BTP-II Report: Communication System with Hardware Encryption

Mohil Patel - 160070002

Guide:
Prof. Madhav Desai
Git Repo: `https://github.com/mohilp1998/BTP`

Spring 2020

Abstract

The project aims at developing an end-to-end communication system which can provide secure communication over the public internet without disrupting the conversation between the users. The audio system designed here provides hardware-level security to ensure safety against many security vulnerabilities attacks and adds a layer of protection above the software-level security.

To provide security in the audio system, we designed an end-to-end communication system hardware. In this new device, we introduced an additional hardware block in the audio data transfer pipeline. This block implements AES-128 encryption on the audio data at hardware-level and provides hardware-level security. We also added the capacity to program the device with a new encryption key easily. Additionally, we also added password protection over the hardware device as an extra security measure.

We demonstrated the complete audio system with two independent setups of the device and a local intranet server. The latency added by the audio system, to the conversation, is around 1 second over the local server, which did not hamper the communication. Overall the system did not hinder the natural flow of the conversation while providing additional security.

1 Introduction

With the advent of the public internet in today's era, we are seeing a growth in applications, providing audio communication over the internet. More people are choosing internet-based voice calling over the conventional method provided by telecommunication companies. This growth has been fueled by an increase in the accessibility of the internet around the world. But with this growth, there is also an increase in potential security vulnerabilities as we shift our communication to the public internet. These security issues are aggravated by the fact that most of the applications available are mostly software-based. With new cybersecurity attacks, like side-channel attacks, being discovered every day, this necessitates the development of specialized hardware to secure communication over the public internet.

To overcome these new security vulnerabilities, in this project, we aim at developing an end-to-end communication system which can provide secure communication over the public internet. The audio system designed in this project implements hardware-level security to ensure safety against many potential vulnerabilities and adds an extra level of protection above software-level security.

Figure 1 shows a top-level view of the complete system developed in this project. The audio transmission in our project starts at the microphone circuit and ends at the speaker circuit. We designed the speaker circuit to convert the digital stream of data, received via SPI protocol, to an analogue signal and output the audio after basic filtering. The microphone circuit, on the other hand, captures the audio signal and digitize it through ADC. The next block in Figure 1 is the microcontroller, the microcontroller handles all the digitized data and handles two major jobs. First, it interfaces with the analog circuits and ensures proper sampling & reconstruction of the analogue signal according to the Nyquist rate [1]. Second, it packages the digitized stream of data and transmits it to the FPGA Block via UART. This FPGA Block is a novel addition in this project, and it provides the hardware-level security aspects of the project. This hardware block implements AES-128 [2] and encrypts the audio stream providing additional security. The encrypted data is transmitted to the PC client via the UART protocol. The PC client then sends the data to the server, which forwards it to the other user. We developed the PC client such that we can easily change the encryption key whenever required. We also added password protection on the complete hardware to ensure additional safety, in case, it falls in the hands of an adversary.

We demonstrated the complete audio system by assembling two independent setups and testing the communication over them. To establish a connection between these setups, we hosted a local server. This local server provided a ping time of 1ms. The ping was kept minimal as the experiment was aimed at finding the latency added solely by the audio system. The test showed a lag of around 1 second during data transmission. This latency did not hamper the communication between the users. Overall the system did not hinder the natural flow of conversation while providing additional security.

The rest of the report is structured as follows. Section 2,3,4 and 5 covers details regarding the analog Circuits, the microcontroller, the FPGA Block (i.e. Encryption Module) and the PC client & Server respectively. In Section 6, we discuss the demonstration setup and the results. And lastly, we conclude with Section 7.

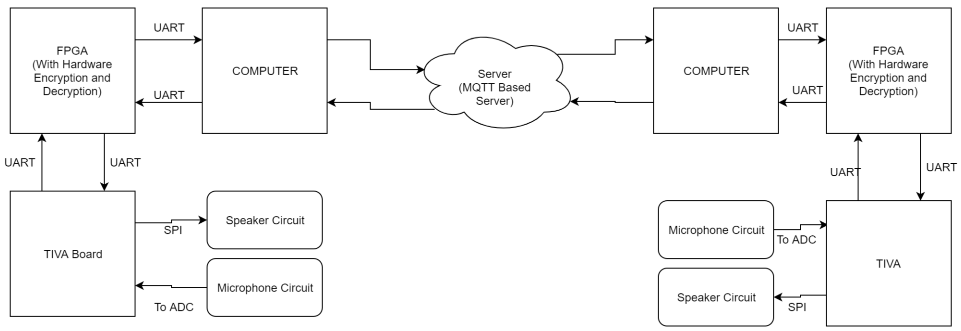

Figure 1: Complete System Block Diagram

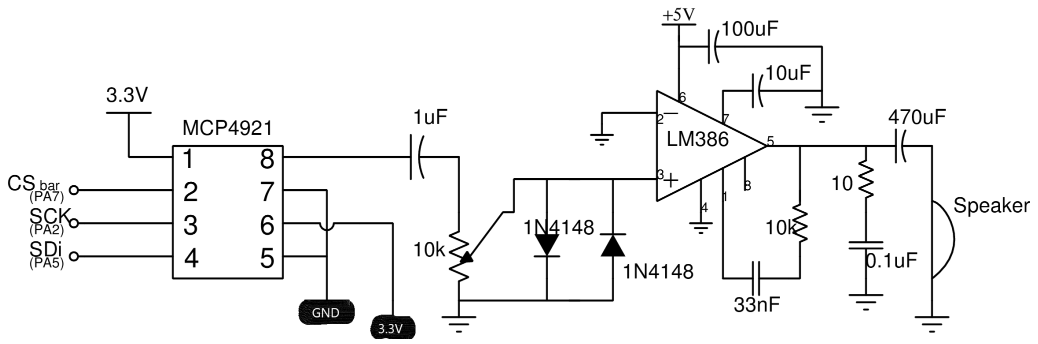

Figure 2: Speaker Circuit Diagram

2 Analog Circuits

As seen in Figure 1, representing the top-level view of the complete system, terminating blocks in the audio transmission pipelines are the analog circuits. There are two analog circuits used in our project, first is a speaker circuit and second is a microphone circuit. The microphone circuit is used to capture the audio into a digital stream, which is later digitized by the microcontroller ADC. On the other hand, the speaker circuit converts the digitized data stream, provided by the microcontroller, into the analogue signal and later converts the audio signal to voice via a speaker after basic filtering. In the following subsections, we are going to discuss these circuits in more details..

2.1 Speaker Circuit

In our audio system, we use the speaker circuit to convert the digital data stream provided by the microcontroller into an audio output. Figure 2 shows the complete block diagram of the circuit we used. The microcontroller transmits the digital data stream to the speaker circuit using SPI protocol (the pins shown in Figure 2 corresponds to the pin Layout for TIVA TM4C123G Launchpad

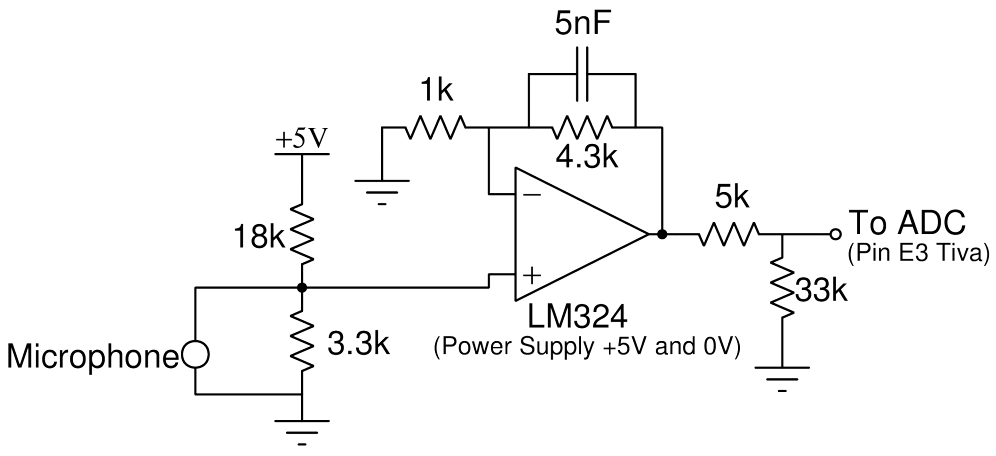

Figure 3: Microphone Circuit Diagram

Board [3]). The SPI stream is received by the IC MCP4921 [4], which is a DAC (digital to analog converter IC). For our implement, we set the SPI protocol to run at 1MHZ speed. Once the IC receives the data, it converts it into an analogue signal. This analogue signal then transmits through a capacitor and a 10k resistor potentiometer. The capacitor works as a DC blocker, whereas the pot is used to control the volume of the output signal. After this, the audio signal goes through an OPAMP LM386 [5] which provides amplification and active filtering to reduce noise. At last, we convert the analogue signal to audio using a speaker.

The filtering provided by the speaker circuit, shown in Figure 2, is minimal and can be improved. We believe more complex noise removal filtering can be implemented to improve audio quality. One such method can be the implementation of pre-emphasis and de-emphasis circuit to reduce the impact of the noise.

2.2 Microphone Circuit

Similar to the speaker circuit, the microphone circuit is also one of the terminating blocks in our audio system. The system uses the microphone circuit to capture and convert the audio into an analogue signal. This analogue signal is then converted into a digital data stream by the microcontroller ADC. Figure 3 shows the circuit diagram of the microphone circuit. The microphone circuit is a minimal circuit which uses an active low pass filter to reduce high-frequency noise and a voltage divider to limit the signal within the limits of the ADC. The low pass filter is implemented using the OPAMP LM324 [6], which has a cutoff frequency of 8kHz and provides a gain of 4.3. And a simple voltage divider is implemented using resistors 5k and 33k.

The microphone circuit we used is a basic circuit implementation and can be improved in many ways. One of the methods can be the implementation of pre-emphasis and de-emphasis, as discussed in the previous section. Many other filtering techniques can also be explored to improve circuit design.

3 Microcontroller

The microcontroller is the next block in the data stream path after the analog circuits. We use the microcontroller block for two major purposes. They are as follows:

1. Microcontroller interfaces with the analog circuits and ensures proper sampling of the analogue signal from the microphone circuit, and proper reconstruction of the analogue signal from the digital data stream for the speaker circuit. It ensures that the Nyquist rate [1] is maintained.
2. It interfaces with the FPGA Block, i.e. encryption module, and handles the packaging and transmission of data to and fro using UART protocol.

In the following parts of this section, we will discuss details regarding the microcontroller implementation and possible improvements.

3.1 Microcontroller Implementation

In our implementation, we used the TIVA TM4C123G Launchpad board [3] as the microcontroller board. We implemented the algorithm using embedded C in CCS (Code Composer Studio) [7]. The flow diagram of the algorithm is captured in Figure 4. In the rest of this section, we will discuss the major block of the flow diagram.

3.1.1 Initial Setup

The initial setup block initializes the microcontroller's peripherals properly and transmit the initialization data required by the encryption module in the FPGA. The following are the major things done by the block:

- **System Initialization**: Sets up the microcontroller to run at 80MHz clock rate.
- **ADC Initialization**: Initialize the ADC peripheral and setup up the ADC pin to sample data in trigger mode, i.e. sampling the data only when a command is issue by microcontroller.
- **SPI Initialization**: Initialize the SPI peripheral and setup the SPI speed to 1MHz and 8 bits per cycle.
- **UART Initialization**: Initialize the UART pin to work at 230400 baud rate, 8 data bit and 1 stop bit configuration.

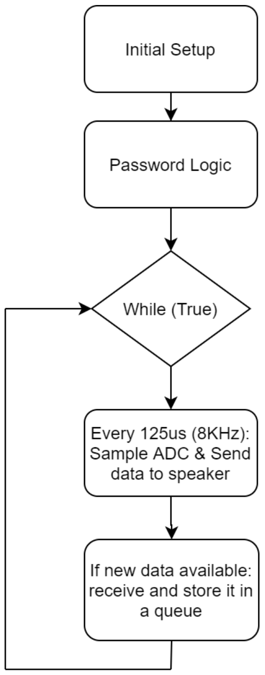
Initial Setup
Password Logic
While (True)
Every 125us (8KHz): Sample ADC & Send data to speaker
If new data available: receive and store it in a queue

Figure 4: Microcontroller Code Flow Diagram

- **Timer Interrupt Initialization**: Initialize the timer interrupt to call the interrupt service routine every 125μs (8kHz, sampling rate of the system).
- **Enable Flash**: Setting up the flash memory to be used to save the password.
- **Transmit Encryption Module data**: Transmits the initialization data, i.e. sbox and rcon, required by Encryption module.

3.1.2 Password Logic

The next critical block in the microcontroller is the password logic. The password logic handles three essential tasks, setting up a new password for new users, storing the new set password and checking the password. Following are the critical parts of this block:

- **Password Initialization**: This part handles the process of setting up a new password for the first time users. It asks the new users to set up a new password and stores it in the flash memory.
- **Password Checking**: This part asks the user for the password and checks whether it matches the password stored in the flash memory. If it is correct only then the user will be allowed to proceed forward else, the algorithm terminates there.

3.1.3 While Loop

The last essential element of the microcontroller is the while loop. This loop handles all the data transfer between the analog circuits and the FPGA block. On the one hand, it ensures proper sampling and reconstruction for the analog circuits, whereas, on the other side, it handles packaging and data transmission for FPGA Block. Following are the critical parts of this block:

- **Sampling and Reconstruction of Signal**: This part uses the timer interrupt to sample new data every 125μs (i.e. 8kHz sampling rate) & transmit it via UART to FPGA block. It also sends the received data from the FPGA block via SPI to the speaker circuit every 125μs, this ensures proper reconstruction of the audio signal.
- **Polling for UART data**: Here we continuous poll for new UART data from the FPGA, and whenever we receive the new data, we store it in a queue from which the data is retrieved every 125μs (by earlier part) to send it to the speaker.

3.2 Possible Improvements

We believe that the microcontroller logic can be improved to add more security to our system. One of the ideas can be the addition of a keyboard and LCD setup on the microcontroller. This setup can be used to input the password and the new encryption key to the audio system. Currently, we are using the PC client to transfer the password and the encryption key to the hardware. But if a keypad and LCD is added to the hardware setup, we will no longer require to transfer critical data from the PC client, and it will improve the security of the system. This idea is one suggestion to improve the security of the device, and many other ideas can be explored.

4 FPGA Block

FPGA block is the next block in our audio system. FPGA Block implements the encryption engine to provide the hardware-level security aspects of the project. It consists of two major part, first is the Encryption Module, and the second is the UART. A top-level view of the FPGA Block can be seen in Figure 5.

The UART in the FPGA is coded in VHDL language. The UART parameters are set to receive and send data at the baud rate of 230400 bit/s. It connects the FPGA Block to the microcontroller and the PC client as can be seen in Figure 5.

On the other hand, we developed the encryption module using the AHIR tool [8]. AHIR is a new system developed by Prof. Madhav Desai, Electrical Engineering IIT Bombay. AHIR enables hardware compilation of a circuit description. The input entry is a high-level programming language, and the output is fully functional VHDL. We used this tool to develop the VHDL code for the encryption module.

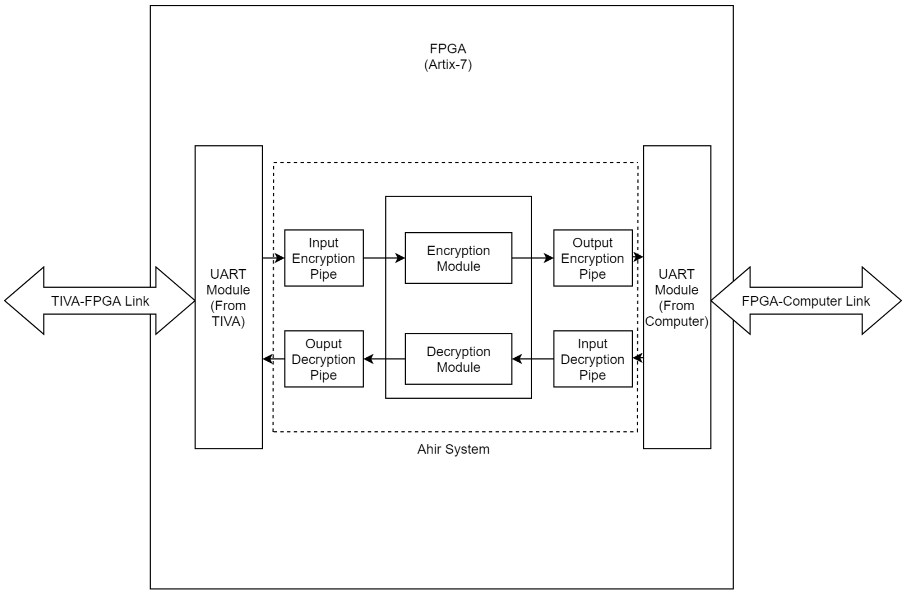

Figure 5: FPGA Block Overview

In the rest of this section, we cover the details regarding the encryption module.

4.1 Encryption Scheme

In this project, we used the AES-128 [2] encryption scheme to encrypt the audio data. The mode used is counter-mode [9]. Figure 6 shows a depiction of the counter-mode operation [9]. The counter-mode of operation requires both encrypting and decrypting to be in-sync. It creates a notion of state in the system which must be maintained by both encrypting and decrypting blocks, else data decryption may go wrong. Even though counter-mode has a critical drawback of maintaining states, we still selected this mode of operation as compared to other methods it provides security against a large number of cyberattacks while keeping the complexity of the process to minimal.

4.2 Encryption Engine Algorithm

We implemented the encryption module using the AHIR tool [8]. As stated earlier, the AHIR tool takes as input a high-level circuit description and produces a VHDL as the output. In the AHIR tool, the high-level description is provided using AA language. AA language is explicitly developed to give the circuit descriptions to the AHIR tool. We used AA language to build the encryption module. In this part of the section, we discuss the encryption module implementation from an algorithmic point of view. The flow diagram of the

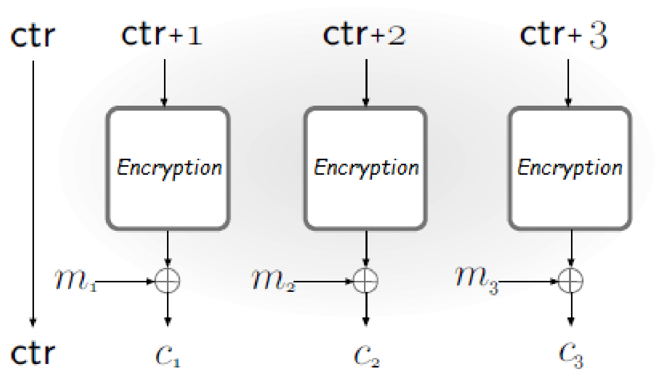

Figure 6: Counter Mode Encryption

encryption module is shown in Figure 7.

4.2.1 Initial Setup

This block mainly helps in the initial setup of the system. It includes two major parts, first is the reception of sbox and rcon [2] from the microcontroller and the second is letting the password messages go through, in between the microcontroller and the PC client.

4.2.2 Handshaking

As we discussed in the encryption scheme section, we require both the users to remain in-sync for the counter-mode operation [9] of the encryption scheme. During the initial setup time, as there is uncertainty in the sync status of both the users, we use this handshaking block to set up the initial sync between two users. It implements a handshaking protocol in which users on each send message to the other asking for awake status. And, once both of them receive the confirmation from the other user, only then the communication starts.

4.2.3 Decryption Loop

As can be seen in the Figure 6, in the counter mode of operation the counter is encrypted, which is then XORed with the message, and the resulting is the ciphertext. The decryption also requires encrypting the counter and then XORing with the ciphertext to retrieve data.

As we can see in Figure 7, in the decryption loop, initially the counter is encrypted after which we wait till we receive data worth AES-128 [2] block size. Once the data is collected, it is then decrypted by XORing, and then we verify the checksum, which was generated by encryption loop while encrypting at the other side. In our system 1 block of AES consists of 14 data bytes and 2 checksum bytes, thus a total of 16 bytes, i.e. 128 bits. If the checksum is correct,

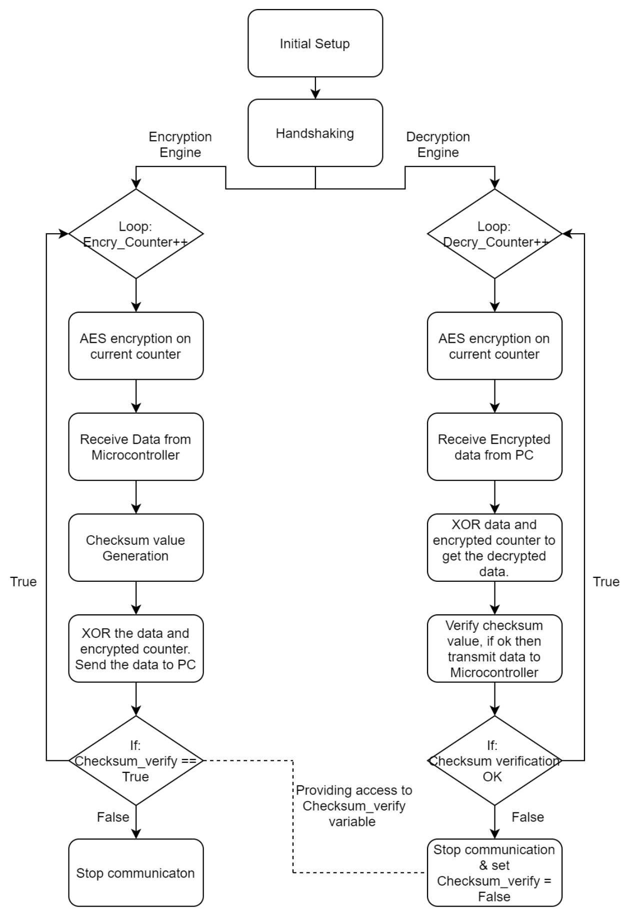
Initial Setup
Handshaking
Encryption Engine
Decryption Engine
Loop: Encry_Counter++
AES encryption on current counter
Receive Data from Microcontroller
Checksum value Generation
XOR the data and encrypted counter. Send the data to PC
If: Checksum_verify == True
True
False
Stop communicaton
Providing access to Checksum_verify variable
Loop: Decry_Counter++
AES encryption on current counter
Receive Encrypted data from PC
XOR data and encrypted counter to get the decrypted data.
Verify checksum value, if ok then transmit data to Microcontroller
If: Checksum verification OK
True
False
Stop communication & set Checksum_verify = False

Figure 7: Encryption Engine Flow Diagram

we transmit the data forward and continue to next iteration of the loop. Else we stop the decryption process & sets a variable checksum_verify as false to inform the encryption engine of possible out of sync condition and stop its operation.

4.2.4 Encryption Loop

Similar to the decryption loop, we start with encrypting the counter, after which we receive the data. Once the data is received, then we calculate the checksum and include it in the block for encryption. The complete data is then XORed with the encrypted counter to encrypt the message and is transmitted ahead. After sending the data we check the checksum_verify variable for possible out of sync, if we are not out of sync, then we continue for next iteration else the system is stopped completely.

4.3 Resource Utilization of FPGA Block

The FPGA block was compiled using Vivado 2019.1 [10], and is tested on Nexys 4 DDR Board [11]. The resource utilization is as follows, **7186 LUTs** and **17510 registers**.

5 PC Modules & Server Details

In our audio system, PC client plays three essential roles:

- It communicates data with the FPGA block.
- It transmits and collects the data from the server.
- It sets up the password and new encryption key for each session.

We implemented our PC client using python. The flow diagram of the PC client is shown in Figure 8.

We also need a server to transmit the audio data between two users. The primary job of the server is to send the data to and fro between both the users. In the rest of this section, we cover all the details regarding the server and the PC client.

5.1 Server

As discussed earlier, the primary job of a server in our system is to send audio data to and fro between both the users. To implement this facility, we implemented an MQTT [12] based server in our audio system. MQTT is a light-weight messaging protocol and requires a tiny amount of network bandwidth to operate. MQTT uses a publish/subscribe [13] mechanism for its messaging protocol. In our server, we used two independent channels to communicate data between the users. On one channel, user A publishes while user B subscribes to the channel and continuously receive the data. Whereas on the other channel, user B publishes and user A subscribes to receive the data.

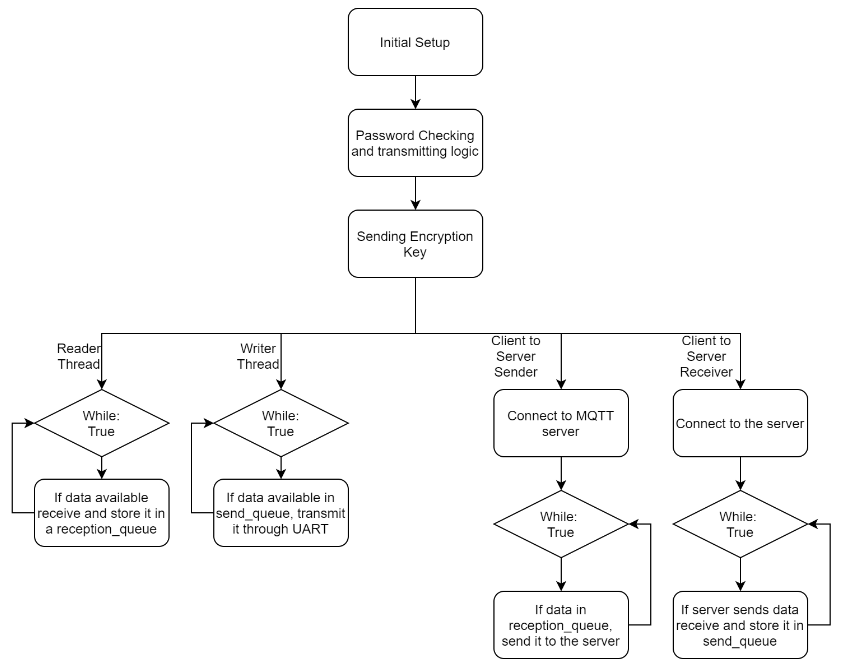
Initial Setup
Password Checking and transmitting logic
Sending Encryption Key
Reader Thread
Writer Thread
Client to Server Sender
Client to Server Receiver
While: True
If data available receive and store it in a reception_queue
If data available in send_queue, transmit it through UART
Connect to MQTT server
If data in reception_queue, send it to the server
Connect to the server
If server sends data receive and store it in send_queue

Figure 8: PC Client Flow Diagram

5.2 PC Client Details

In this part of the section, we cover details regarding the implementation of the PC client in our system. The flow diagram of the PC client is shown in Figure 8. Here we discuss each part of this flow chart in details.

5.2.1 Initial Setup

The PC client communicates with the FPGA block using the UART protocol. In the initial setup block, we set up the UART with 230400 baud rate, 8-bit data packet and 1 bit stop configuration to ensure that we can communicate data properly. Initial setup blocks also set up necessary thread-safe queues to transfer data safely from one thread to the other.

5.2.2 Password and Encryption Key

This block handles all the communication between user and the microcontroller during password setup and checking. It receives the instructions from the microcontroller and transmits the user inputs to the microcontroller. This block also handles the transmission of the current session encryption key to the encryption module.

5.2.3 FPGA to PC Client Communication

We use two thread to communicate data between FPGA and the PC client. One thread focuses on receiving the data from the FPGA, and the other thread pushes the data to the FPGA via UART.

Reader Thread: The reader thread uses continuous polling to look for the new data received from the encryption module. Once it receives the data packet, it pushes the data into a queue. This queue is later used by another thread to transmit data to the server.

Writer Thread: The writer thread puts the data it received from the server into the out UART buffer, from where the data is transmitted to the FPGA block. The writer thread receives the server data from a queue. It is the job of another thread to collect and store the data from the server into this queue.

5.2.4 PC Client to Server Communication

We handle the communication between the PC client and the server using two thread. Similar to the interface between the PC client and the FPGA block, here one thread transmits the data to the server, whereas the other receives the data from the server.

Sender Thread: The sender thread of client to server communication starts by setting up the communication link between the server and the client as a sender. Once set, it begins sending the data one by one from the queue in which reader thread has stored the data.

Receiver Thread: The receiver thread, similar to sender thread, initially establishes the communication between client and receiver as the receiver. Once the connection is established, it receives the data from the server and writes it to a queue which is later used by the writer thread to transmit data further.

6 Demonstration

The audio system, designed in this paper, is meant to be used in the real world. With this section, we aim to evaluate its real-world performance. To assess the system, we assembled two independent setups and evaluated the communication between two users. We tried to observe the latency in the transmission and its effects on the conversation. The final results show that the system adds a small delay of around 1 second, which does not impede the natural flow of conversation. Overall the system achieves its objective to add additional hardware security without hampering the communication excessively.

6.1 Setup

To set up the complete prototype of the audio system, we implemented each of its different elements. We built the analog circuits on the bread-boards. We used TIVA-TM4C123G Launpad Board [3] as the Microcontroller and programmed them using embedded C. For FPGA, we used Nexys 4 DDR Boards [11] and implemented them using the AHIR tool [8]. AHIR is a tool designed by Prof. Madhav Desai, Electrical Department IIT Bombay, which enables the hardware compilation of a circuit description. The input is a high-level programming language, and the output is a fully functional VHDL. To implement the PC client, we used python scripts. And the server used in the setup was an MQTT server [12], hosted over the local intranet.

In our implementation, we assembled two independent setups of the audio system. Each setup had its own analog circuits, microcontroller, FPGA and a PC for the PC client. The server was hosted over the local intranet and which provided a minimal ping delay of 1ms. The local hosting was done to avoid the impact of the latency added by public internet and to get a clear idea of the latency added by the audio system only. Finally, to test the audio system, we asked two users to communicate via the setups, and we evaluated the latency as well as the hindrance to natural conversation flow.

6.2 Results

The evaluation of the audio system provides two essential results. First, it shows the latency added by the audio system. Latency is critical as an excess latency will affect communication. And second, it identifies how the complete audio system affects the natural conversation flow.

The latency exists in the audio system because the data has to go through multiple different elements to reach from user one to the other. The data travels from the analog circuit to microcontroller. The microcontroller, after processing and formatting the data, transmits it to the FPGA block. The FPGA block adds more latency due to the encryption module and sends it to the PC client. The PC client stores it and sends it to the server, from where the other user receives it and the data goes through the complete reverse cycle. The total latency added by this flow is around 1 second in our testing setup. This latency is not very significant and does not affect the communication.

As for the natural flow of conversation, it can be impacted by multiple aspects of the communication system. One of the parts is the latency which as we saw was not significant and did not affect the conversation. The second part is the loss of data over the public internet. In our testing, we did not lose any data, and the conversation went smoothly. Overall the communication over the system felt natural, without any hindrances.

6.3 Real World Server Latency

Even though our demonstration used a local intranet server with a minimal ping of 1ms, this server can easily be shifted to any public domain server. To support this statement, we measured the average ping time for the intercontinental delays, from various sources, which was around 342ms [14]. This result shows that even if we use an intercontinental server, the latency will increase to around 1.5 seconds. This latency is still small enough not to impact the communication system and does not hamper natural conversation flow.

7 Conclusion

The audio system, in this project, develops an end-to-end communication system which provides secure communication over the public internet. The audio system improves over other similar products by delivering hardware-level security and overcomes many vulnerabilities faced by other products. It achieves the hardware-level security by introducing a hardware block within the audio data transfer pipeline. In our implementation, we used an FPGA block to work as the hardware block and implemented AES-128 encryption scheme to secure the data at the hardware level itself. Our demonstration shows that the latency added by the whole system is around 1 second and does not impact the natural flow of conversation between the users.

In this project, we focussed mainly on securing communication over the public internet. But the key concept of introducing a hardware block within the audio data transfer pipeline can be used to explore many future projects. Following are few such ideas:

Compression. The hardware block can be used to implement compression on the audio data stream. This compression will help in reducing the load on the network and help communicate comfortably, even with low bandwidth availability.

Language Processing. The hardware block can be used to design complex natural language processing modules at the hardware level. The specialized hardware will benefit from the faster performance and can help in providing a real-time implementation of language processing.

Acknowledgments

Thanks to Akshat Singhal (170070040) and Vidur Shah (183079035) for their help in this project. Thanks to WEL Lab, IIT Bombay and VLSI Design Lab, IIT Bombay for providing tools for this project. And, thanks to Prof. Madhav Desai for his continuous guidance during the project.

References

[1] M. H. Weik, *Nyquist theorem*, pp. 1127–1127. Boston, MA: Springer US, 2001.

[2] F. P. Miller, A. F. Vandome, and J. McBrewster, *Advanced Encryption Standard*. Alpha Press, 2009.

[3] TEXAS INSTRUMENTS, *Tiva^{TM} C Series TM4C123G LaunchPad Evaluation Board*, April 2013.

[4] MICROCHIP, *8/10/12-Bit Voltage Output Digital-to-Analog Converter with SPI Interface*, April 2010.

[5] TEXAS INSTRUMENTS, *LM386 Low Voltage Audio Power Amplifier*, May 2017.

[6] TEXAS INSTRUMENTS, *LMx24-N, LM2902-N Low-Power, Quad-Operational Amplifiers*, January 2015.

[7] T. INSTRUMENTS, "Ccstudio code composer studio 9.10." `https://www.ti.com/tool/CCSTUDIO`.

[8] S. Sahasrabuddhe, H. Raja, K. Arya, and M. Desai, "Ahir: A hardware intermediate representation for hardware generation from high-level programs," pp. 245–250, 01 2007.

[9] Y. L. Jonathan Katz, *Counter Mode of Operation in Encryption*, pp. 92–92. Boca Raton, FL: CRC Press, US.

[10] Xilinx, "Vivado 2019.1." `https://www.xilinx.com/products/design-tools/vivado.html`.

[11] Xilinx, "Nexys 4 ddr board." `https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start`.

[12] MQTT, "Mqtt version 5.0." `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf`.

[13] MQTT, "paho-mqtt." `https://pypi.org/project/paho-mqtt/`.

[14] "Ping test." `https://www.cloudping.info/`.