Reconfigurable Arduino Crypto FPGA Shield

Team Members
Dustin Schnelle
Gomathy Venkata Krishnan
Meiqi Zhao
Ryan Bornhorst

Faculty Advisor
Dr. Christof Teuscher

Sponsor Company & Members
Galois
Joe Kiniry
Dan Zimmerman

Table of Contents

1. Summary **3**

2. Introduction & Motivation **3**

2.1 Background 3

2.1.1 Software Flow Diagram 5

2.1.2 Encryption (AES) 5

2.2 Motivation 6

2.2.1 Why use an FPGA 6

2.2.2 Why use Formal Methods 6

3. Problem & Proposed Solutions **6**

3.1 Problem 6

3.2 Proposed solutions 7

3.3 Steps Taken 7

4. Analytical Methods, Tools, & Theory 7

4.1 Methods 7

4.1.1 Software Testing (C version) 7

4.1.2 Hardware Testing 8

4.2 Tools Used 9

4.3 Development Boards 9

4.3.1 Mojo v3 9

4.3.2 Papilio One 250K 9

4.3.3 PulseRain M10 10

4.3.4 Nexys DDR4 10

4.4 Theory 11

5. Results 11

6. Conclusion **13**

7. Project Schedule **13**

8. Testing Procedures **18**

8.1 Testing Software Benchmark Files 18

8.2 Testing on Vivado HLx 18

8.3 Testing on Vivado Microblaze 18

9. Overall Project Strengths & Weaknesses **19**

9.1 Project Strengths 19

9.2 Project Weaknesses 19

9.3 Challenges Faced 19

9.3.1 Cryptol and SAW: 19
9.3.2 OpenSSL Benchmarking: 19
9.3.3 Vivado Synthesis: 20
10. Future Work **20**
11. Bibliography **21**

1. Summary

Our motivation for creating a Reconfigurable Arduino Crypto FPGA Shield is the growth in the embedded systems market. Systems like the Arduino rarely consider security as a first priority. Unsecure systems are a problem when it comes to transmitting sensitive information. In order to secure embedded systems like these we propose using an FPGA shield. An FPGA shield used a Field Programmable Gate Array that can be attached to one of these embedded boards and reconfigured to meet the needs of the system. By implementing a cryptographic API through the FPGA we can dedicate hardware to secure the system. Hardware is better than software because it limits user interactions and doesn't use any of the systems resources to encrypt data into or out of the system.

One problem with cryptographic APIs is that they are usually handwritten and rarely verified to be correct regarding the mathematical specification they are implementing. This can also be a major security flaw in a system. Therefore we are using a formally verified Verilog implementation of AES-128 generated through tools like Cryptol and SAW to guarantee that our hardware implementation is correct. These tools can be used to test mathematical properties as well as prove correctness and equivalence to the formal specification.

Another reason to use hardware, such as an FPGA for encryption is that it can outperform most software implementations. To prove this we tested throughput of the AES-128 algorithm on various CPUs, MCUs, and FPGAs and compared their results. Our results were able to show that FPGA implementations were at least hundreds of times faster than the software implementations and in comparison to some of the embedded MCUs, they were millions of times faster. Therefore a formally verified version of AES-128 using an FPGA is going to give the best results in terms of security and performance.

2. Introduction & Motivation

2.1 Background

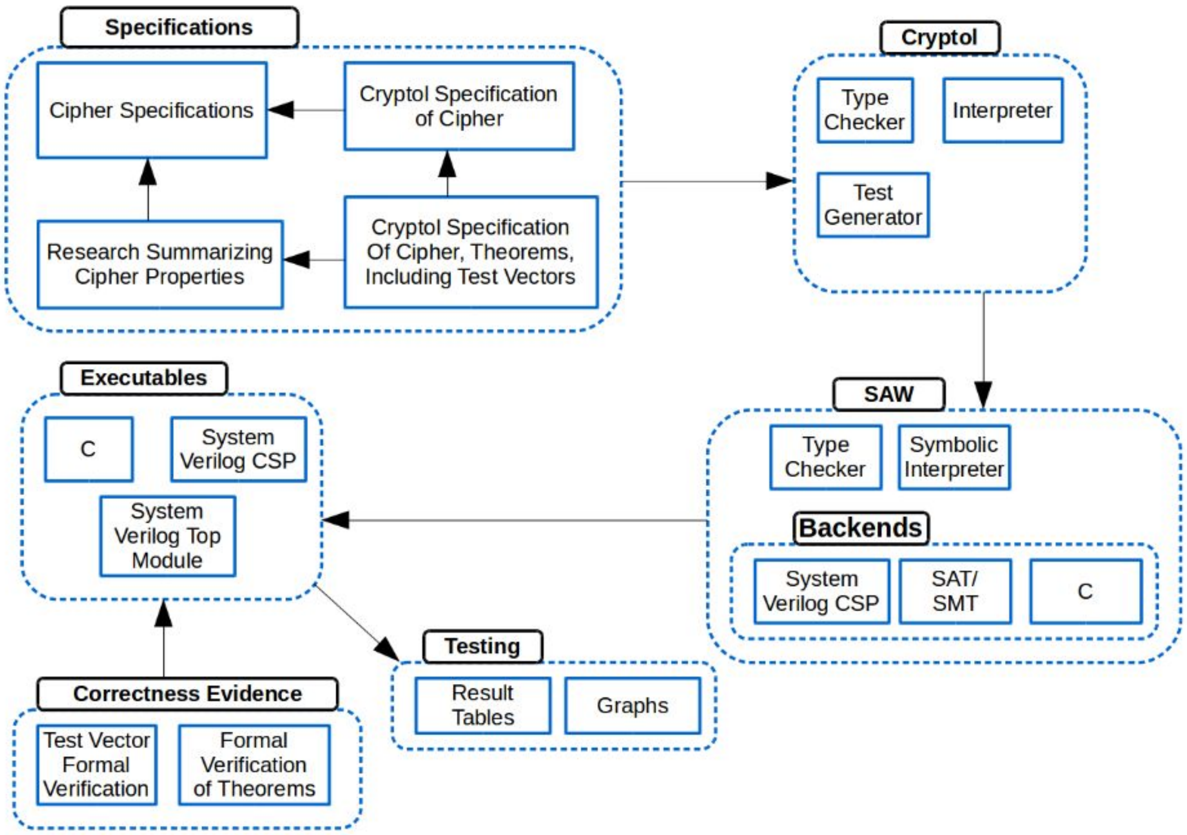
Inexpensive open-source hardware platforms, such as the popular Arduino , have had a huge impact on the embedded systems industry. Especially in the world of the Internet-of-Things (IoT) (Burrus, 2014). Arduino's open-source microcontroller platform has become popular to both hobbyist and educators alike because of the easy-to-use hardware and software interface. The low cost of the Arduino platform also helps as well. As Arduino has become more popular in the realm of academics, the amount of documentation for the Arduino has greatly increased to provide even more support. There are already two courses taught at the University of California that focus on teaching the Arduino platform, its programming environment, and interfacing principles for an IoT specialization (El-Abd, 2017).

With the massive growth of the IoT the world is changing, allowing for the innovation of new designs and embedded devices within the home. This growth in devices depending on a connection to the internet means that we need to focus on security as a first class component when transmitting sensitive data over a network. It's becoming increasingly necessary to ensure the security of embedded systems connected to the network, so adding cryptography (Kessler, 2018) to the functionality and security.

One solution to the problem is using an FPGA shield to perform the encryption and add to the functionality of the Arduino. The current Arduino FPGA shield's on the market however don't guarantee high assurance (Free & Fair, 2016) of the software to provide objective evidence of the system's correctness and security. Usually this evidence comes from a hand-written test bench, but we are looking to obtain this evidence from formal assurance or what is known as formal methods (Freedberg Jr., 2016). Many of the systems that are important to national security to undergo rigorous certification and evaluation regimes, but some systems require stronger evaluation and certification and evaluation regimes, but some systems require stronger evaluation and certification to provide guarantees and mathematical evidence that they function exactly as intended at all times.

In the past there have been a few attempts of creating a shield for the Arduino to add the ability to perform encryption functions on the Arduino platform like the CryptoShield developed by SparkFun. Shields like the CryptoShield use dedicated chips to perform functions like SHA (Landman, Ross and Williams, n.d.). It also featured a dedicated chip to handle HMAC (Krawczyk, Bellare and Canetti, 1997) which may be used to simultaneously verify both the data integrity and the authentication of a message, but the shield doesn't guarantee high-assurance. The fact that it uses dedicated chips also means that the functionality of the encryption features can't be reconfigured like they could with the use of an FPGA. The introduction of FPGA based encryption would allow for flexibility of the algorithms being implemented. The implementation of an FPGA Arduino shield is not the first of its kind as it was once for a Hackaday project, but this FPGA shield focused on expanding the overall functional of the Arduino and doesn't focus on cryptography or mention it. Even then the shield doesn't guarantee high-assurance of crypto algorithms, so this means that there is a major need for a reconfigurable FPGA based Arduino shield when it comes to dealing with the ever expanding IoT.

2.1.1 Software Flow Diagram



2.1.2 Encryption (AES)

Encryption is the process of making sensitive data more secure and less likely to be intercepted by anyone unauthorized to view the data. For the entirety of the project we used AES, which is known as the Advanced Encryption Standard (Csrc.nist.gov, 2001). AES is a symmetric encryption, which means that the same key is used for both encryption and decryption. The standard specifies the Rijndael algorithm (Selent, 2010), which uses a symmetric block cipher to process fixed size data blocks. AES uses keys with lengths of 128-bits, 192-bits, and 256-bits. This produces three different types of the standard AES-128, AES-192, and AES-256.

The standard encryption uses AES-128 where both the block and key size are 128-bits. The size of the block is commonly denoted by Nb and the key size is commonly denoted as Nk. The algorithm uses a specified number of rounds to transform the data for each block. The initial block is added to an expanded key derived from the initial cipher key. Then the next round consists of operations of the S-box, shifts, and a MixColumn. The results is added to the next expanded key and when all of the rounds are finished the final result is an encrypted cipher block.

2.2 Motivation

2.2.1 Why use an FPGA

FPGAs are the top choice when it comes to this application since they are reconfigurable, low cost and have adequate resources to process data. In this project, we use FPGAs rather than microcontrollers, microprocessors and GPUs because FPGAs can be programmed and debugged easily.

There are three main reasons for using an FPGA to implement the crypto algorithm functions:

(a) The performance of software crypto on inexpensive Arduino devices is poor.
(b) The threat model for hardware crypto is significantly different than that for software-only crypto.
(c) Actually running the crypto algorithm implementation on hardware will provide accurate information about the processing speed and throughput of the FPGA.

Creating an FPGA shield compatible with an Arduino can help process encryptions. It will eventually become a popular prototype that can be deployed commercially to perform encryption and benchmark crypto algorithms automatically.

2.2.2 Why use Formal Methods

Most of the crypto software libraries found online are manually written and are susceptible to security leakages. Creating high-assurance cryptographic algorithms would be the solution to this problem and would ensure no security breaches or flaws. High-assurance means that the implementation of software, firmware and hardware in the crypto FPGA shield includes objective evidence of the system's correctness and security. Tools to support rigorous system engineering can be used so that everyday software and hardware engineers can use hidden formal methods.

Formal methods are the application of mathematics to systems design, engineering, validation and verification to create systems that are proven to be correct and secure.
Tools like Cryptol and SAW are thus used to automatically generate high-performance high-assurance cryptographic algorithm implementations in C and Verilog.

3. Problem & Proposed Solutions

3.1 Problem

Crypto code found today is either ad hoc or is manually written. This makes it very insecure. There are no dedicated embedded systems that verify security. Arduino and other inexpensive hardware platforms in addition to a device that can process crypto code can be a huge breakthrough in this day and age. Software only crypto verifications exist but an affordable and efficient hardware model to perform encryptions that provide formal and rigorous assurance is not available in this Internet of Things era.

3.2 Proposed solutions

Some options to creating a hardware platform for testing if the crypto algorithm is correct and secure includes the following:

- High assurance crypto FPGA shield for Arduino
- Verification of crypto algorithms on GPU

3.3 Steps Taken

- Compare the performance of different platforms by running benchmark files and decide which chip or system to use.
- If a shield will be created then look up embedded boards and understand routing of different components on board. Also, ensure pins that coincide with Arduino IO pins are mapped correctly.

4. Analytical Methods, Tools, & Theory

4.1 Methods

4.1.1 Software Testing (C version)

For testing AES-128 software on the CPUs and MCUs we chose to use OpenSSL encryption library written in C. We used their speed benchmark to generate a report that showed us the total bytes encrypted in one second for a block size of 16. There was no standard benchmark file written for the MCUs so we just copied over the same files needed to run them on the CPUs. We then used these header and C files to duplicate our own version of the benchmark on the Arduino, Mojo, and Papilio development boards. The results from these speed tests were used to calculate the overall throughput and compared against the FPGA implementations.

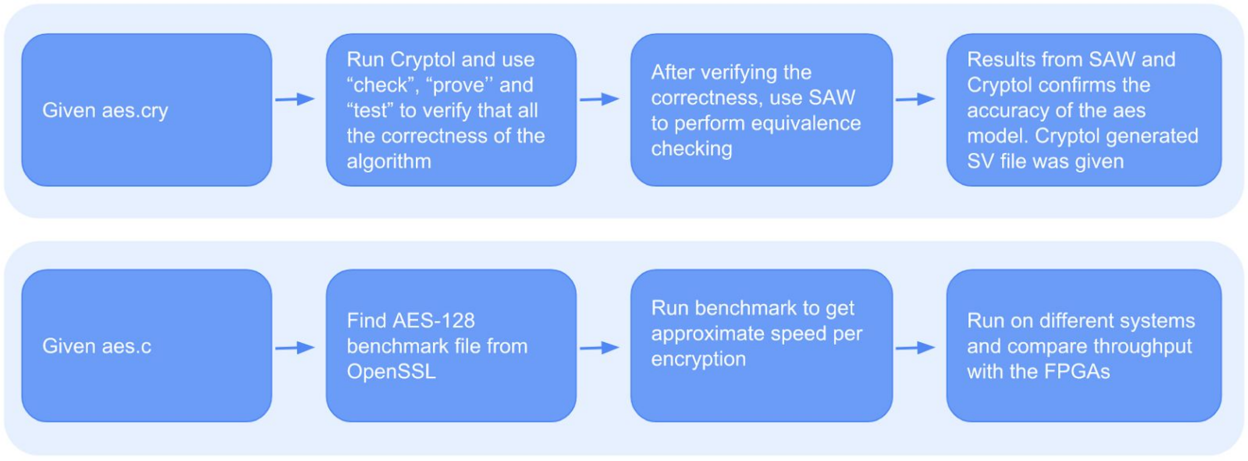

Given aes.cry
Run Cryptol and use “check”, “prove” and “test” to verify that all the correctness of the algorithm
After verifying the correctness, use SAW to perform equivalence checking
Results from SAW and Cryptol confirms the accuracy of the aes model. Cryptol generated SV file was given
Given aes.c
Find AES-128 benchmark file from OpenSSL
Run benchmark to get approximate speed per encryption
Run on different systems and compare throughput with the FPGAs

Figure 1: Verification and Software Testing Flow Diagrams

4.1.2 Hardware Testing

Loading the bitstream onto the FPGAs and observing whether the plain text is encrypted to cipher text is the way to verify that the given AES implementation is correct. The maximum throughput, resource utilization and timing reports would be used as methods to benchmark the results and compare the performances.

The top module given to us isn't synthesizable as it contains channels, interfaces and 128 bit vectors. Since the verilog project does not have a constraints file (.EDIF or .XDC), pin mapping and floorplanning fails. We therefore followed the below process to load and test the AES implementation given to us.

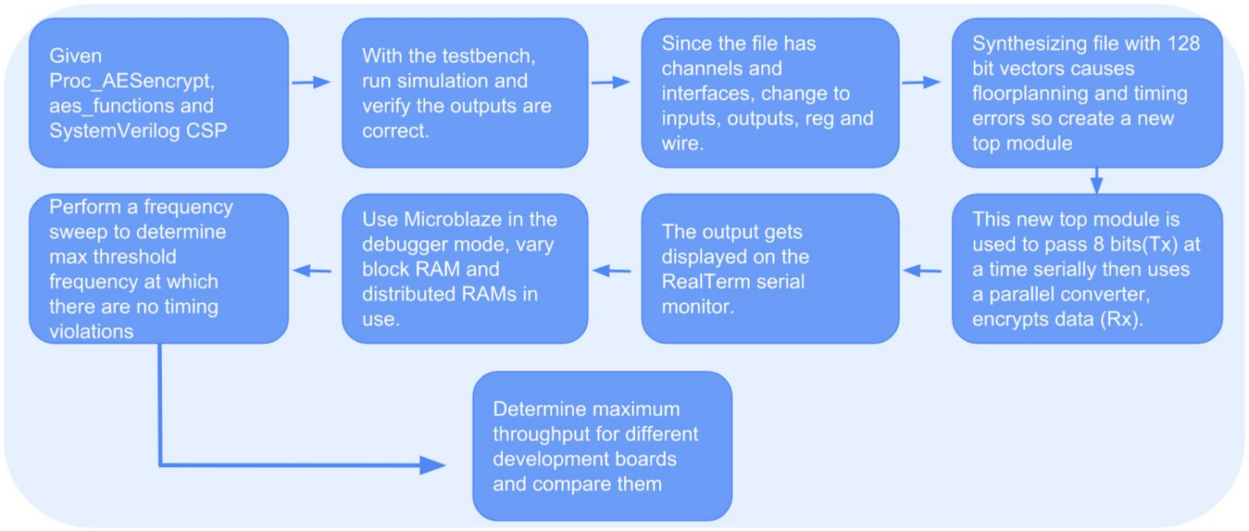

Given Proc_AESencrypt, aes_functions and SystemVerilog CSP
With the testbench, run simulation and verify the outputs are correct.
Since the file has channels and interfaces, change to inputs, outputs, reg and wire.
Synthesizing file with 128 bit vectors causes floorplanning and timing errors so create a new top module
This new top module is used to pass 8 bits(Tx) at a time serially then uses a parallel converter, encrypts data (Rx).
The output gets displayed on the RealTerm serial monitor.
Use Microblaze in the debugger mode, vary block RAM and distributed RAMs in use.
Perform a frequency sweep to determine max threshold frequency at which there are no timing violations
Determine maximum throughput for different development boards and compare them

Figure 2: Verilog Synthesis Flow Diagram

We had different boards with different FPGAs on them and this meant different tools had to be used to synthesize, implement and generate the bitstream. We used the following tools for these FPGAs:

FPGA	Tools
Xilinx series 6 and below : Spartan 6 and Spartan 3E	Xilinx ISE and Yosys synthesizer
Xilinx 7 series: Artix 7	Xilinx HLx and Microblaze
Intel M10	Quartus

Figure 3: Tools Required for Different FPGA Platforms

Synplify and other synthesizers were experimented with in the process. Yosys synthesizer has a feature to optimize and edit the bitstream file to make it compatible with different FPGAs. This tool was used to create a bitstream for the Mojo v3 board. Xilinx ISE can be used to upload the bitstream.

Microblaze was used in addition with the Digital Clock Manager to perform the following steps:

- Find the maximum throughput at default frequency
- Find the maximum frequency until which there will be no timing violations (frequency sweep)
- Display timing reports and an interactive implementation schematic that will show resource utilization, critical path and LUTs in use.
- Display messages when errors are encountered (incorrect cipher text)

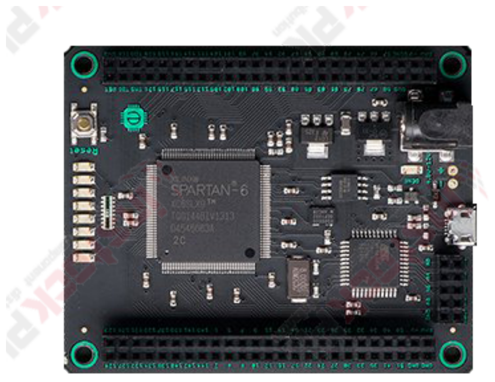
4.2 Tools Used

The tools used for this project involved software verification tools such as Cryptol and SAW. We used the OpenSSL cryptography API to implement AES-128 on the CPUs and MCUs. The CPUs tested on were the Intel Xeon CPU E3 and the AMD A10-5800K. The MCUs tested on were the ATmega32U4 and the ATmega328P. We used three different development boards, the Nexys DDR4, Mojo v3, and Papilio One 250K. These boards were used because they contained the Artix-7, Spartan-6, and Spartan-3 FPGAs, respectively. Vivado Design Suite, Yosys, and Synplify were used to perform synthesis and implementation. Microblaze, Picoblaze, and Digital Clock Manager were used to achieve timing closure.

4.3 Development Boards

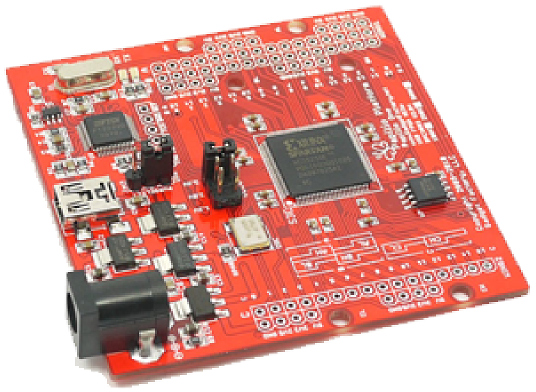
4.3.1 Mojo v3

The Mojo v3 is a FPGA development board that uses a Xilinx Spartan 6 FPGA and a ATmega32U4 microcontroller, like what is found on the Arduino Uno. The microcontroller is used to for configuring the FPGA, USB communications, and reading the analog pins. The Mojo's on board flash memory allows of the storage of an FPGA configuration file.



4.3.2 Papilio One 250K

The Papilio One 250K is a open source FPGA development board that uses a Xilinx Spartan 3E FPGA and an AVR8 softcore processor. The Papilio allows for you to read Arduino code in a custom version of the Arduino IDE. The device is not directly supported in Xilinx ISE, but you can use a script file to load bitstreams generated by Xilinx ISE.

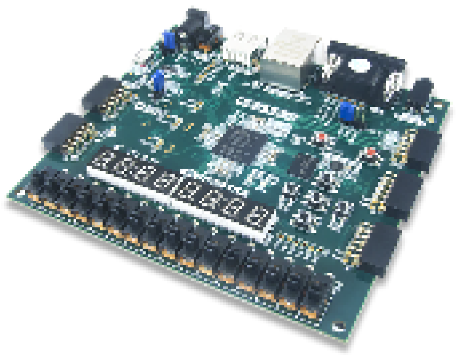


4.3.3 PulseRain M10

The PulseRain M10 is a FPGA development board that uses a Intel M10 FPGA with an embedded soft-core MCU operating at 96MHz. The PulseRain offers a Arduino compatible software interface. The Intel M10 FPGA offers 378 Kb of Block RAM and 178 KB of flash memory. Instead of using a software bootloader for device programming the PulseRain saves RAM space by using a on chip debugger.

4.3.4 Nexys DDR4

The Nexys DDR4 development board features a Xilinx Artix-7 FPGA and is a ready-to-use digital circuit development platform designed to bring additional industry applications into a classroom environment. The board offers 128 MiB DDR2 and 15,850 logic slices containing four 6-input LUTs and 8 flip-flops each.



4.4 Theory

For this project, we wanted to emphasize the importance of software verification when using cryptographic APIs to protect data. We also wanted to make a case for using FPGAs to run the encryption instead of handwritten software in C. Software verification in cryptography is important because if the software is not proven to be correct with regards to the mathematical specification being implemented then it may not be secure. Cryptol and SAW are tools that are used to prove correctness and equivalence of various cryptography implementations. They were also used to generate a formally verified synthesizable version of AES-128 in Verilog. To prove that the FPGAs were able to outperform the software implementations we used overall throughput in Gbps to compare them.

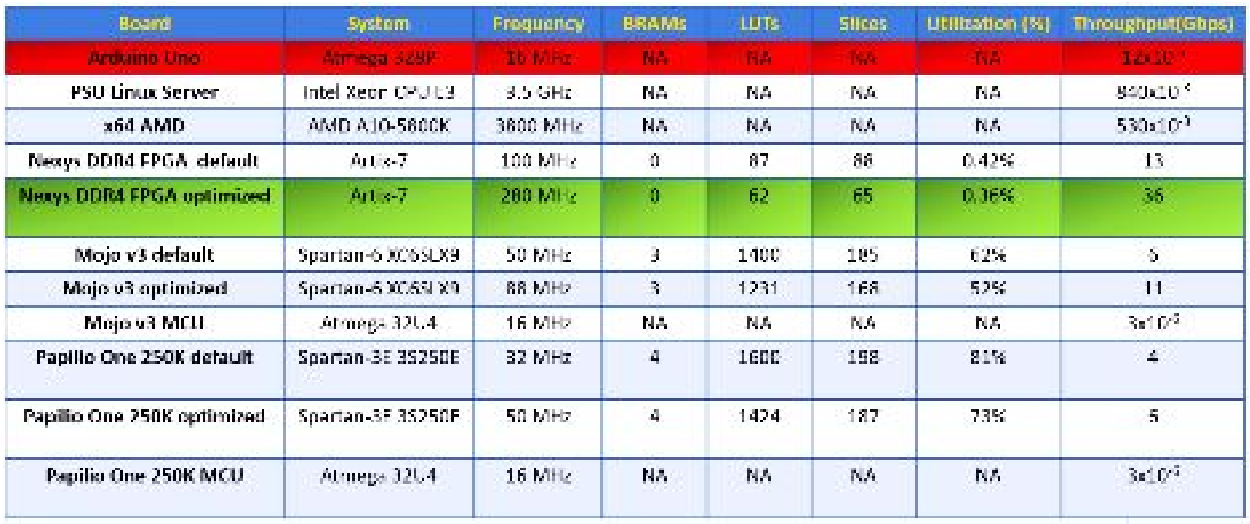
5. Results

5.1 Testing C file

OpenSSL's version of AES-128 encryption was tested on the Intel Xeon CPU E3, AMD A10-5800K, ATmega32U4, and the ATmega328P. All results for throughput were represented in the form of Gigabits per second. The Intel Xeon CPU E3 exhibited the best throughput at 0.84 Gbps. The ATmega32U4 had the worst throughput at 0.000003 Gbps.

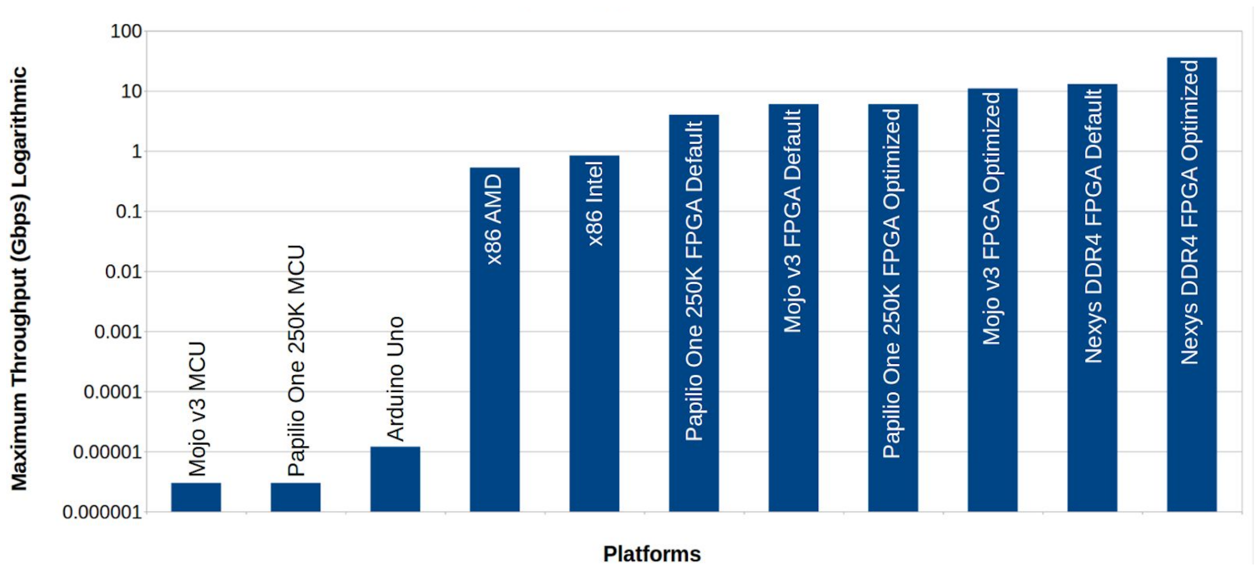
5.2 Testing on FPGA

When loading the bitstream onto different FPGAs and running the benchmark file on microprocessors and platforms, these are the results we obtained. The maximum throughput is measured in Gbps and tells us how it long it takes to perform a single encryption. This number stays pretty constant and does not vary depending on the number of encryptions that have to be performed.

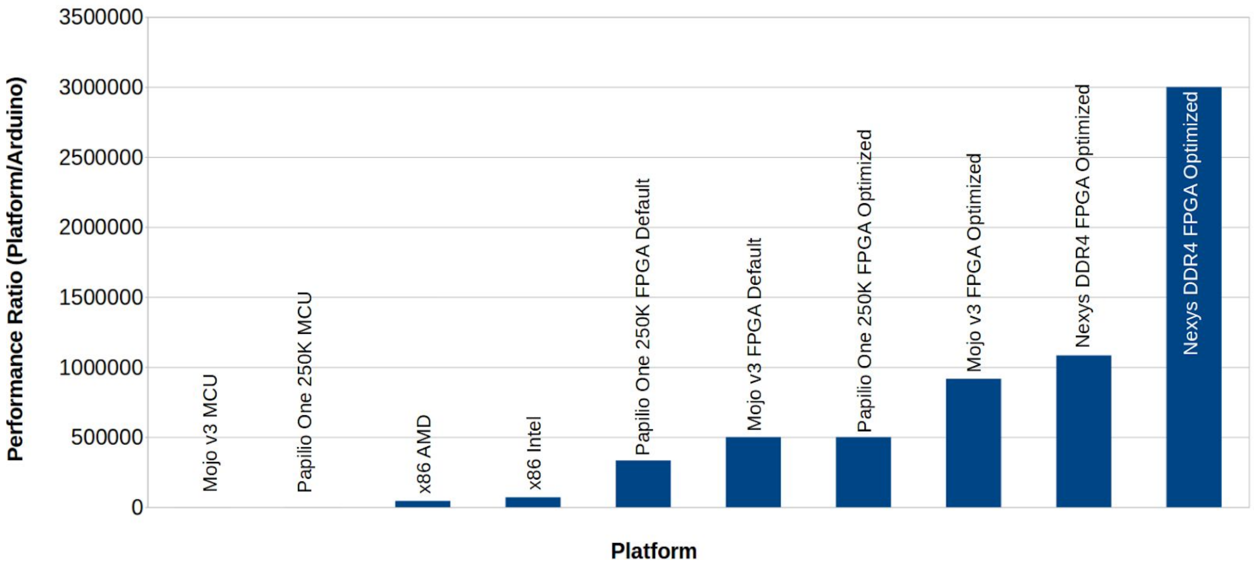


Board	System	Frequency	BRAMs	LUTs	Slices	Utilization (%)	Throughput(Gbps)
Arduino Uno	Atmega 328P	16 MHz	NA	NA	NA	NA	12×10^{-6}
PSU Linux Server	Intel Xeon CPU E3	3.5 GHz	NA	NA	NA	NA	840×10^{-3}
x64 AMD	AMD A10-5800K	3800 MHz	NA	NA	NA	NA	530×10^{-3}
Nexys DDR4 FPGA default	Artix-7	100 MHz	0	87	88	0.42%	13
Nexys DDR4 FPGA optimized	Artix-7	280 MHz	0	62	65	0.36%	36
Mojo v3 default	Spartan-6 XC6SLX9	50 MHz	3	1400	185	62%	5
Mojo v3 optimized	Spartan-6 XC6SLX9	88 MHz	3	1231	168	52%	11
Mojo v3 MCU	Atmega 32U4	16 MHz	NA	NA	NA	NA	3×10^{-6}
Papilio One 250K default	Spartan-3E 3S250E	32 MHz	4	1600	198	81%	4
Papilio One 250K optimized	Spartan-3E 3S250E	50 MHz	4	1424	187	73%	5
Papilio One 250K MCU	Atmega 32U4	16 MHz	NA	NA	NA	NA	3×10^{-6}

Table 1: Performance Comparison Across Different Platforms



Graph 1: Performance Comparison Across Different Platforms



Graph 2: Performance Ratio of Different Platforms with Respect to Arduino Uno MCU

This analysis is done to reference how fast other platforms perform in comparison with the Arduino Uno which has the most basic microcontroller (Atmel ATmega328P).
We notice that the performance of the Nexys DDR4 Development board is the best. It has a maximum throughput of 36 Gbps and has the Artix 7 FPGA. An important observation to note will be the resource utilization in the Artix-7 chip. 0.36% of the resources were consumed to perform the encryptions. This number will decrease once BRAMs are used.

6. Conclusion

On comparing the performance on different platforms, we can conclude that an FPGA can encrypt data much faster than conventional software and microcontrollers. The performance of the FPGAs can be made to match a GPU if the FPGA is advanced. We saw how an FPGA is 1,000,000x- 3,000,000x faster than the ATmel microcontroller.

Using an FPGA to encrypt data is certainly a good option since the system is isolated thus providing consistent outputs all the time. Tools like Microblaze and the Digital Clock Manager can be used to calculate the throughput, display the cipher text and also run a check to see if the outputs are as expected. Debugging using an FPGA is much less complex than doing the same in a GPU. The same can be said about benchmarking on a GPU which is more cost and labor intensive due to the complexity of simply loading the input stream of plain text.

From this project, we have concluded that the Artix-7 is capable of handling heavy computation and a large number of encryptions and can thus be used in the custom FPGA shield that would be compatible with the Arduino Uno. We also compared our values to throughput values found in case studies and observed that the performance of the AES implementation given to us was 30% - 40% better than them. This proves that the implementation does match the high- performance high-assurance criteria.

7. Project Schedule

December 2017:

- Project specifications given by Galios
- Read and understand what crypto algorithms are
- Research how to create an FPGA shield compatible with an Arduino

January 2018:

- Make a list of questions for sponsors to get a better idea of the project
- First meeting with Galois sponsors
- Create Project Design Specification
    - Write project background and motivation
    - Define overall project objective
    - Develop marketing requirements
    - Develop engineering requirements
    - Linking marketing and engineering requirements
    - Finalize the project design specification
- Research the Arduino platform
- Research different FPGAs
    - Research Xilinx FPGAs
    - Research Microsemiconductor FPGAs
    - Research Lattice FPGAs
    - Research Intel FPGAs

    - Compare and choose FPGA
- Research development boards for testing
- Preform initial software research
- AES algorithm research

February 2018:

- Create BOM for embedded boards to buy for FPGA evaluation purposes
    - Research documentation on the Mojo v3 development board
    - Research documentation on the Papilio One 250K development board
- Research AES128, SHA-2, and RNG algorithms
- Create Project Proposal
    - Create initial project proposal template
    - Create block diagram of the hardware
    - Create software flow diagram
    - Create Bill of Materials
    - Create testing and debugging plan
    - Create project scheduling
- Cryptol and SAW research
    - Cryptol tool research and reading documentation
    - Test basic crytol functions
    - SAW tool research and reading documentation
    - Test basic SAW functions
- Research OpenSSL benchmarking
- Add research information to the projects Wiki
    - Information about development boards under test
    - Add documentation about the Cryptol and SAW tools
    - Add information about AES-128 algorithm
    - Add information about SHA algorithm

March 2018:

- Review proposal, make necessary changes
- Start experimenting with Mojo v3 and Papilio One 250k board
    - Run basic introduction code on the Mojo v3
    - Run basic introduction code on the Papilio One 250K
- Understand the usage of formal methods and testing with Cryptol and SAW with the help of sponsors in the demo session
- Collect PulseRain board and look for tools to load bitstream on Intel M10 FPGA
- Run AES-128 algorithm on OpenSSL
- Benchmark the AES-128 C file
    - Create initial benchmarks for AES-128 in C
    - Test the benchmark for AES-128

April 2018:

- Continue working with Mojo, Papilio and PulseRain board and try to debug errors and fix preliminary configuration issues
- Test the aes.c and benchmark file on various platforms
    - Load aes.c and run the benchmark on the Arduino Uno Platform
    - Load aes.c and run the benchmark on the Mojo v3
- Create graphs for the results of the benchmarking on the MCUs
- Papilio configuration and setup problems continue
- Simulate and synthesize gulphaac v files : debug errors
- After meeting with Galois we are only focusing on the AES-128 algorithm

May 2018:

- Test the aes.c and benchmark file on various platforms
    - Load aes.c and run the benchmark on the Intel x86 machine
    - Load aes.c and run the benchmark on the AMD x86 machine
    - Run benchmark for 10,000 trails on an x86 machine to confirm accuracy
    - Run benchmark for a range of trails to confirm standard deviation of tests
- Simulation on Vivado works
- Synthesis fails because of floorplanning errors, lack of constraints file and because of 128 bit inputs and outputs used as channels and interfaces
- Convert channels and interfaces to inputs and outputs to make it synthesizable. Since floorplanning error doesn't resolve, create a new top module.
- Create a wrapper function or a UART top module that can synthesize and will process 8 bits at a time with a transmitter and receiver. Display output using RealTerm serial monitor.
- Project synthesizes and generates the bitstream. Load bitstream onto the board and look at the timing reports and output.
- Calculate throughput of the Nexys DDR4 board since Vivado HLx works for 7-series FPGAs.

June 2018:

- Mojo v3 uses Spartan 6 so Vivado ISE is used to generate the bitstream. Since the board isn't recognized and bitstream generated has errors, use Yosys for bitstream generation and switch back to Vivado to upload the bitstream onto the board.
- Use Microblaze and Digital Clock Manager to perform the frequency sweep and find the default and maximum frequency at which there are no timing violations.
- Use dedicated resources on FPGA differently to optimize the performance: BRAM, Distributed RAM etc.
- Make a table with the default throughput and the optimized value with the higher frequency.
- Create graphs for the FPGA maximum throughputs and compare with the performance of other platforms.
- Create final presentation for Galois and Dr. Teuscher
    - Create slide for practice run with Dr. Teuscher
    - Practice the presentation for practice run with Dr. Teuscher
    - Modify the slide submit them to Dan and Joe for review

    - Make final modifications
    - Practice presentation
- Create final report (Whole team)
    - Create the summary
    - Create the introduction and motivation
    - Talk about the problem and the proposed solution
    - Create final project schedule to summarize accomplishments
    - Create conclusion to summarize results
    - Talk about testing methods
    - Talk about the challenges faced and future work to be done
- Create capstone poster (Whole team)
    - Create initial poster to be review by Dr. Teuscher
    - Modify poster and present to Dan and Joe
    - Make final modifications to poster and send to print

Dustin	Gomathy	Meiqi	Ryan
● Research various FPGAs that can be used with the Arduino ● Researched AES-128 algorithm ● Tried to get the Papilio One 250K working ● Benchmarked aes.c code on Intel and AMD x64 machines ● Fixed issue with aes.c needing too much memory on the Arduino ● Benchmarked aes.c code on Arduino Uno ● Test Cryptol and SAW ● Added research information on AES-128 to GitHub Wiki ● Graphs for aes.c benchmarking on different platforms. ● Graphs for FPGA throughputs and comparison with other platforms. ● Graphs of the throughput with respect to the Arduino MCU ● Performed tests with 10,000 trials using aes.c to confirm the accuracy when performing 10000000 encryptions	● Research various FPGAs that can be used with the Arduino. ● Researched AES-128 algorithm. ● Got the Mojo v3 MCU and FPGA working. ● Tested Cryptol and SAW. ● Benchmarked aes.c on Arduino Mega MCU. ● Cryptol and SAW testing on aes.cry ● Simulated the verilog files and verified that the output was correct in Vivado. ● Synthesized the code by creating top module with UART Tx and Rx modules. ● Fixed floorplanning errors and timing violation errors. ● Implemented and generated bitstream and loaded onto Nexys DDR4 board. ● Used Microblaze and Digital Clock Manager for performing frequency sweeps and fixed timing violations. ● Calculated maximum throughput. ● Generated bitstream for Mojo v3 board and calculated throughput. ● Created tables comparing throughputs of various platforms and ones in case studies.	● Research various FPGAs that can be used with the Arduino ● Researched AES-128 algorithm ● Researched the SHA-1 and SHA-2 algorithms ● Tested the speed of AES-128, SHA-1 and SHA-2 algorithms on Openssl ● Tested Crypto and SAW ● Tried to get the Papilio One 250K working ● Tested the Benchmark c file ● Cryptol and SAW testing on aes.cry	● Research various FPGAs that can be used with the Arduino. ● Researched AES-128 algorithm. ● Got the Mojo v3 MCU and FPGA working. ● Tested Cryptol and SAW. ● Benchmarked aes.c on Arduino Uno MCU and Mojo v3 MCU. ● Cryptol testing on aes.cry. ● Wrote c wrapper function to benchmark aes.c on different platforms. ● Wrote c wrapper functions to generate standard deviation for graphs. ● Wrote c wrapper functions to test aes.kat file with known good outputs. ● Graphs for aes.c benchmarking on different platforms with standard deviation. ● Simulated the verilog files and verified that the output was correct in QuestaSim. ● Synthesized the code by creating top module with UART Tx module. ● Fixed floorplanning errors and timing violation errors. ● Implemented and generated bitstream and loaded on Nexys DDR4 board. ● Calculated maximum throughput from timing reports.

8. Testing Procedures

8.1 Testing Software Benchmark Files

For testing software on the CPUs and MCUs we used aes.c and associated files from the OpenSSL software library. On the CPUs we were able to use a specific benchmark file generated by OpenSSL that gave us results in the form of KBytes encrypted per second. On the MCUs we had to port over a lot of the C and header files from the library and mimic what was done in the benchmark files. This allowed us to obtain the software throughput for all the systems that we tested. A file containing 100 different keys, plain text data values, and known good encrypted outputs of block size 16 was used to confirm that the encryptions were working as expected.

8.2 Testing on Vivado HLx

With the UART top module that used two FIFOs, one for transmitting (Tx) and the other for receiving (Rx), we were able to display data on a serial monitor by configuring it at a specific baud rate.
Using 115,200 as the baud rate, 8 bits of the input were serially taken at a time 16 times and then was internally converted to a parallel stream that would be encrypted using aes_functions.v. After being encrypted, the output will be displayed 8 bits at a time. We checked the output against the expected value and also compared the timing reports with the reports from other open source tools like Yosys and Synplify.

8.3 Testing on Vivado Microblaze

On the Nexys DDR4, we used Microblaze and the Digital Clock Manager to estimate the maximum throughput. Since a frequency sweep had to be performed to find the default and the maximum frequency at which encryption can be done without timing violations, we used this tool. Microblaze was used to do the following:

- Display the last eight bits of the output or the cipher text on the Microblaze console
- Print a dot every time an encryption was performed without error
- Change hardware allocation settings to see how performance differed when Block RAMs or Distributed RAMs were used.
- Observe ciphertext generated after crossing the maximum threshold frequency and use this as a basis to decide the maximum frequency at which the system can work fault free.
- Generate automatic breakpoints that will be inserted by the tool to indicate errors including timing violations and floorplanning errors.

9. Overall Project Strengths & Weaknesses

9.1 Project Strengths

- FPGA can encrypt data faster, more accurately and securely than a MCU.
- The performance of an FPGA can be optimized using tools that will help in dedicated resource utilization, finding and optimizing the critical path and use techniques like pipelining and loop unrolling to widen the operating frequency band. This will ensure that there are no timing violations.
- FPGA can be reconfigured, debugged and can process input and outputs without threading techniques.
- Since FPGAs work in an isolated system, no external factor affects the performance.

9.2 Project Weaknesses

- Only one algorithm, AES128 was tested because of the time taken to setup all the FPGA boards. We did not have the synthesizable .v files for the other algorithms.
- A variety of tools had to be used for each FPGA dev board as different dev boards used different FPGAs
- We didn’t have time to compile the code on platforms like PCIe, GPU, CoreGen etc to compare the values so used case studies as a reference.
- We used RealTerm serial monitor instead of an Arduino API as we used the Nexys DDR4 board primarily. The Nexys DDR4 board does not have a MCU.

9.3 Challenges Faced

9.3.1 Cryptol and SAW:

We used Cryptol to check, test and prove that the AES implementation. A mathematical function is created that will be used to test all the properties and SAW will be used for equivalence testing. Both these tools showed that the AES implementation was 100% formally assured. We used the tools for synthesis, validation and verification of the algorithm. These tools automatically generate high-performance high-assurance verilog code that is secure and correct.

9.3.2 OpenSSL Benchmarking:

The C files from OpenSSL were not well documented and did not really follow any strict coding standards. This made finding the files needed to run software benchmarking on systems we used in this project difficult, especially on some of the embedded MCU’s. Also to get a more accurate measurement of the software on CPUs, we ran these tests hundreds of times in order to find a result that had low variance in standard deviation.

9.3.3 Vivado Synthesis:

Challenges	Solutions
Channels and interfaces in the top module were not synthesizable.	Using input, output, wire, reg, and other options.
128 bit inputs and outputs caused place and route errors due to lack of constraints file.	● All boards don't necessarily have 384 I/O pins. ● We used FIFOs to read smaller streams (8 bits at a time) of data, process and display the output.
Not all the boards have a LCD screen or monitor on the board so TeraTerm/serial monitor has to be configured to display results.	● Used UART for transmitting and receiving data so that result can be displayed on the serial monitor. ● Used Picoblaze/Microblaze in debug mode to check for timing violations.
Loading the bitstream onto different FPGA boards (Xilinx chips 6 series, 7 series and Intel M10).	● For Xilinx series 6: Xilinx ISE and Yosys synthesizer. ● For Xilinx 7 series: Xilinx HLx ● For Intel M10: Quartus
Finding the optimized frequency that can produce maximum throughput	Used Picoblaze/Microblaze in debug mode and tested multiple frequencies from default frequency (clock) to maximum threshold frequency by doing a frequency sweep.

10. Future Work

Since we have managed to evaluate the best FPGA in terms of performance and resources used, we will either use the Artix-7 chip or optimize the performance of any other FPGA to match the required throughput. With Microblaze and an extended Vivado license, a 6-series FPGA can be made to perform with a good maximum throughput. We can use techniques like multithreading, loop unrolling and pipelining to optimize the top module.

RealTerm can be replaced with a custom made API that would enable communication between the FPGA and the Arduino. This would reduce the Tx and Rx time. We have currently verified the AES-128 implementation. We will do the same for SHA-2 and RNG (DRBG).

The final goal would be to create an FPGA shield for the Arduino Uno that would consume low power, generate high throughput and would be formally verified using formal methods. This shield can be easily deployed in companies and will be a hardware crypto platform to verify crypto algorithms.

11. Bibliography

Burrus, D. (2014). *The Internet of Things Is Far Bigger Than Anyone Realizes*. [online] WIRED. Available at: https://www.wired.com/insights/2014/11/the-internet-of-things-bigger/

Csrc.nist.gov. (2001). *Advanced Encryption Standard (AES)*. [online] Available at: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

Gisselquist, D. (n.d.). *wbuart32*. [online] GitHub. Available at: https://github.com/ZipCPU/wbuart32

El-Abd, M. (2017). A Review of Embedded Systems Education in the Arduino Age: Lessons Learned and Future Directions. *International Journal of Engineering Pedagogy*, [online] 7(2). Available at: http://online-journals.org/index.php/i-jep/article/view/6845/4454

Freedberg Jr., S. (2016). *Faster Than Thought: DARPA, Artificial Intelligence, & The Third Offset Strategy*. [online] Breaking Defense. Available at: https://breakingdefense.com/2016/02/faster-than-thought-darpa-artificial-intelligence-the-third-offset-strategy/

Fpga4fun.com. (n.d.). *How the RS-232 serial interface works*. [online] Available at: https://www.fpga4fun.com/SerialInterface1.html

Kessler, G. (2018). *An Overview of Cryptography*. [online] Gary Kessler Associates. Available at: https://www.garykessler.net/library/crypto.html#intro

Krawczyk, H., Bellare, M. and Canetti, R. (1997). *RFC 2104 - HMAC: Keyed-Hashing for Message Authentication*. [online] Tools.ietf.org. Available at: https://tools.ietf.org/html/rfc2104

Landman, N., Ross, E. and Williams, C. (n.d.). *Secure Hashing Algorithms*. [online] Brilliant.org. Available at: https://brilliant.org/wiki/secure-hashing-algorithms/

Selent, D. (2010). *Advanced Encryption Standard*. [online] Rivier University. Available at: https://www2.rivier.edu/journal/ROAJ-Fall-2010/J455-Selent-AES.pdf