laws and your actions shall be reported as cheating in the final exam!

- **Code sharing between groups is NOT allowed and leads to 0 points.**

- If you acquire a ***part*** of your software from a book, website, etc … kindly reference it properly, else it will be considered as plagiarism.

- You are only allowed to base your project on PIC16877a or PIC1684a.

- All programming must be done in **PIC ASSEMBLY** language only; using high level languages in the project will get you a Zero.

- Your submitted work must be professional:
  - Hardware: you are submitting a product, all electrical and electronic components must be hidden from the user, only user-accessed components are visible, hide the wiring, be neat. ***Still, the instructor should be easily able to examine the internal components at the time of discussion when required!***
  - Software: your work should be fully documented, all inputs/outputs should be listed, and each subroutine/macro should be fully documented! Use functional comments! Refer to the last section in experiment 2 regarding documentation.

- You should submit two types of flowcharts:
  1. An abstract general flowchart of the whole program.
  2. A flowchart for each of your subroutines/macros (except codes taken from lab experiments).

- Students are not allowed to move between groups once they are formed, so choose your group carefully from the beginning! ***We are not responsible if your colleagues in the group chose to drop the class, we will not allow you to join another group!***

- Divide the work such that each student is responsible for a specific task, **YET EVERY** student is required to answer for **ANY QUESTIONS** in relation to any submitted work of the project.

Report Guidelines

You should submit a hard copy of your report and it should contain the following parts:

- Introduction
- Requirements and subsystems description along with their respective flowcharts (as described above in the notes section)

- Circuit diagram of your hardware (use Proteus to draw the circuit)
- Snap shots of the actual hardware implementation with a brief description
- The contribution of each student in the project
- Major obstacles faced during the design process

Good Luck and Have Fun Building the Project