ECSE 444 Group 14 Final Report

Kai Fan Zheng
260962377
McGill University
Montreal, Canada
kaifan.zheng@mail.mcgill.ca

Niilo Vuokila
260926706
McGill University
Montreal, Canada
niilo.vuokila@mail.mcgill.ca

Noshin Saiyara Chowdhury
260971544
McGill University
Montreal, Canada
noshin.chowdhury2@mail.mcgill.ca

Sarah Ajji
260925797
McGill University
Montreal, Canada
sarah.ajji@mail.mcgill.ca

I. Overview of the System

The design problem for this project was to implement a secure two-way communication system, for personal or confidential data to be sent. Many encryption methods already exist, some being more secure than others. For this project, we chose the tried and true RSA encryption method that relies on prime numbers for its encryption mechanism.

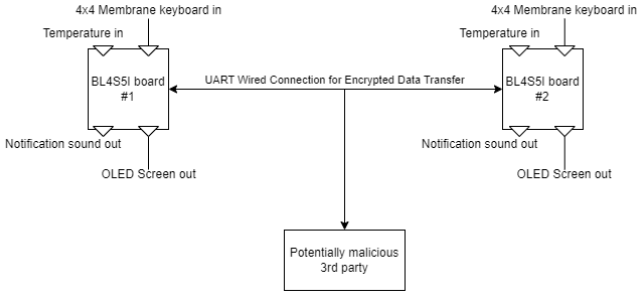

Figure 1. Diagram overview of the system

This project implements 2-way communication over UART, encrypted by RSA. The system consists of 2 interconnected boards with an optional third board to demonstrate encryption and the inability of a third party to intervene or read 2-way communications.

The boards are configured to play a ringtone when a message is received. To write a message, each board has a 4x4 membrane keyboard that can be used to provide input. To display, an OLED screen is used with a simple UI system. The boards also have the capability of reading temperature and humidity data from the built in HTS221 sensor.

II. Description of Design Approach

A. RSA Cryptosystem

The RSA Cryptosystem encrypts messages for communication using a set of public and private keys. Only the receiver knows the private key, but the public key is visible to everyone, using which anyone can encrypt a message to send to the receiver, but since no one but the receiver knows the private key, only the receiver can decrypt the message, thus making it secure from intruders. The system works with the mathematical concept that modular arithmetic is difficult to inverse. To form the private key, we can use two prime numbers. The larger the primes, the more secure the system is since it is more difficult to reverse the mod of a larger number. The algorithm is summarized in the following diagram.

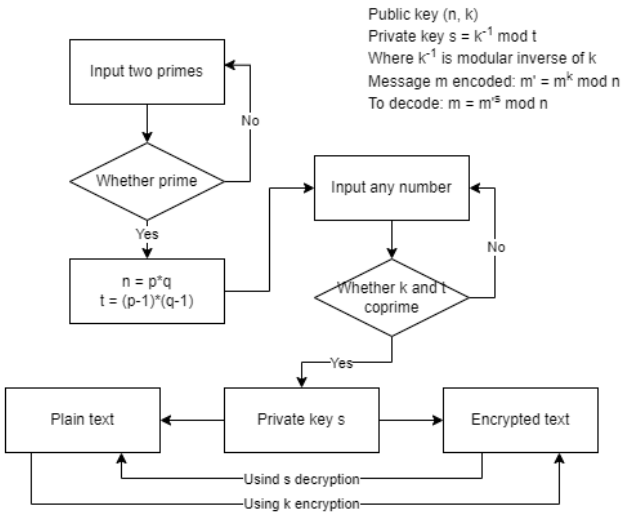

Figure 2. Flowchart with explanation for the RSA cryptosystem

B. UART Communication Driver

To communicate between two or more boards, we are using serial communication with wires through UART ports on the STM-32 boards. We decided on using UART after our extensive research and attempts on using Wi-Fi for communication proved to be unsuccessful. We use UART protocols and driver functions from HAL library to program the transmission of data by utilizing the RX and TX pins which are for receiving and transmitting information respectively. We configure the pins in MX using the same baud rate of 9600bit/s for all boards in the network. During transmission and reception UART converts raw data into data packets, which during integration of the Cryptosystem we discovered to be 8-bits. We were unable to transmit encrypted messages since the encrypted value can be much larger than the original message, giving larger than 8-bit values for our test messages. To solve this, we converted the numbers to a string which is essentially a char array, to be transmitted. We separated each number with a '.'as a separator, which upon reception by the receiving board can be split into the different numbers as strings and converted to integers to be decrypted.

C. Temperature and humidity sensor functionality

The temperature and humidity sensor (HTS221) also remained unchanged from the initial demo and report. The BL4S5I board support package is used to access the sensor and read the temperature and humidity data. In order to access this

data as the user, one board sends 'B' for temperature or 'C' for humidity to another board that's been paired. In return, the other board sends back the temperature and humidity values. This design choice was made so that the temperature data is logically available to the user. For instance, if the two boards are reasonably far away from each other, a user of the system can see temperature data from another location without physically moving. Otherwise, there was not much design involved with this part of the project. The sensor works as expected and outputs the temperature and the output is shown correctly when requested by a user.

D. Speaker Output (ringtone)

The ringtone remained the same throughout the project, simply consisting of 4 tones and their harmonics. The entire ringtone is generated when the board starts up and is stored in memory. Since the array is of length 22932 with 8 bits per sample, the array can be safely stored in memory without running into allocation issues for other components. The ringtone being stored in memory also allows the DAC to output the waveform through DMA, driven by a 44.1kHz timer, without talking up many CPU resources. This was done because we knew we had the capacity for memory to store the ringtone, and also wanted the system to be as responsive as possible since user input/output is a big part of the project. However, an alternate, even less impactful method of saving the ringtone is to store it in code as a predefined array. The reason this is not done in our project is because we wanted to retain a dynamic generation process, so that the ringtone would be easy to change later. Also, doing this would negatively impact the readability of our code.

In terms of results, the connected piezoelectric buzzer works as expected and plays back the four notes in the correct order at 44.1kHz. Performance impact apart from generating the ringtone at the start is minimal.

E. 4x4 Membrane Keyboard Functionality

The membrane keyboard works based on polling. This was not so much a design decision as it was a necessity. The keyboard works by detecting a change in one of the row outputs, and the column is checked by asserting a signal and seeing if the response is active or not. With this, the keyboard is not very well suited for interrupts, as determining the column is inherently a polling-like action. Each key is mapped as follows:

1	**2**	**3**	**A**
4	**5**	**6**	**B**
7	**8**	**9**	**C**
*****	**0**	**#**	**D**

Table 1. Membrane keyboard input mapping

Due to the layout, the messages to be sent are not very suitable for normal text but focused on sending number sequences. If we wanted the limited size keyboard to be more dynamic, we could have implemented a system similar to old cellphones, shown below:

1	**2 abc**	**3 def**	**A**
4 ghi	**5 jkl**	**6 mno**	**B**
7 pqrs	**8 tuv**	**9 wxyz**	**C**
*** +**	**0 _**	**#**	**D**

Table 2. Possible keyboard layout

However, since this would have made our implementation much more complicated, we decided to opt for a simple layout that could demonstrate the functionality. For a fully functioning messaging system, the alternate layout would be more suitable, but would also require more testing and development time to get working smoothly, as repeatedly pressing button to get a different input is quite demanding for getting the timing correct.

F. OLED Screen Output

The OLED screen is driven by a library created by N. Mohideen [1]. This library allows the program to print characters in varying sizes onto the screen. With this driver, we were able to create a menu on the screen where the user can choose the functionality. For example, the user can view a history of messages received, send a message or pair to another device. The complete flow of the UI is demonstrated below in Figure 3.

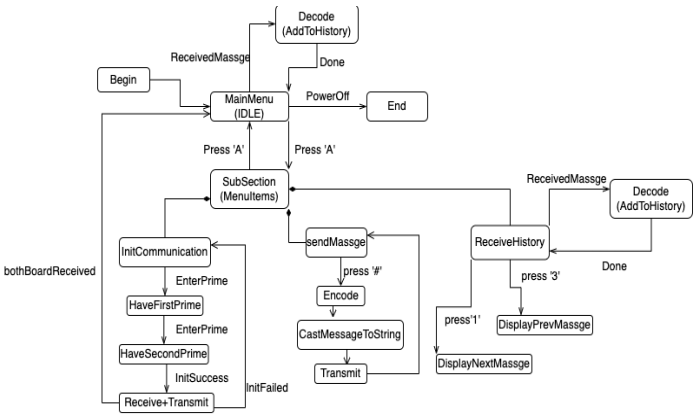

Figure 3. Complete UI flow

This system was chosen because it makes it much easier to choose between the functionalities of the board. Without a screen as output, it would be difficult to write a message and to know what was received. An alternate approach would be to use some form of audio interface to say messages aloud. The screen was opted for as it is easier to use and a more conventional and thus familiar interface to most people. Also, for demonstrations, a screen makes the project's function clearer.

III. OPTIMIZATIONS

A. Improved RSA Performance Through Assembly

RSA involves a lot of computation in order to encrypt and decrypt data. This makes it a good candidate for improvement

through arm assembly instructions instead of the previous C implementation.
We translated three functions from C to assembly: emod, isCoprime, and isPrime.

emod:

Three parameters are passed to the function: a, b, and c through the registers R0, R1, and R2 respectively. When the function is called, the variable registers (R4-R7) are pushed to the stack to be used in the implementation. Since the function will be returned through R0, the value in R0, a, must be moved to another register, R3. Then if R1 is 0, it branches to ret1 which stores 1 in R0 then pops the variable registers before branching back to the link register to restore the state of the register to what it was before the subroutine call. Since the ARM processor does not support the modulo operator, three instructions were used to implement b%2 and the result was stored in R5.

The instructions below describe x%y where the result is stored in Rd.

```
udiv    Rd, Rx, Ry
mul     Rd, Rd, Ry
sub     Rd, Rx, Rd
```

If R5 is 0, the function branches to ret2; otherwise, it branches to ret3. ret2 pushes the link register onto the stack, changes the parameter, b, to be passed on to the function by dividing R1 by 2, then calls *emod* function again. The return value of *emod* is returned through R0; then, R0 is multiplied by R0 and stored in R5 (emod*emod). To obtain (emod*emod) % c, the three instructions described above are used; the result is stored in R0. Then, the link register and variable registers are popped, and it branches back to the link register. Ret3 also changes the parameter, b, to be passed on to the function by subtracting R1 by 1 before pushing the link register onto the stack and calling *emod* again. The value of *emod* is returned through R0 which is then multiplied to a%c; then we reduce the result modulo c, store the result in R0 and return.

isCoprime:

Two parameters are passed to the function: a and b through the registers R0 and R1 respectively. The iterator starting at number 2 is stored in R2, then it loops until R2 is more than the value stored in R0 added to 1 (a+1) at which point 1 is stored in R0 and the function returns. Inside the loop, if a%i (calculated as described above) is 0, it branches to check2; otherwise, R2 is incremented, and it branches back to the loop. Check2 is used to check if b%i is 0; if this condition is also satisfied, then it branches to ret2 which stores 0 in R0 and branches back to the link register. Otherwise, R2 is incremented and the function branches back to the loop.

isPrime:

The parameter a is passed to the function through R0. The iterator starting at number 2 is stored in R1, then it loops until R1 is more than the value stored in R0 at which point 1 is stored in R0 and the function returns. The loop calculates a%i; if it is 0, it branches to ret0 which stores 0 in R0 and branches back to the link register. Otherwise, R1 is incremented, and it branches back to the loop.

To evaluate whether implementing RSA cryptosystem in assembly enhanced the performance of our system, we measured the execution time for each method. The results indicate that the speed of the execution is almost twice as fast using assembly

Measuring the method execution time for 1000 executions per method, the improvement is quantified below in 3:

1000 Executions per method:			
Method name	Assembly (ms)	C (ms)	Speedup
isPrime	11.037025	22.62191	2.049638648
isCoprime	30.574174	60.97544	1.994344541
emod	9.359162	12.98996	1.387940822

Table 3. Optimization results

IV. Conclusions

Our system could be used for 2-way communication between two physically connected locations. Additionally, as the system is able to transmit temperature data, it could be used for a weather prediction network.

Our boards use a wired UART connection, but it might be more applicable to use something like the LoRa module for wireless communication. This would enable better range for the boards to communicate, up to 15km as specified by the LoRa documentation [2]. In contrast, with our current wired communication, this would be very difficult to achieve and maintain.

References

[1] N. Mohideen, "OLED with Blue Pill using STM32CubeIDE," Micropeta, [Online]. Available: https://www.micropeta.com/video19. [Accessed 21 November 2022].

[2] Ebyte, "manualslib," manualslib, [Online]. Available: https://www.manualslib.com/products/Ebyte-E32-433t30d-10450577.html. [Accessed 4 December 2022].