CSE 462M Capstone Design Project Final Report

Submitted to Professor Hall and the Department of Computer Science & Engineering

Optimizing Cryptographic Performance and Education: AES Encryption and FPGA Hardware Acceleration

Group Members:

Gianna Corinne Pedicini

Department of Computer Science & Engineering

B.S. Computer Engineering and B.S. Computer Science

g.c.pedicini@wustl.edu

Isabella Marie Lombardo

Department of Computer Science & Engineering

B.S. Computer Engineering

i.lombardo@wustl.edu

Date of Submission: May 2, 2025

Project Duration: January 2025 – May 2025

Abstract

This project addresses the challenge of making modern encryption algorithms more understandable and accessible by developing an educational platform that demonstrates the Advanced Encryption Standard (AES) encryption process in real time. The solution features a web-based interface that accepts user-input plaintext and displays the encrypted output, while also leveraging a PYNQ-Z2 board to accelerate the encryption process via hardware. The backend is implemented using the Python Bottle framework, enabling seamless communication between the front-end interface and the hardware over a REST API. HTML, CSS, and JavaScript form the interactive frontend within a Jupyter Notebook environment.

The platform integrates both a hardware-accelerated AES pipeline and a software-based reference model for verification and comparison. Key AES steps such as SubBytes, ShiftRows, MixColumns, AddRoundKey, and the key expansion algorithm are implemented and validated. Performance testing showed a significant reduction in encryption time when using the hardware-accelerated approach compared to the software model, demonstrating the efficiency benefits of offloading cryptographic computation to an FPGA.

The result is a functional and educational tool that bridges the gap between theoretical cryptography and practical implementation. By providing real-time feedback, step-by-step explanations, and performance comparisons, this platform helps users gain deeper insights into AES and the advantages of hardware acceleration.

Introduction

As data leaks continue to make daily headlines, encryption is no longer just an option—it's a necessity. In this past year alone, the average cost of a single data breach soared to an all-time high of 4.88 million dollars (“Cost of a Data Breach Report 2024”). From medical records to confidential government documents, data privacy is essential for individuals, businesses, and national security alike. The Advanced Encryption Standard (AES) is the strongest form of protection against these digital attacks, arming itself with a multitude of large, cryptographic keys and an array of data manipulation techniques (Awati). Maintaining the highest level of security over digital media, AES encryption is widely employed by the U.S. government and industry leaders alike for secure data transmission and storage (Awati).

Despite AES encryption's importance, understanding the step-by-step processes involved in secure encryption remains complex and inaccessible to many learners. Online documentation of AES often assumes prior familiarity with linear algebra and finite field arithmetic, making it difficult for novices to grasp. While some tools like CrypTool 2 offer visualizations of cryptographic algorithms, they often lack interactive, real-time demonstrations that integrate hardware acceleration, limiting their effectiveness in conveying the practical aspects of encryption processes (Esslinger). Furthermore, while encryption can be software-driven, hardware acceleration offers the potential for significant improvements in speed and security — especially in embedded systems and edge computing — yet practical, interactive demonstrations of this are limited or non-existent for educational use (Dandass et al.). This gap in understanding AES encryption and hardware offloading techniques creates an educational void that demands attention.

Problem Statement

To address this gap, this project aims to develop an educational website that provides a clear, interactive demonstration of the AES encryption process, integrating a PYNQ-Z2 FPGA circuit board to demonstrate the hardware acceleration of AES encryption. Users will be able to input plaintext, observe AES-128 encryption, and compare hardware-accelerated results against software-only implementations in real time. These interactions will be facilitated via a REST API between the frontend and the FPGA backend. By merging intuitive visualization with embedded hardware, the project seeks to make advanced cryptographic concepts both accessible and engaging for educational environments.

Project Objectives and Requirements

The primary objective of this project is to create an interactive educational platform that illustrates the internal workings of the Advanced Encryption Standard (AES) algorithm while demonstrating the benefits of hardware acceleration using a PYNQ-Z2 FPGA board. Originally, the platform was envisioned to include user-inputted keys and to illustrate each step of AES encryption in a user-friendly and visually engaging manner. The system is intended to serve as a teaching tool for students, educators, and early learners in cybersecurity and computer engineering, offering an intuitive and practical experience of symmetric encryption.

At its core, the platform enables users to input plaintext messages before observing the encryption process unfold. A key feature of the system is its ability to offload encryption computations to an FPGA board in real time and compare this hardware-based implementation to a software-only equivalent executed on the host processor. This side-by-side comparison

provides users with insight into how hardware acceleration impacts performance, especially in contexts where large amounts of data must be processed efficiently and securely.

To support these goals, the platform must meet a range of functional requirements. First, the user interface must be accessible through a web browser, allowing users to explore the website, enter messages, and request encryption or decryption operations. The system must then transmit this data to the backend server using a RESTful API. The backend coordinates the execution of encryption either through Python software routines or by communicating with the PYNQ-Z2 board to trigger the hardware-accelerated process. Upon completion, the encrypted ciphertext (or decrypted output, if requested) must be returned to the frontend and clearly displayed to the user along with metadata such as execution time. This system architecture is visualized below as a block diagram showing the flow from user input to encryption via software and hardware paths, with REST API communication between frontend, backend, and PYNQ-Z2 (Figure 1).

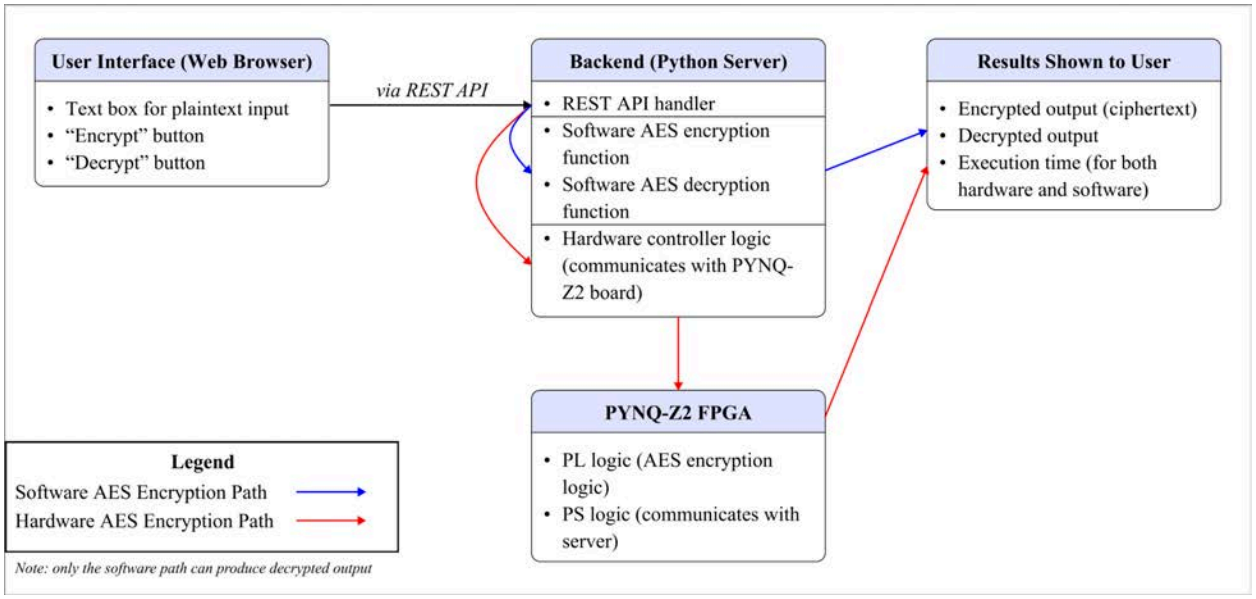


***Figure 1**: System Architecture Block Diagram.*

The platform must also satisfy several non-functional requirements. In terms of performance, the FPGA-based encryption should demonstrate quantifiable speed improvements compared to the software implementation, particularly when running batch operations. Usability is also paramount: the platform must feature an intuitive, modern interface that guides users through the encryption process without requiring prior knowledge of hardware systems or cryptography. Responsiveness is another critical criterion; the interface should respond promptly to user inputs and function consistently across desktop and tablet devices. The system must also be scalable in its backend design, ensuring it can support multiple concurrent users without crashing or lagging, especially in classroom environments or public demonstrations. Security is a fundamental concern, given that user inputs include plaintext messages; communication between the frontend, backend, and FPGA must be encrypted and secure. Lastly, maintainability is essential: the codebase must be modular and well-documented to enable future enhancements, such as supporting multiple key lengths, user-input keys, or visualizations of user input at each step of encryption.

Supporting these requirements are the following key design objectives. The first is educational clarity, breaking down the complex AES process into an engaging, visual experience that demystifies cryptographic transformations for learners at various levels. The second is real-time feedback, achieved by allowing users to observe and compare execution times between hardware and software in a dynamic, transparent manner. The third objective is seamless hardware integration, such that the underlying FPGA operations remain hidden behind an intuitive frontend, allowing users to focus on the algorithm rather than the technicalities of interfacing with hardware. Lastly, the platform prioritizes aesthetic appeal, featuring a polished,

modern design that helps maintain user engagement while reinforcing the credibility and professionalism of the educational experience.

Ethical Considerations

Alongside its technical objectives, the project also acknowledges several ethical considerations. Since the platform accepts user input for encryption, care must be taken to prevent data leakage and ensure that user-provided plaintext are not stored or exposed during processing. A poorly implemented encryption algorithm could mislead learners or create a false sense of security, making correctness and clarity imperative. Additionally, while the tool is designed for educational purposes, there remains a risk that it could be misused, such as obscuring malicious content or facilitating unauthorized communications.

A second ethical concern lies in the accuracy of the encryption algorithm's implementation. Since the platform is intended as an educational tool, an incorrectly implemented AES algorithm could mislead students and potentially propagate misinformation about how encryption works. This could contribute to a false sense of security, especially if students or educators mistakenly rely on it for anything beyond instructional purposes. To address this, the implementation must be thoroughly tested against known AES test vectors to ensure correctness, and the platform must clearly label itself as an educational demonstration, not a secure encryption tool for real-world use.

Additionally, the open and accessible nature of the tool raises questions about misuse. While the goal is to promote understanding of cryptographic systems, there is always a possibility that someone could use the tool to encrypt malicious content or facilitate unauthorized communication. Although this risk cannot be fully eliminated, the platform limits exposure by

processing data locally in real time and all educational documentation accompanying the tool will emphasize responsible use, proper context, and ethical handling of encryption technologies.

An ethical dilemma encountered during the development process involved balancing accessibility with security. The goal of creating a transparent and approachable interface sometimes conflicted with the desire to restrict misuse. For example, allowing users to enter arbitrary text or custom keys could offer a more complete learning experience, but also increase the risk of misuse or data mishandling. To resolve this, initial iterations of the platform restrict encryption to fixed keys and plaintext lengths to limit scope while ensuring educational effectiveness. Future updates may revisit this design decision once additional safeguards are in place.

Finally, inclusivity and accessibility are also part of ethical design. The interface must be visually accessible, responsive across devices, and free of unnecessary complexity, ensuring that learners with varying levels of technical background can use the platform effectively. Overall, this project strives to model ethical responsibility by prioritizing user privacy, technical accuracy, transparent documentation, and responsible educational practices.

In summary, this project combines software engineering, hardware design, and cryptographic theory to deliver an educational tool that is both functionally rich and pedagogically valuable. Through careful attention to performance, usability, and system robustness, the project aims to lower the barrier to understanding AES encryption while demonstrating the real-world benefits of hardware acceleration in secure computing.

Methods

Our project is deployed on a PYNQ-Z2 board, which contains both a processing system (PS) and programmable logic (FPGA). The user interface is accessible through a web browser and features several pages, one of which contains the option to encrypt a user-inputted message using the FPGA on the board. The website as a whole is maintained in the Jupyter Notebook on the board. Once the plaintext is received, it is transmitted to the backend, which is implemented using the Python Bottle framework, via a REST API. Then, the plaintext data is sent to the FPGA using an AXI4-Stream channel managed by the Direct Memory Access (DMA) controller. The DMA pulls the plaintext data from DDR memory and sends it to the AES IP block to be encrypted. The AES IP block then sends the ciphertext to the DMA, which writes it back to the DDR to be returned to the Bottle server and displayed on the website frontend.

Because the goal of this project was to create a working example of AES encryption with an interactive website, the team's initial design approach included creating AES encryption and decryption for testing in Python, creating AES encryption in Vitis HLS that was capable of processing arbitrarily large data, testing both with each other and a third party website, and combining that tested work into a single, interactive website that showed intermediate, step-wise results of encrypting a user-inputted plaintext. Due to limited time and scope, the team was unable to implement processing arbitrarily large data and sending intermediate results, but they did send the final ciphertext to the website frontend.

The AES algorithm itself is a block cipher that separates data into 16-byte blocks and encrypts them one at a time. During encryption, each block is arranged into a column-major-ordered 4x4 array known as the state. The process begins with the plaintext and the key: the "encryption password" used throughout the process to effectively encrypt the

plaintext. There are three versions of AES, each differing in the size of their keys: AES-128, AES-192, and AES-256. The number refers to the key length in bits. This project focused on implementing AES-128 over -192 or -256 due to efficiency and simplicity. Further mentions of AES will implicitly refer to AES-128.

The first step of AES is expanding the key to create a key for each round in addition to keeping the initial key. The key expansion process is visualized in Figure 2 below. RotWord performs a one-byte left circular shift. SubWord is an application of the AES S-Box to each of the four bytes of a word (32 bits). Rcon, short for "round constant," is the sequence of 32-bit constants XOR'd into the first word of each round key. It is generated in Equation 1 below:

$$Rcon[i] = (2^{i-1}) \,||\, 0x00 \,||\, 0x00 \,||\, 0x00$$

Equation 1: *Rcon.*

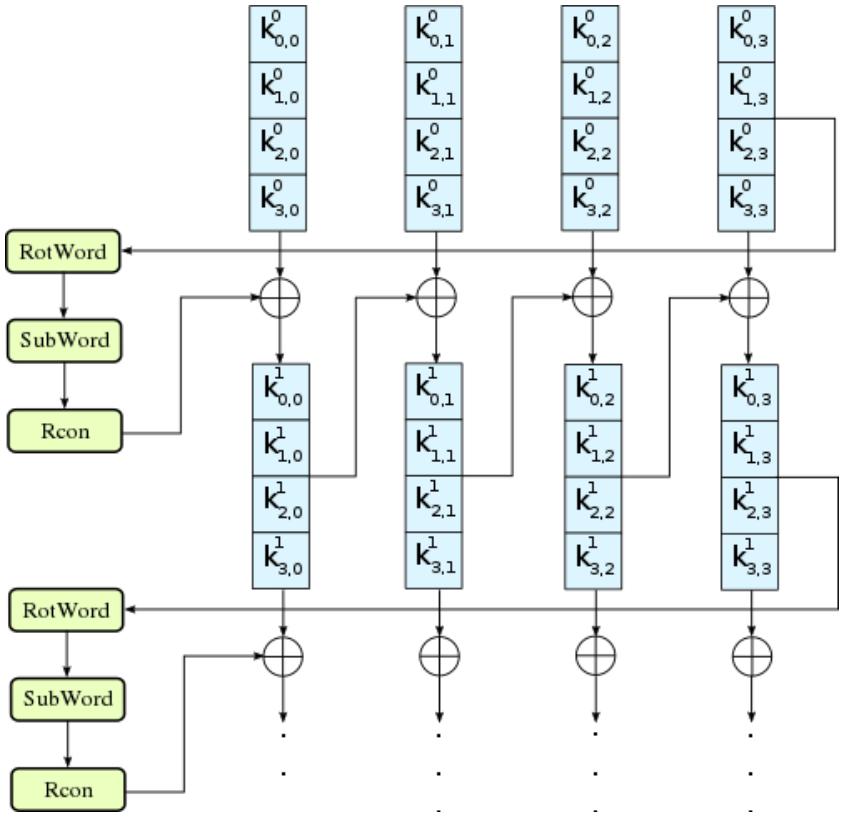


Figure 2: *The key expansion algorithm (AES Key Schedule).*

After expanding the keys, each byte of the state is combined with each byte of the original key using bitwise XOR. This process is identical to the AddRoundKey step, and a visualization can be seen in Figure 7. For nine of the ten rounds (the total number of rounds will differ for a different version of AES), the following steps are completed: SubBytes, ShiftRows, MixColumns, and AddRoundKey.

SubBytes

In the SubBytes step, each byte in the state is replaced with a byte in the substitution box (S-Box). The S-Box, included in Appendix D, acts as a look-up table and was formulated specifically for AES encryption. The value of a given byte in the state array is used as the index to obtain its substitution. Figure 3 visualizes the SubBytes step.

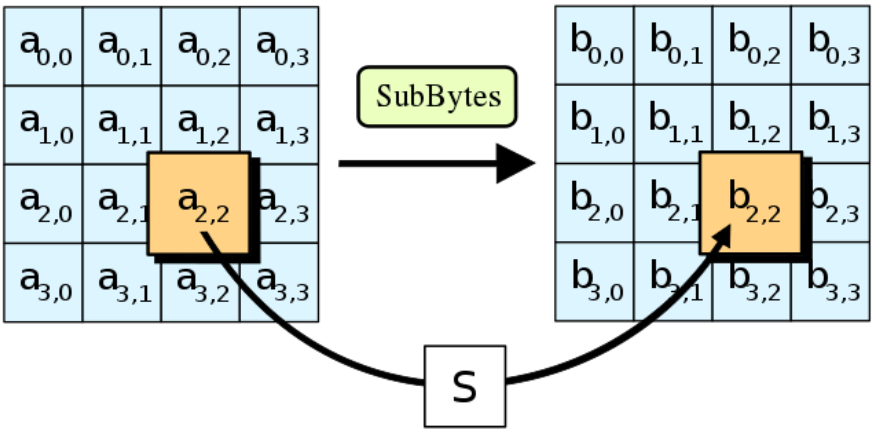


Figure 3: *The SubBytes step (Advanced Encryption Standard).*

ShiftRows

In the ShiftRows step, each row of the state is shifted to the left a certain number of times. The first row is not shifted, while the second, third, and fourth rows are shifted one, twice, and thrice, respectively. Figure 4 visualizes the ShiftRows step.

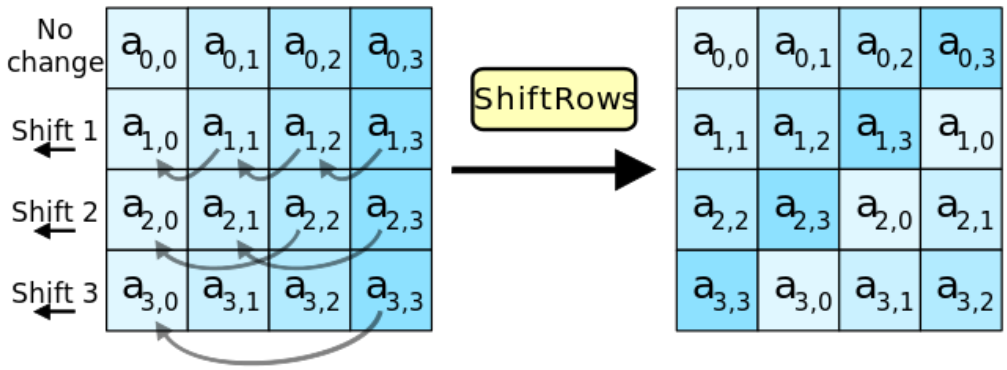


Figure 4: *The ShiftRows step (Advanced Encryption Standard).*

MixColumns

In the MixColumns step, each column of the state is multiplied with a known matrix c(x). Figure 5 shows c(x), and Figure 6 visualizes the MixColumns step as a whole.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Figure 5: *c(x).*

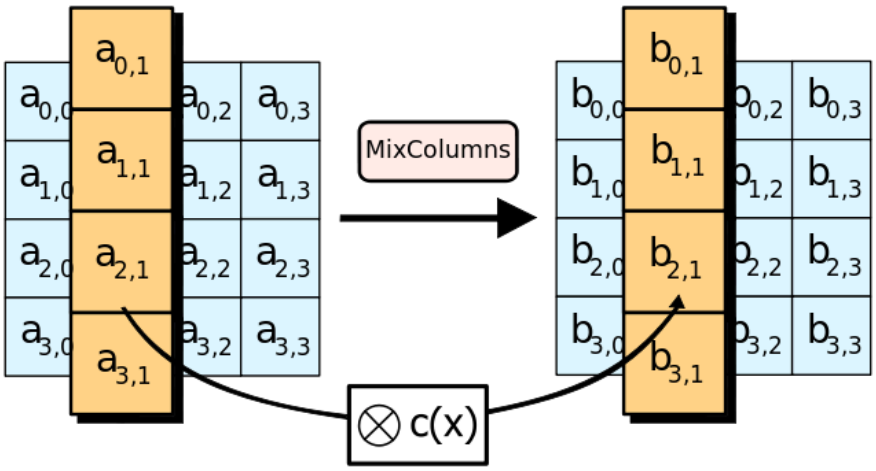


Figure 6: *The MixColumns step (Advanced Encryption Standard).*

AddRoundKey

In the AddRoundKey Step, each byte of the state is bitwise XOR'd with each byte of the round key generated for that round. Figure 7 visualizes the AddRoundKey step.

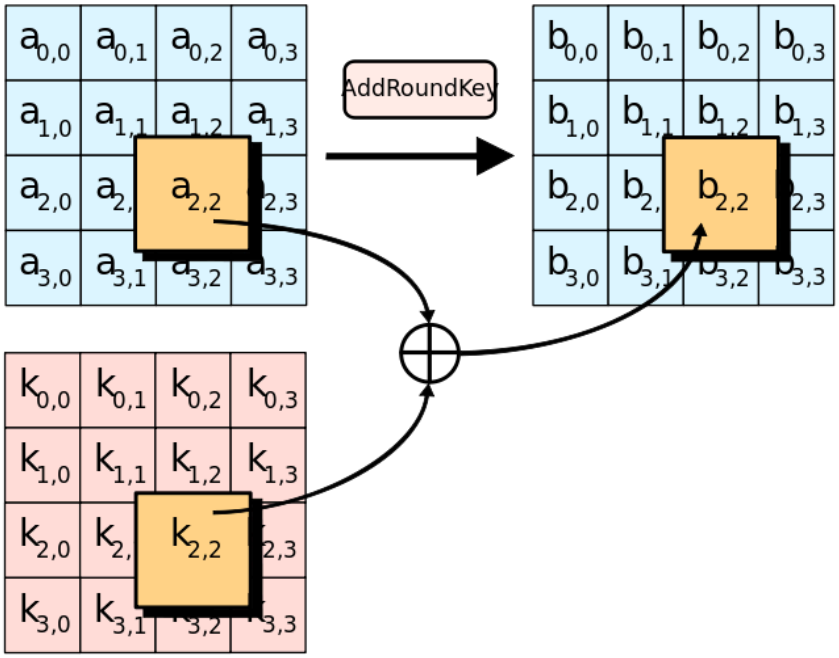


Figure 7: *The AddRoundKeyStep (Advanced Encryption Standard).*

In Round 10, the MixColumns step is omitted.

For the hardware components of the design, Vitis HLS was chosen as the development environment for the hardware implementation of AES for several reasons. While it was initially an unfamiliar workspace, it was much easier to design the hardware in C and package it as a single IP block rather than design it in SystemVerilog. The choice was made to create the AES implementation as a single IP block because it provided simplicity of integration and easy testing. Additionally, the environment itself made testing and debugging straightforward, and the ability to use HLS pragmas, such as loop unrolling and array partitioning, made improving

timing simple. Within Vitis HLS, the choice was made to hardcode the large, constant arrays associated with AES encryption and the encryption key. This eliminated the need for register writes, and the key is secure within the FPGA. For the data path, the design used a free-running kernel, which eliminated handshaking, and AXI-stream ports, which aligned well with the size of the input and output; each was 128 bits, which could be divided into four 32-bit words, which is the standard data width of the DMA. The block diagram of the hardware design is in Figure 8 below. AesAXIWrapper exposes one s_axilite bundle and two axis ports. These are connected to the PS via an AXI SmartConnect interconnect and an AXI DMA block for data transfers.

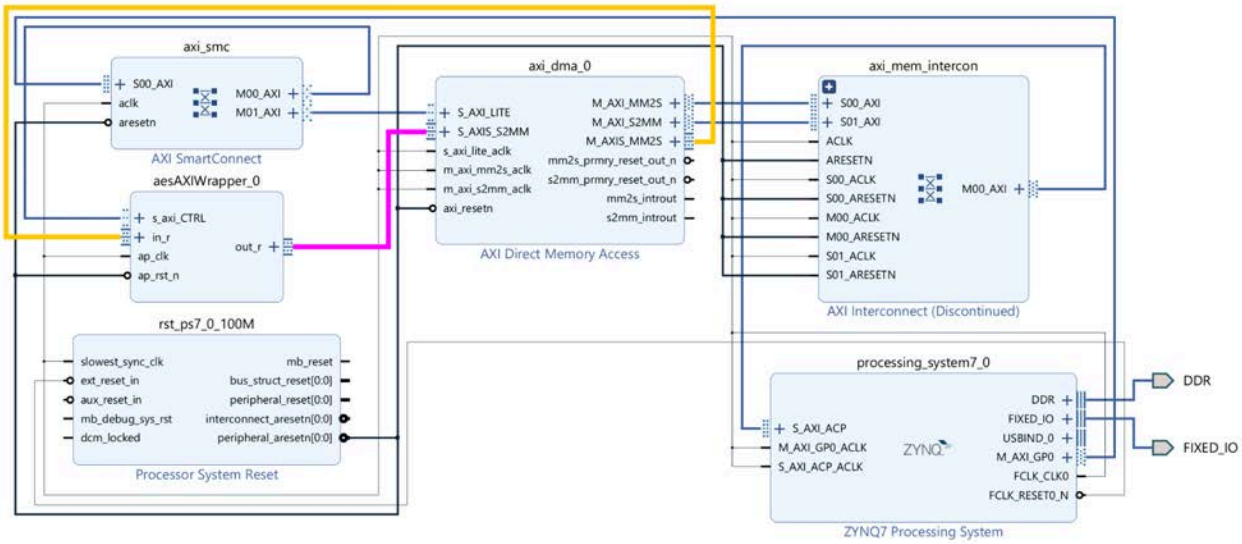


Figure 8: *Vivado block diagram of the AES hardware design.*

For the software components of this design, the team developed the core in Python within the Jupyter Notebook on the PYNQ-Z2 board. This had the benefit of a modular structure, which made testing and development more organized. The backend server was built with Python Bottle using JSON format, which was chosen for its lightweight integration with Jupyter and efficient communication between the frontend and backend. The frontend was designed using HTML, CSS, and Javascript, which were web development languages already familiar to the team.

Demonstration Summary

During the final in-class presentation, the team delivered a comprehensive and professional demonstration of the educational AES encryption platform. The system integrates a web-based user interface with hardware-accelerated encryption on a PYNQ-Z2 FPGA board, showcasing both technical functionality and instructional utility.

The demonstration successfully highlighted the platform's capability to perform AES-128 encryption via two distinct pathways: a software-based implementation written in Python and a hardware-accelerated implementation written in C and executed on the FPGA. Through the web interface, plaintext input was submitted by the user, processed in real time via a RESTful API, and the resulting ciphertext was returned to the frontend and displayed.

Key features showcased included the following: real-time communication between frontend and FPGA for encryption processing; accurate execution of the AES algorithm on both software and hardware backends; a responsive and accessible user interface designed for educational engagement; and visual explanations of the AES process and system architecture.

Although time limitations precluded a full walkthrough of the website, stakeholders were able to observe the core functionality of the system in action. A link to the public repository, which includes both the frontend and backend codebases, in addition to a video of a full user experience, is provided in Appendix A for further inspection.

Additionally, full-page screenshots of the user interface are provided in Appendix B to document the educational features and design elements of the website. This platform represents a significant outcome of the project and demonstrates the successful integration of hardware acceleration, user-focused design, and ethical software engineering principles.

Data and Procedures

The only data collection that occurred within this project was for timing analysis. To determine whether running AES encryption on hardware was faster than running it on software, Gianna developed a Python script that generates 100 randomized strings, times the software and hardware encryption of each, and averages the results. Figures 9 and 10 contain the randomized string generation, the call to the software encryption, and the data transmission for the hardware encryption. The timing data was acquired by enveloping the software and hardware encryptions in calls to perf_counter(), which generate "start" and "end" timestamps. The overall time for each encryption is determined by subtracting the "start" timestamp from the "end" timestamp.

```
# Arrays to hold timings
software_times = []
hardware_times = []
n = 100 # number of rounds


# Run the loop n times
for i in range(n):
    random_string = ''.join(random.choices(string.ascii_letters + string.digits, k=16))
    input_value = random_string
    data = input_value.encode('utf-8')



    # Time Software AES Encryption
    start = time.perf_counter()
    software_encrypt = aes_encrypt(input_value, key)
    end = time.perf_counter()
    software_times.append(end - start)
```

Figure 9: *Python code that generates 100 random strings and encrypts them with a software implementation of AES encryption. Each encryption is marked with a "start" and "end" timestamp, which determine its overall time.*

```
start = time.perf_counter()

dma.recvchannel.transfer(out_buffer)
dma.sendchannel.transfer(in_buffer)
dma.sendchannel.wait()
dma.recvchannel.wait()

end = time.perf_counter()
hardware_times.append(end - start)
```

Figure 10: *Python code that creates "start" and "end" timestamps around the data transmission for the hardware implementation of AES encryption. Each encryption is marked with a "start" and "end" timestamp, which determine its overall time.*

Results

The results of the AES encryption performance tests are presented in Figures 11 and 12. Figure 11 compares the execution time of hardware and software AES encryption over 100 iterations, while Figure 12 shows the average execution time for both implementations across these 100 iterations.

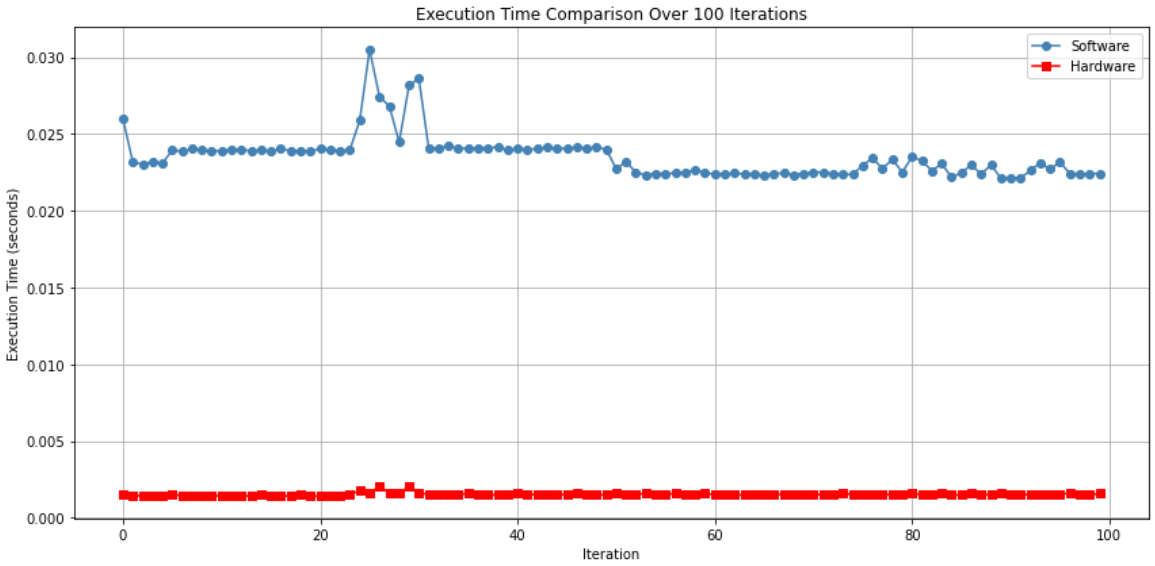


Figure 11: *Execution Time Comparison Over 100 Iterations.*

Figure 11 above illustrates the execution time of the AES encryption process, measured in milliseconds, for both the hardware and software implementations over the course of 100 iterations. The data shows that the hardware-based AES implementation significantly outperforms the software version in terms of time efficiency. Some inputs take more time to process overall, making the difference in performance more pronounced, with hardware encryption consistently taking less time to complete each iteration compared to software encryption.

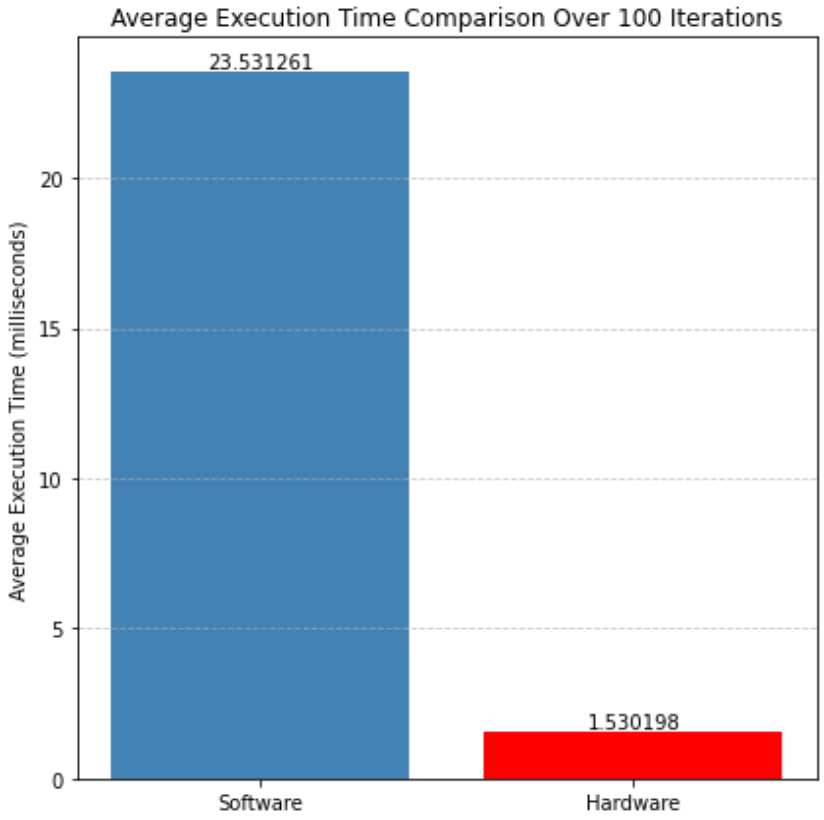


Figure 12: *Average Execution Time Comparison Over 100 Iterations.*

Figure 12 presents the average execution time for both hardware and software AES encryption across the 100 iterations. The average execution time for hardware encryption is noticeably lower than that of the software version, reinforcing the conclusion that the hardware-accelerated AES implementation provides substantial speed advantages. The results

show that, on average, the hardware implementation performs AES encryption 20 times faster, making it a more efficient solution for high-throughput applications.

Discussion

The primary objective of this project was to deliver an interactive, educational web platform that demonstrates hardware-accelerated AES encryption, complete with support for intermediate-result visualization and arbitrarily-sized data processing. Most of these goals were met: the website features several educational pages as well as a real-time, interactive encryption section, which accepts a 16-byte plaintext input, sends it to the FPGA for encryption, and returns the final ciphertext. While the team was unable to implement sending intermediate results of the various encryption steps or processing more than one 16-byte block of data within the time constraints of the project timeline, they did successfully accelerate AES using an FPGA.

Overall, the hardware implementation of AES encryption performed 20 times faster on average than the software implementation. Additionally, Table 1 and Figure 13 show the resource utilization of the entire hardware design. The design used only 7.3% of the LUTs and 4.76% of the flip-flops available on the FPGA, signalling that this is a highly efficient implementation of AES.

Resource	Utilization	Available	Utilization %
LUT	3886	53200	7.30
LUTRAM	226	17400	1.30
FF	5063	106400	4.76
BRAM	10	140	7.14

Table 1: *Resource utilization report of the hardware implementation.*

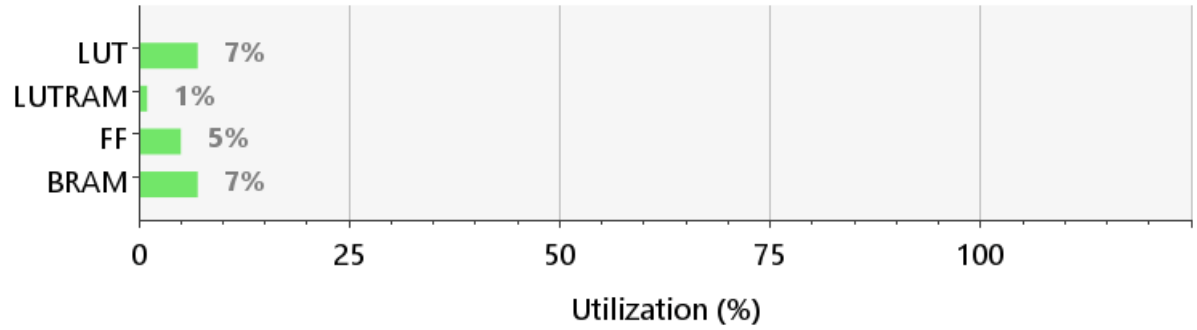


Figure 13: *Visualization of resource utilization percentages of the hardware implementation.*

This proven speed and efficiency are the main benefits of hardware acceleration, and there are numerous examples of AES usage where these benefits would make a significant impact. For example, critical patient data in an emergency care setting could be transmitted that much faster, potentially saving lives. In addition, the high efficiency allows for a more widespread impact since one FPGA would be able to contain more than one AES encryptor.

Our development methodology revealed various strengths and limitations. Working in Vitis HLS had the benefit of being able to develop in C, a language the team was more familiar with than SystemVerilog. Additionally, Vitis HLS pragmas for loop unrolling and array partitioning made it straightforward to optimize the timing as much as possible. However, Vitis HLS had an initial learning curve, and late-stage debugging proved challenging. After end-to-end communication was established, any changes made in Vitis HLS required repackaging the IP block, re-importing that block into Vivado, and regenerating the bitstream, which was time-consuming. Using Bottle for the backend server offered the most flexibility when developing the REST API, but the lack of documentation was a challenging hurdle to overcome. Developing the website frontend using HTML, CSS, and JavaScript was also straightforward since the team was familiar with those languages.

There were several key trade-offs that shaped the final project design. Packaging the entirety of AES encryption into one IP block simplified integration and allowed for maximum timing optimizations, but limited granularity and flexibility. In addition, the choice to hardcode the SBOX, RCON, and key removes the need to write to registers and keeps the key secure within the FPGA. However, any changes to the key require recompilation.

In comparison to peer capstone projects, this project's hardware accelerator similarly outperforms a software baseline. This project also used Vitis HLS over Vivado and delivered a full web-enabled pipeline, which was done by some of the other capstone teams. In addition, when comparing against other AES implementations, ours offers the same level of correctness and a decryption option.

Future work could address our remaining objectives: extending the AXI4-stream interface to support larger inputs, returning intermediate results back to the website for deeper educational impact, and allowing users to input their own AES key. Additionally, support for AES-192 and AES-256 could be added.

Reflecting on the semester, there were several aspects of the project that went well. Namely, the team worked out a system of dividing work that was highly successful. Given the different computer systems of the team members, it made sense to delegate the software work that could be completed remotely to the MacOS user and the hardware work to the Windows user. Each team member also put in an equal amount of work, making the team dynamic highly positive. In addition, while the project's goals evolved and changed throughout the semester, every milestone was passed successfully, and progress reports aided in ensuring progress was made incrementally throughout the semester. On the other side, improvements could have been made in time management and testing efforts. While progress was made over the course of the

semester, it could have been made more consistently on a day-to-day basis to avoid unnecessary stress. Additionally, initial effort was made to build a test suite to robustly test the correctness of the team's implementation of AES, but the test suite was not successfully implemented due to time constraints. Overall, this project deepened the team's practical skills in software-hardware co-design and taught lessons in teamwork, best development practices, and the ability to pivot and adapt to situations. In hindsight, the team would have approached this project with more research and a more specific initial design. Because this project was larger in scope and contained unfamiliar components, it would have been beneficial to do more research on the parts that the team was not as familiar with so that the initial design could have been more thought out. That way, the pivots and adjustments made to the project may not have been necessary.

Conclusion

The implementation of AES encryption with hardware acceleration has proven to be an effective method for optimizing performance. Through the comparative analysis of hardware and software implementations, it is clear that hardware acceleration significantly reduces execution time, offering a notable advantage in high-throughput applications. This project successfully demonstrated the practical benefits of hardware-based AES encryption, aligning with the project's goals to improve cryptographic efficiency. These findings underscore the importance of leveraging hardware acceleration for computationally intensive tasks, such as encryption, where speed is a critical factor.

While the current implementation meets the basic project requirements, there are several areas for future development. One key improvement would be the ability to handle larger, arbitrarily sized inputs, which would enable more comprehensive timing analysis and

demonstrate how the system scales with increasing data sizes. Additionally, integrating the capability to display intermediate AES encryption steps in real-time on the webpage would provide an interactive and educational tool for users to better understand the encryption process. Furthermore, accommodating different AES key sizes, such as 128, 192, and 256 bits, would enhance the flexibility and applicability of the system, making it suitable for a wider range of security needs. Addressing these areas would pave the way for a polished, production-ready product that is scalable, adaptable, and capable of meeting diverse cryptographic requirements.

Deliverables

All deliverables for this project were organized into three milestones, which can be viewed in the project GANTT chart in Appendix C.

For Milestone 1, the proposed deliverables included a basic website frontend supporting user input, a working REST API, and bi-directional data transfer between the microprocessor and the FPGA on the board, resulting in a simple computation being correctly completed on the FPGA. During the demonstration, the team delivered a website frontend on which users could input a positive number to be multiplied by two. The user input data was successfully sent between the microprocessor and the FPGA, where the number was multiplied by two, and sent back to be displayed on the website frontend. A log of previous requests was also displayed.

Milestone 2 focused on integrating several AES encryption substeps into the communication framework created for Milestone 1. The proposed deliverables included full Python implementations of both AES encryption and decryption, implementations of the SubBytes, AddRoundKey, and key expansion steps within Vivado, an updated website frontend that accepted a max-16 byte long plaintext, and the same bi-directional data transfer

implemented in Milestone 1. It was also proposed that the results of each encryption step developed in Vivado would be displayed on the website frontend. However, these deliverables changed during development. The Python-based encryption and decryption were completed as proposed, and the website and REST API were updated to accept plaintext and display the ciphertext output. Regarding the hardware, the decision was made to deliver a complete AES encryption implementation in Vitis HLS. Milestone 1 was completed using Vitis HLS for the hardware, and since it was de-risked, it made sense to make the switch. Additionally, attempting to execute the separate data transfers for each encryption step began to delay development, so rather than demonstrating half of the steps with full communication, the team decided to demonstrate a full implementation of AES with no communication. During the demonstration, the team compared the outputs from the Vitis HLS implementation, the Python implementation, and a third-party AES encryption website to verify the correctness.

The final milestone aimed to transform the existing system into an educational platform that teaches users the individual steps of AES encryption. The proposed deliverables included a fully interactive and educational website with user input, where the intermediate results of encrypting the inputted plaintext would be displayed as an example on each section of the page, completion of the AES encryption flow in Vivado, and a bi-directional data transfer system that sends the plaintext to the FPGA for encryption and sends the results of each encryption step back to the website. Also proposed was the ability to take in and encrypt arbitrary-sized data. Like for Milestone 2, these deliverables also changed during development. The hardware AES flow was already completed in Vitis HLS, so the focus was on establishing the data transfer of the intermediate results as well as the data transfer of arbitrarily large data. As development began on these features, there were several setbacks that led to the decision to limit the data size to 16

bytes and only send the final ciphertext back to the website frontend. During the demonstration, the team delivered a multi-page educational website, including a page that contained the interactive encryption experience. The user-inputted plaintext was properly sent to the FPGA for encryption, and the ciphertext was returned for display on the website. The final completed deliverables can be found in detail in Appendix B.

Schedule

Figure 14 below contains the GANTT chart outlining the timeline for completion of this project. Larger images of each section are available in Appendix C. While this initially seemed like an appropriate timeline for this project, we deviated from it significantly after Milestone 1. For Milestone 1, we were able to adhere to the timeline as expected. For Milestone 2, we planned to implement the steps of AES encryption one-by-one, and this timeline reflects that proposed trajectory. As we began to work on Milestone 2, however, there were several changes we made to our project that altered this timeline. As aforementioned, the initial goal was to send the intermediate results of the encryption sub-steps to the processing system to be displayed on the website frontend. However, this proved to be more difficult than initially expected, so we pivoted to fully focusing on completing the C implementation of AES encryption that would become the hardware IP block. For Milestone 2, instead of demonstrating communication between the processing system and the FPGA for half of AES encryption's substeps, we demonstrated the entirety of AES encryption in C. The software tasks, such as the Python implementation of AES encryption and decryption, REST API functionality, and a working frontend website were completed as scheduled for Milestone 2. The remaining third of the project, therefore, was focused on implementing the communication between the processing system and the FPGA.

While these adjustments led to significant deviations from our initial project timeline, we adhered fairly well to the alternate timeline. There were minor setbacks throughout the semester due to illness or outside commitments that contributed to delayed progress on a weekly level, but both group members made consistent progress toward the redefined goals.

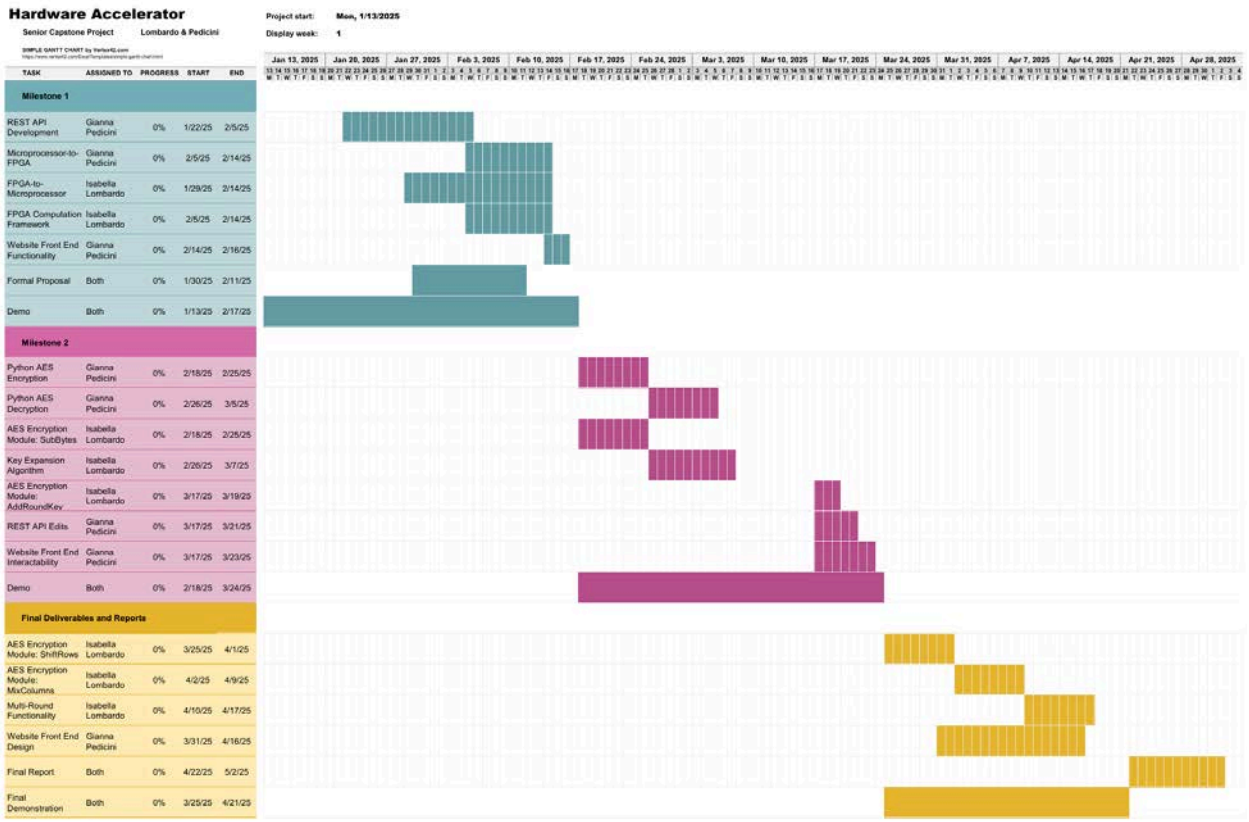


Figure 14: Proposed GANTT chart for the AES hardware acceleration project.

The tasks of this project were distributed evenly between the group members. Gianna was responsible for the following software-oriented tasks: creating and designing the website frontend, developing the website backend, implementing the REST API, completing the timing analysis, and developing the Python implementation of AES encryption and decryption in the Jupyter Notebook on the PYNQ-Z2 board. Isabella was responsible for the following

hardware-oriented tasks: developing and packaging the C implementation of AES in Vitis HLS; wiring together the ZYNQ7 Processing System and Reset, the AES IP Block, and the various AXI components to develop the overlay for the PYNQ-Z2 board; and developing the Python code for the AXI4 communication stream between the Jupyter Notebook and the FPGA. Both group members contributed to the formal proposal, final presentation, and final report equally, and both group members contributed equally to the testing and debugging process.

References

"Advanced Encryption Standard." *Wikipedia*, Wikimedia Foundation, 17 Mar. 2025, en.wikipedia.org/wiki/Advanced_Encryption_Standard.

"AES Key Schedule." *Wikipedia*, Wikimedia Foundation, 26 Apr. 2025, en.wikipedia.org/wiki/AES_key_schedule.

Awati, Rahul. "Advanced Encryption Standard (AES)." *TechTarget*, 2022, https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard.

"Cost of a Data Breach Report 2024." *IBM Security and Ponemon Institute*, 2024, https://www.ibm.com/reports/data-breach.

Dandass, Yoginder, et al. "Hardware Acceleration of AES Using Field Programmable Gate Arrays." *Proceedings of the 2008 IEEE SoutheastCon*, IEEE, 2008, pp. 479–484. https://doi.org/10.1109/SECON.2008.4494322.

Esslinger, Bernhard. "Teaching Cryptology at All Levels Using CrypTool." *Proceedings of the 15th Colloquium for Information Systems Security Education*, 2011, https://www.cryptool.org/media/publications/journals/teachingcryptool.pdf.

Appendices

Appendix A: Helpful Links for Demonstration

Github Link: https://github.com/giannaped8/AES_FPGA.git

Video Demonstration: Full AES Encryption Website Walk Through

Appendix B: Website Screenshots

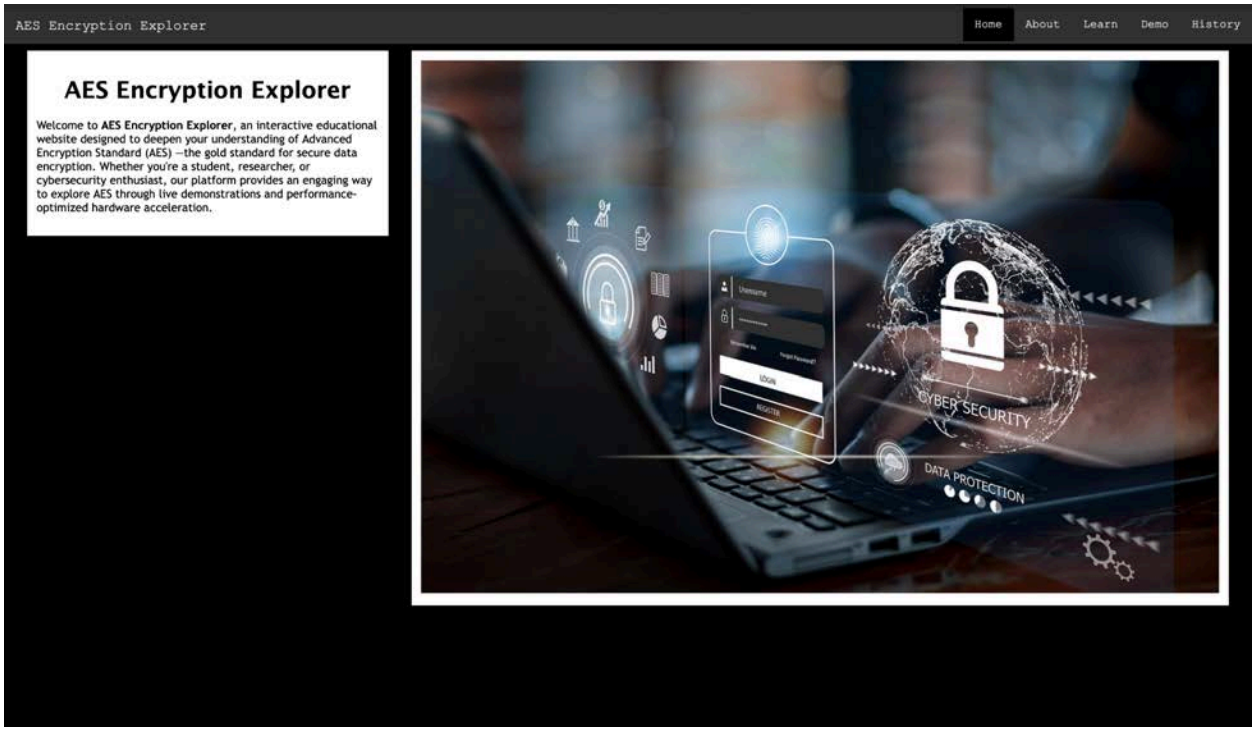


Figure B1: *Home Page (Entire page)*

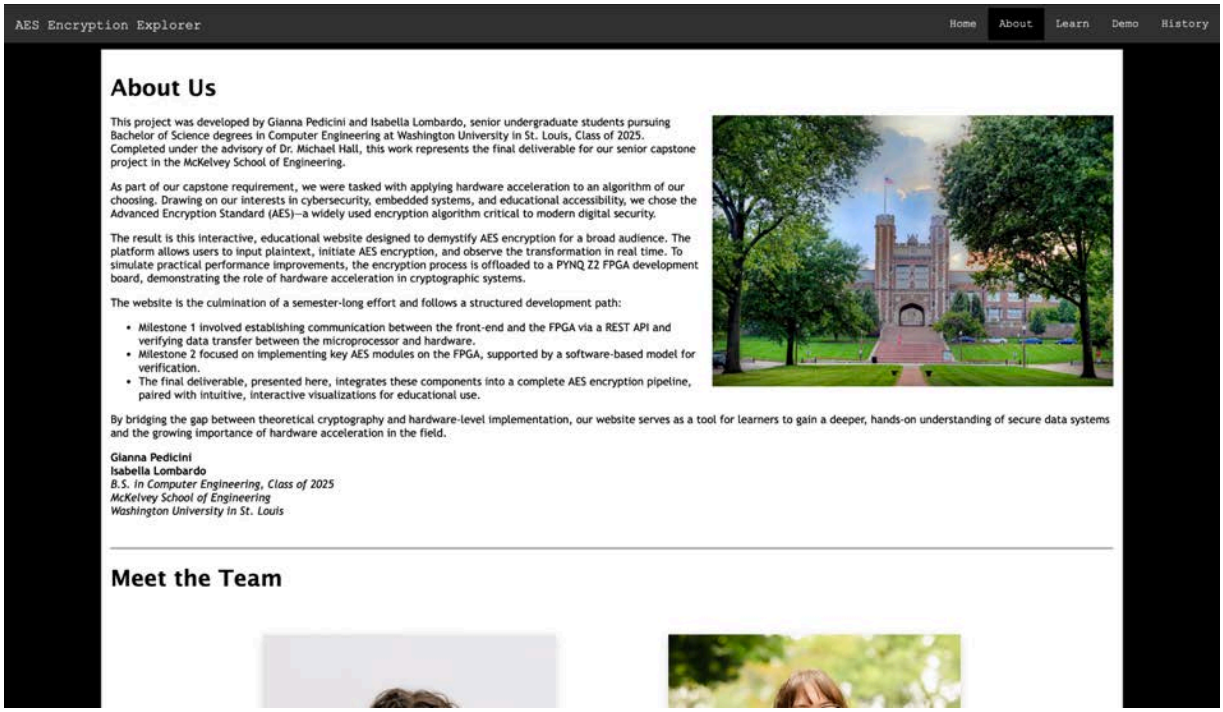


Figure B2: *About Page (Top Portion of Page)*

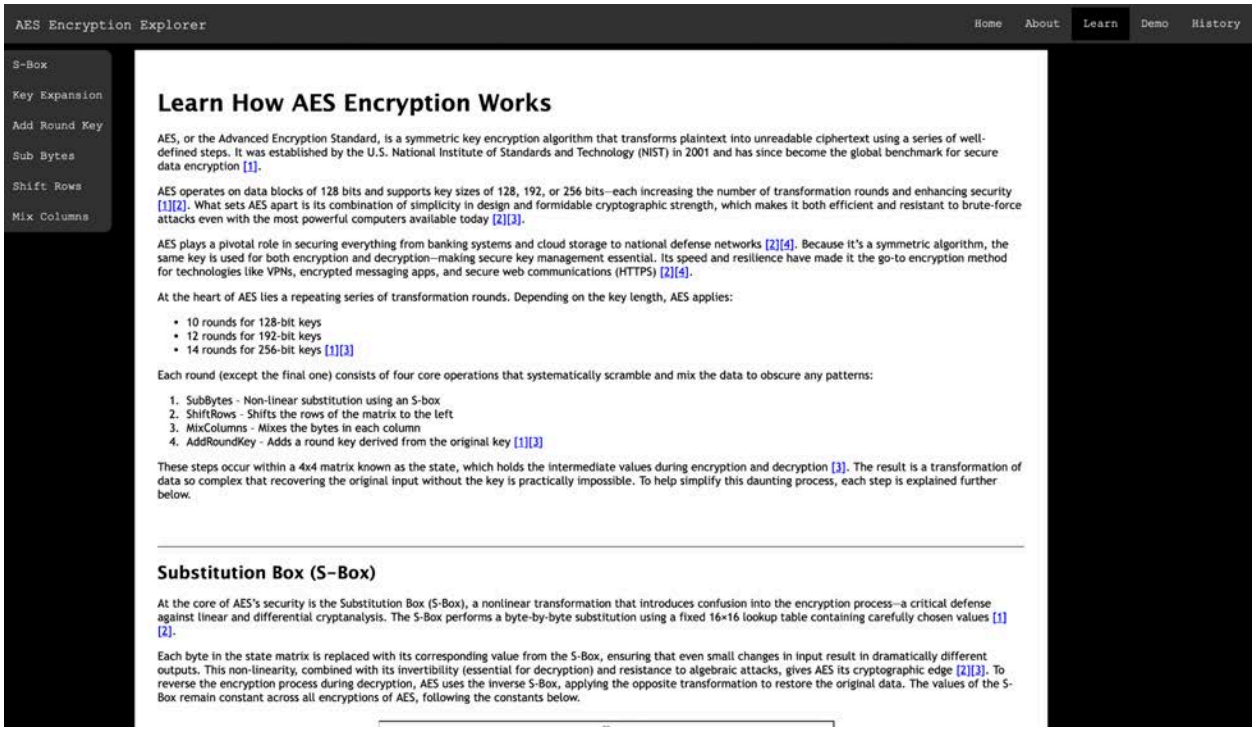


Figure B3: *Learn Page (Top Portion of Page)*

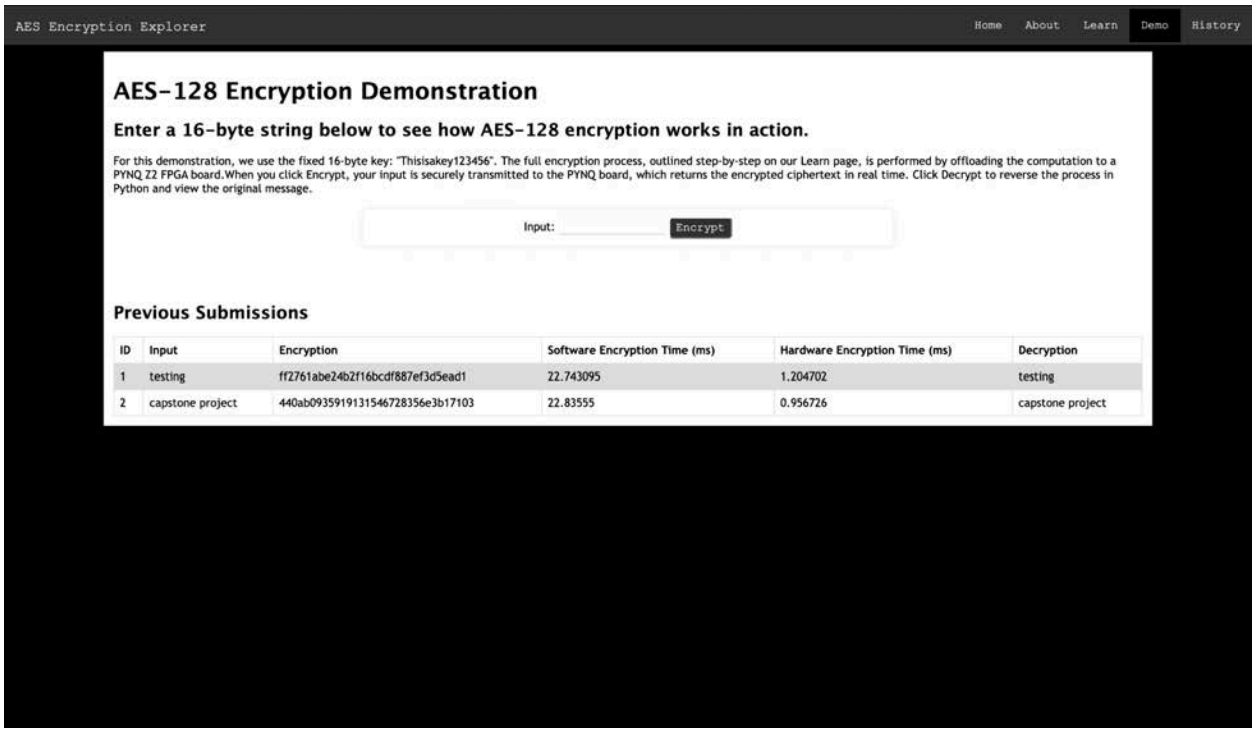


Figure B4: *Demo Page (Entire Page)*

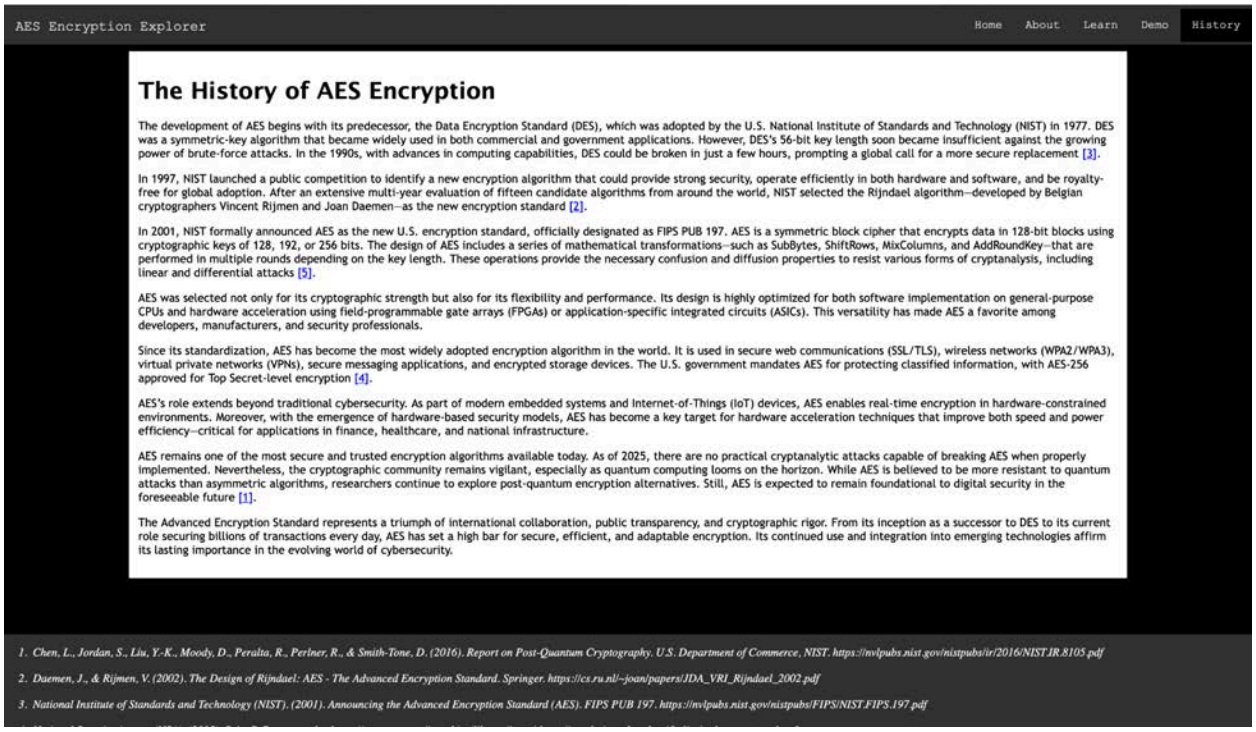
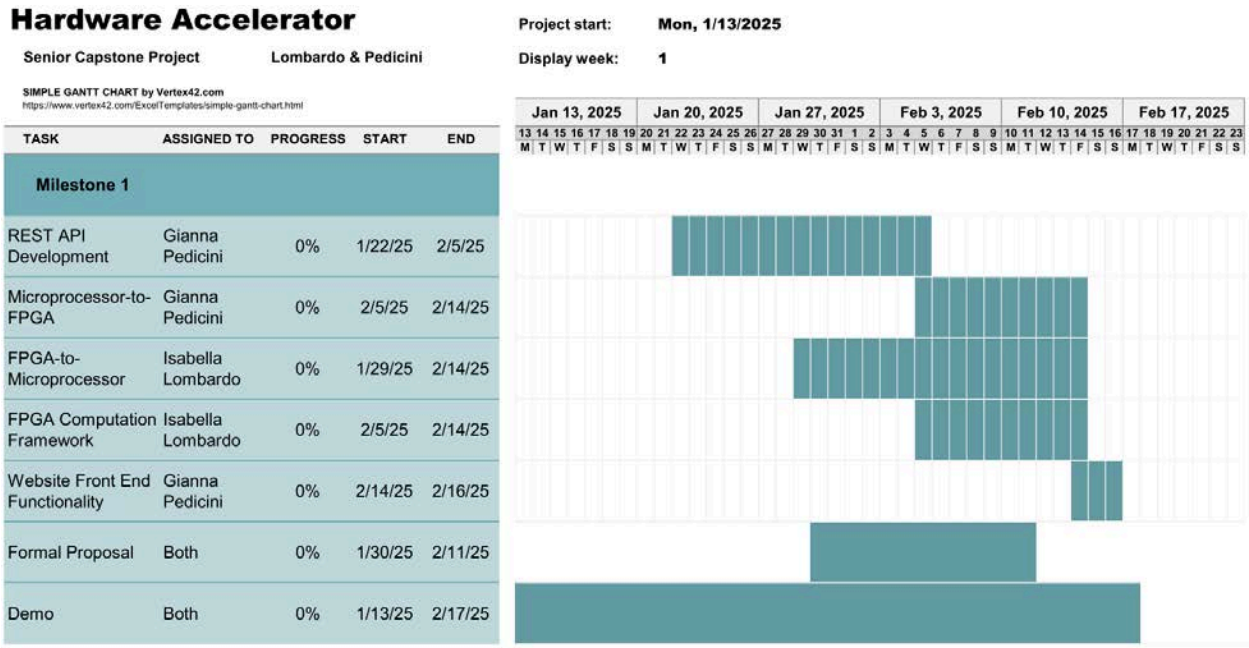


Figure B5: *History Page (Top Portion of Page)*

Appendix C: Additional GANTT Chart Images



Hardware Accelerator

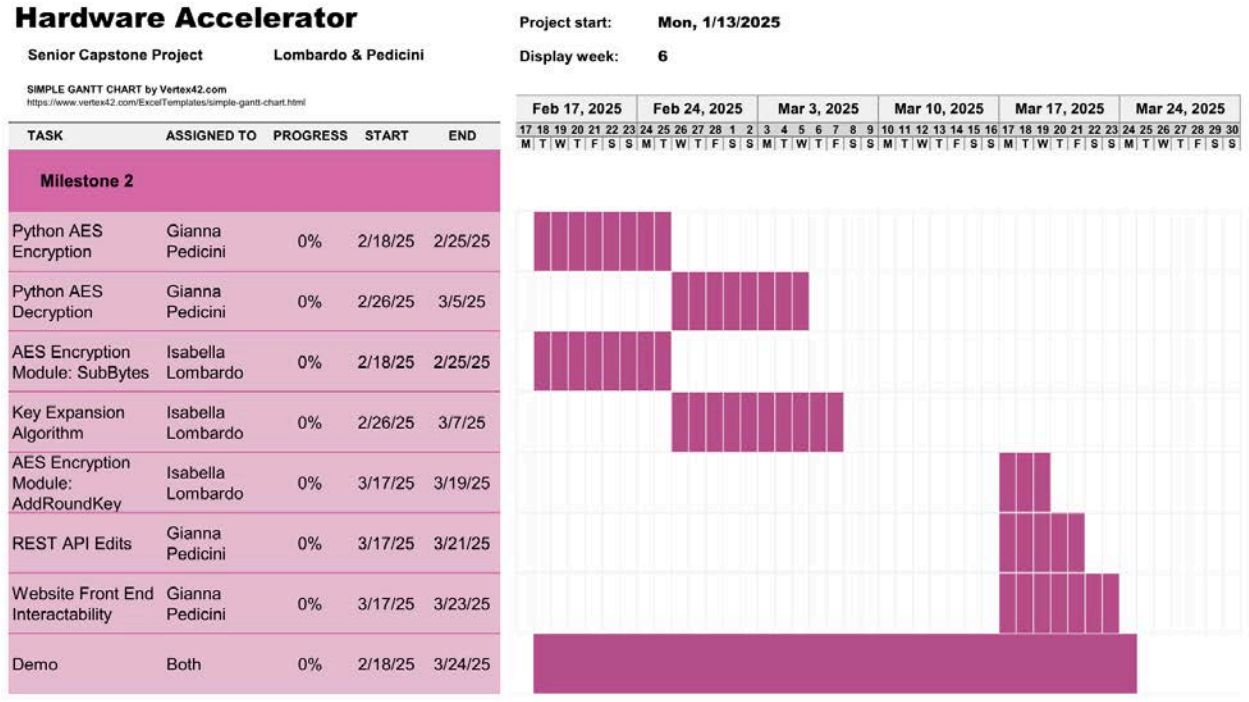
Senior Capstone Project — Lombardo & Pedicini

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Project start: Mon, 1/13/2025
Display week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Milestone 1				
REST API Development	Gianna Pedicini	0%	1/22/25	2/5/25
Microprocessor-to-FPGA	Gianna Pedicini	0%	2/5/25	2/14/25
FPGA-to-Microprocessor	Isabella Lombardo	0%	1/29/25	2/14/25
FPGA Computation Framework	Isabella Lombardo	0%	2/5/25	2/14/25
Website Front End Functionality	Gianna Pedicini	0%	2/14/25	2/16/25
Formal Proposal	Both	0%	1/30/25	2/11/25
Demo	Both	0%	1/13/25	2/17/25

Figure C1: *The "Milestone 1" portion of the proposed project GANTT chart*



Hardware Accelerator

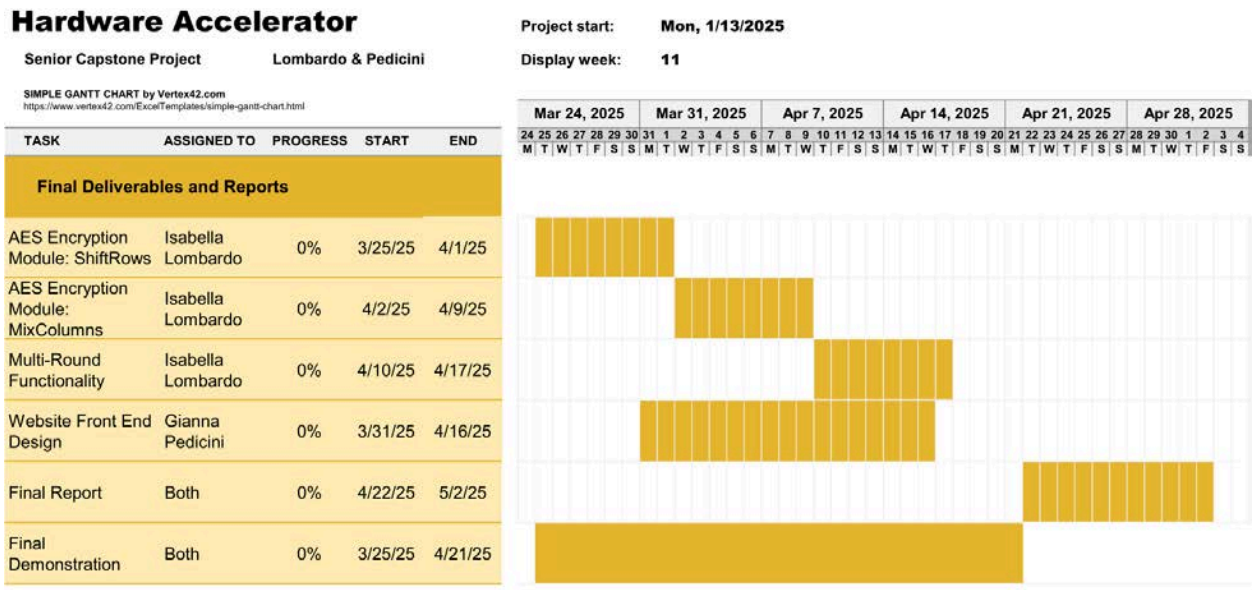
Senior Capstone Project — Lombardo & Pedicini

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Project start: Mon, 1/13/2025
Display week: 6

TASK	ASSIGNED TO	PROGRESS	START	END
Milestone 2				
Python AES Encryption	Gianna Pedicini	0%	2/18/25	2/25/25
Python AES Decryption	Gianna Pedicini	0%	2/26/25	3/5/25
AES Encryption Module: SubBytes	Isabella Lombardo	0%	2/18/25	2/25/25
Key Expansion Algorithm	Isabella Lombardo	0%	2/26/25	3/7/25
AES Encryption Module: AddRoundKey	Isabella Lombardo	0%	3/17/25	3/19/25
REST API Edits	Gianna Pedicini	0%	3/17/25	3/21/25
Website Front End Interactability	Gianna Pedicini	0%	3/17/25	3/23/25
Demo	Both	0%	2/18/25	3/24/25

Figure C2: *The "Milestone 2" portion of the proposed project GANTT chart*



Hardware Accelerator

Senior Capstone Project — Lombardo & Pedicini

Project start: Mon, 1/13/2025

Display week: 11

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

TASK	ASSIGNED TO	PROGRESS	START	END
Final Deliverables and Reports				
AES Encryption Module: ShiftRows	Isabella Lombardo	0%	3/25/25	4/1/25
AES Encryption Module: MixColumns	Isabella Lombardo	0%	4/2/25	4/9/25
Multi-Round Functionality	Isabella Lombardo	0%	4/10/25	4/17/25
Website Front End Design	Gianna Pedicini	0%	3/31/25	4/16/25
Final Report	Both	0%	4/22/25	5/2/25
Final Demonstration	Both	0%	3/25/25	4/21/25

Figure C3: *The "Final Deliverables and Reports" portion of the proposed project GANTT chart*

Appendix D: S-Box Values

```
static const unsigned char SBOX[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16};
```

Figure D1: *The S-Box values.*