Hardware Security Module

ECE 445 Design Document
Team 63
Frankie Papa, Calvin Fisher, Nick Schiesl
TA: Evan Widlowski
2/23/2020

1 Introduction

1.1 Problem and Solution

Hardware Security Modules (HSMs) are devices that have a protected keystore coupled with crypto hardware that provides features such as encryption/decryption or random number generation [1]. With the amount of personal data that companies have to keep secure (such as credit card info, passwords, etc.) it is necessary for these companies to assure their customers that they keep it highly secure. HSMs provide a much higher level of security than the use of software encryption and for this reason are popular as a defense for this info. These devices allow a customer to encrypt their confidential data and store the key to the data in hardware. Now, attackers would have to gain access to the HSM if they wanted the keys that are used to decrypt the data.

The problem is that these products can be very expensive and are not always simple to use for some purposes. For example, one of our members faced an issue while working with the Trusted Platform Module (TPM), a cheap and standard HSM in many personal computers, where he was not able to persist symmetric keys (a feature desired in the project he was working on). These devices can also be very costly ranging from low cost options at around $110.00 to high cost units at around $32,000 [2]. The problem that we are looking to solve is that of finding a low-cost, high-security option. A typical low cost HSM has limited key storage space and limited support, or no support, to symmetric algorithms [2]. Our solution to this is to fill this gap by producing a cheaper HSM that has the ability to store a large amount of keys, provides symmetric encryption algorithms, and has a random number generator with high entropy.

In order to provide a more secure product at this lower price we will make sure that it fulfills the main requirements for FIPS 140-2 Level 3 with room for additions later to fully fulfill each requirement. In NIST's documentation on FIPS 140-2 Level 3 the main features that we are implementing would be at least one approved algorithm/security function, the use of tamper-evident coatings or seals, and systems in place which make it more difficult to gain access to the modules inside as well as the zeroization of keys in the event that the cryptographic module is opened [3]. In this case, our approved algorithm is the Advanced Encryption Standard (AES). We have found that many of the higher end HSMs that are FIPS 140-2 Level 3 or 4 certified and have higher performance are priced near $1000.00 or more and we aim to provide a device capable of some of their features at a fraction of the price [1].

1.2 Visual Aid

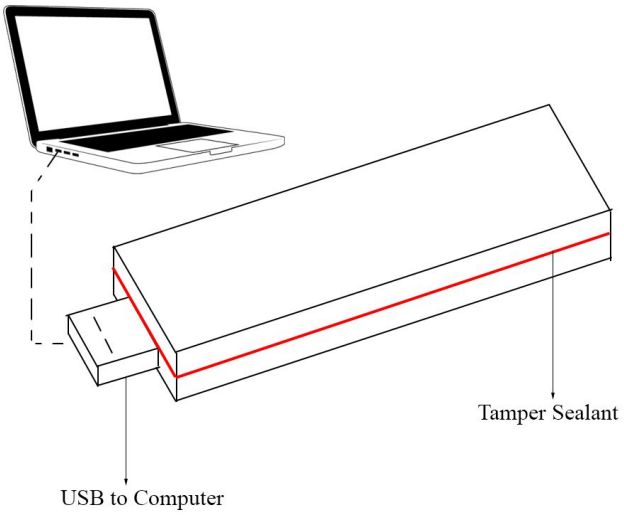

Figure 1. Visual Aid

1.3 High-Level Requirements

1. The device must be able to store at least 10,000 AES-128 keys in nonvolatile memory
2. The device must be able to encrypt and decrypt files at least as fast as 200 Mb/s
3. The random number generator on the device can generate statistically random AES-128 keys
4. Tamper evidence module must be able to zeroize the keystore in the event of physical tampering

2 Design

2.1 Block Diagram

In order for our design to be successful our device is going to need a power supply, a random number generator, a tamper evidence module, and a control module (which doubles as the encryption module). With the parts that we have chosen our power supply will come from the USB connector which has a voltage rating of 30V. The voltage regulator will then supply 5V to each component and 18V to the Random Number Generator (RNG) circuit. The flash memory has 2GB storage which allows us to store a lot of 128-bit keys. The tamper evidence buttons and the conductive wire mesh sensor will alert the microcontroller of tampering and wipe the keys from the keystore. The RNG circuit will generate AES-128 bit keys on command from the microcontroller which will store them in the flash memory. Lastly, inside of the control module the microcontroller will route all data to each piece. Doubling as the encryption module, the microcontroller will also use AES encryption to encrypt or decrypt data sent in through the USB connector using a designated key from the keystore.

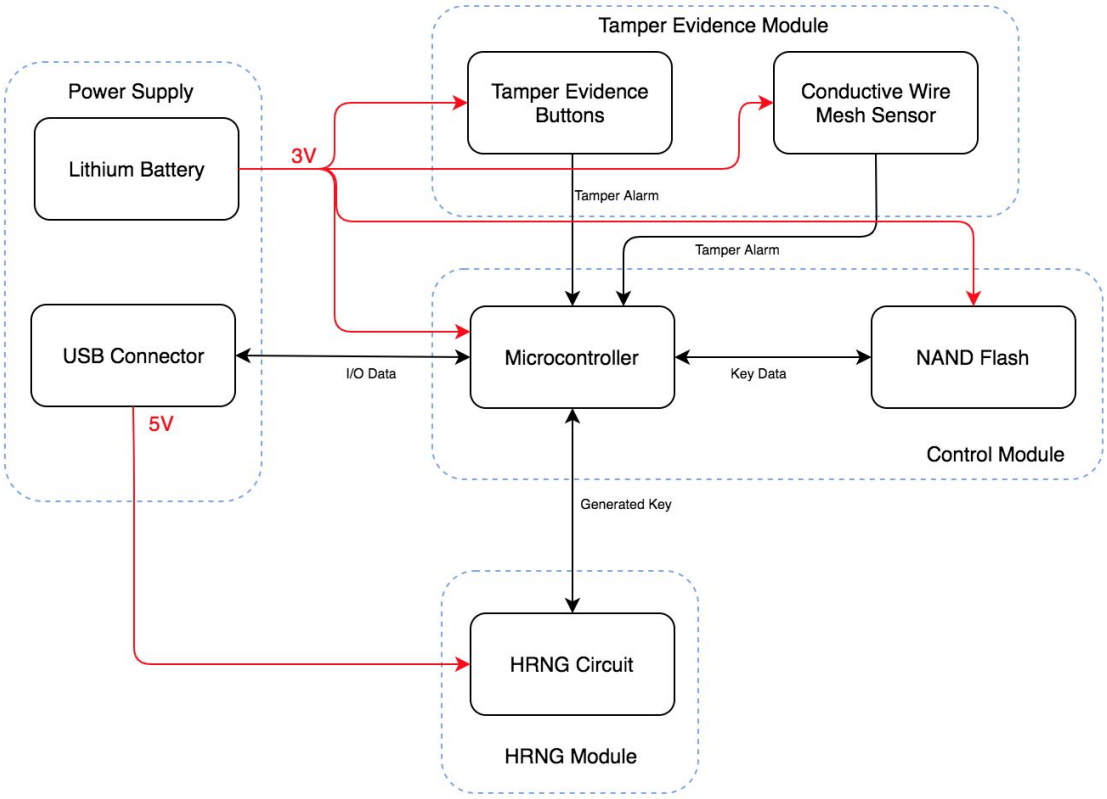

Figure 2. Block Diagram

2.2 Power Supply

The main source of power for our device will be through a USB power supply. The USB Connector will have an output of 5 volts. This supply voltage will allow us to power all of our devices and supply enough current for all of the components. The 3 volt lithium battery will supply power to the microcontroller, NAND flash memory, and the tamper evidence module circuits. This battery is included for the sole purpose of operating the tamper evidence module when the device is not connected by USB to ensure the device's safety.

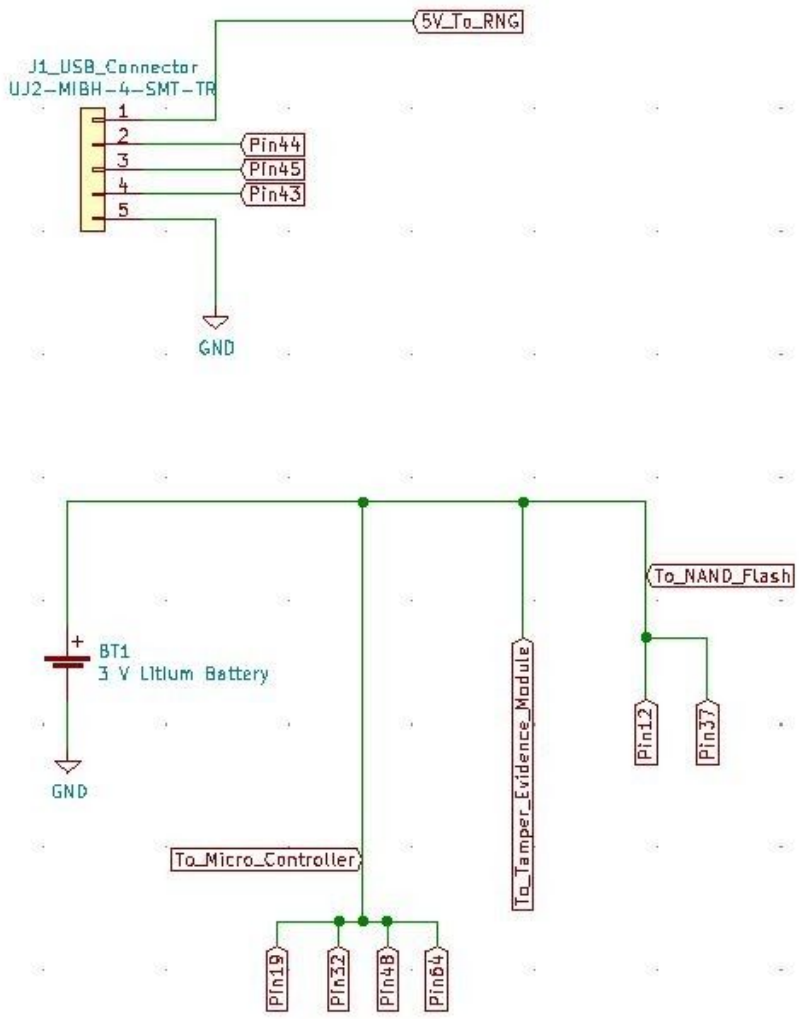

Figure 3. Power Supply Schematics

Requirements	Verifications
1. USB Supplies 5 volts 2. 3V battery powers the tamper evidence module, micro controller, and NAND flash	1. Measure the voltage output of the USB connector with a multimeter to make sure its output is 5 volts. 2. Measure the positive terminal of the battery with a multimeter to make sure it is supplying 3V. Next, probe the power pins of the microcontroller and the NAND flash to test if they are receiving 3 volts and if they are powered on. Then test the tamper evidence module using a multimeter to make sure it has the correct voltage inputs and outputs.

2.3 Control Module

The control module in our HSM doubles as the encryption module as it will be taking in all the data from the USB connector and performing the designated operations (encrypt, decrypt, store key, or generate random key) and executing the code to encrypt or decrypt data. This is all headed by the STM32F730R8 microcontroller that we chose which will be the main base of operations. Each individual module will connect here and we will have software programmed onto it to deal with the incoming data and do whatever the data demands. Figure 4 below shows the flow of operation from the perspective of the software on the microcontroller. Additionally, the control module, in the case of a signal for encryption/decryption, will use Cipher Block Chaining (CBC) AES which is a more complex version of AES which uses the previous block of ciphertext to change the next block of plaintext prior to the AES algorithm. Figures 5 and 6 show diagrams of the CBC AES algorithm. In the case of decryption we use inverses of the functions shown in figure 5 and the algorithm is done in reverse order in order to undo our encryption; we are capable of doing this because it is a symmetric encryption algorithm. We decided that the encryption and decryption algorithms must be able to run at speeds of at least 200 Mb/s. This number is based on another implementation of AES on a microcontroller and gives us room for improvement towards their hardware solution which produced speeds of 372Mb/s [4].

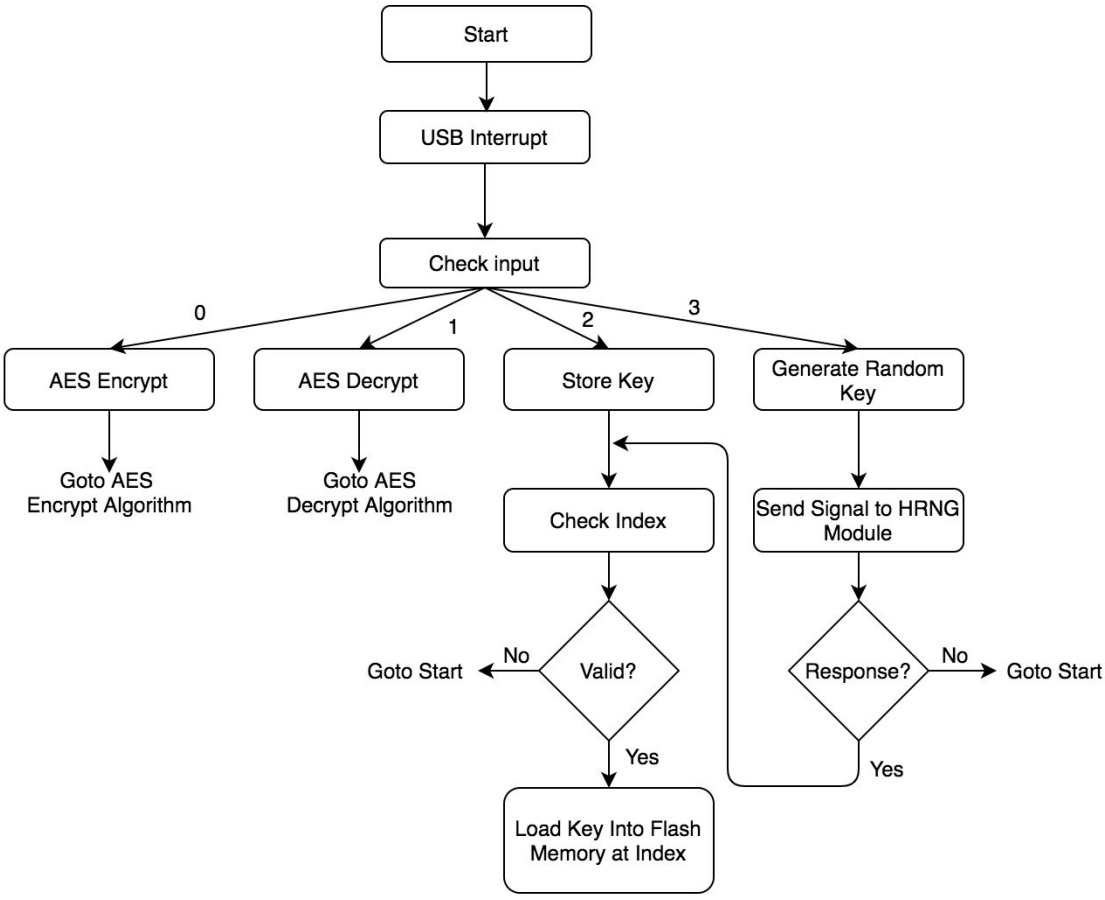

Figure 4. Microcontroller Software Flow Chart

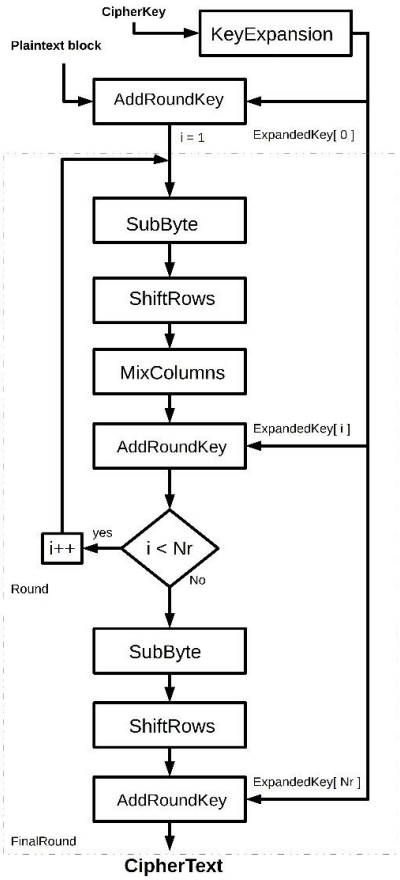

Figure 5. AES Algorithm [5]

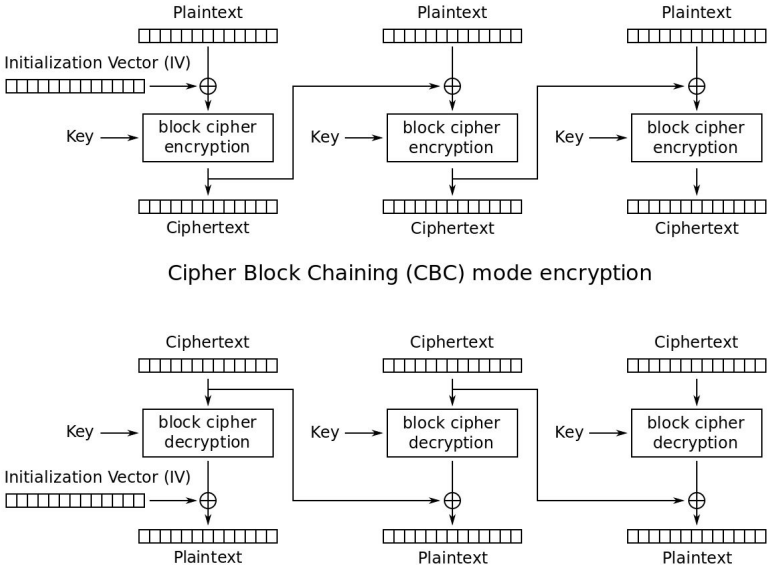

Figure 6. CBC mode Encryption [6]

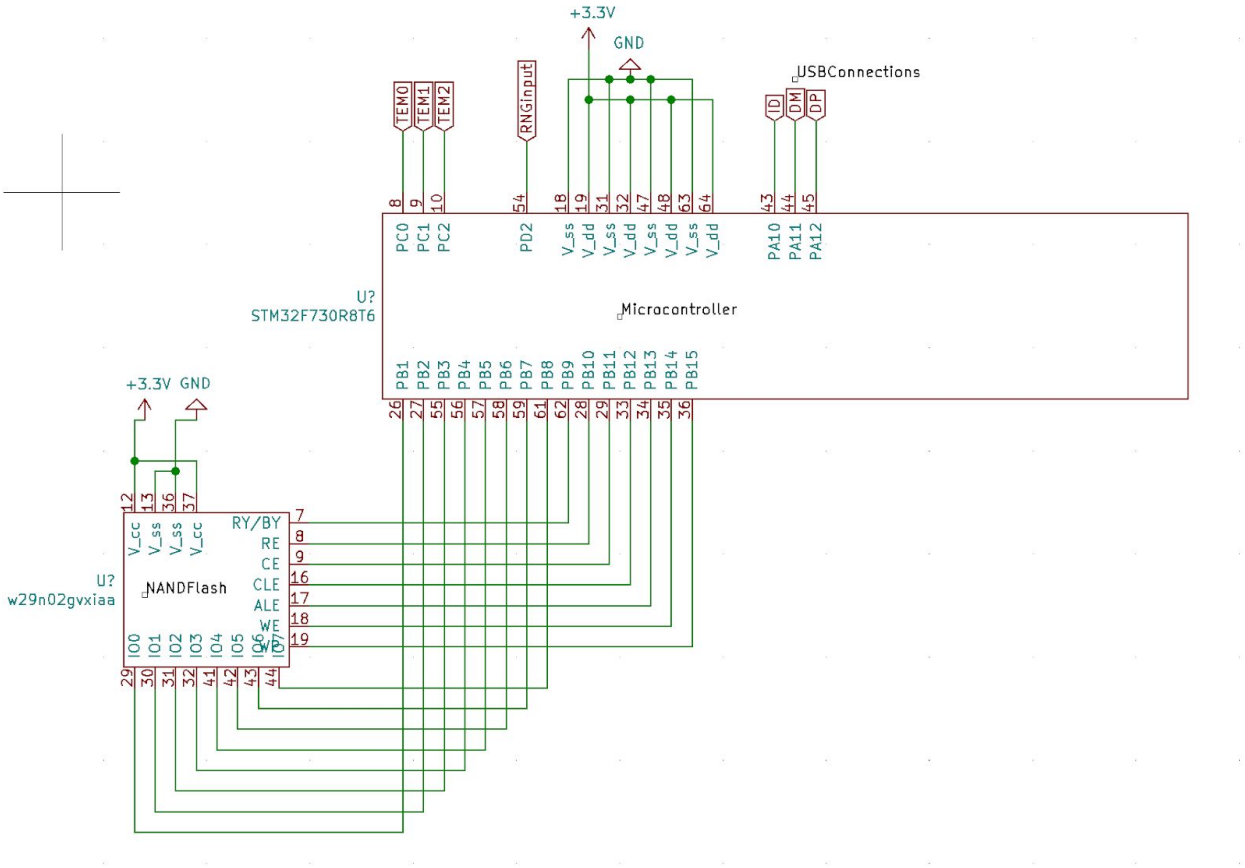

Figure 7. Control Module Schematic

Requirements	Verifications
1. Must be able to store at least 10,000 AES-128 keys in nonvolatile memory 2. Must be able to do the encryption/decryption operations at a speed of at least 200 Mb/s 3. Must be able to input and output data to and from the USB connector	1. Create a C script which will first load 10,000 AES-128 keys onto the device through the USB and then compare the keys loaded to those on the device to confirm it properly stored all 10,000 keys. 2. Write a C script which will encrypt a 1MB file and times the amount of time it takes from data transmission to the device until the ciphertext is transmitted back. Then calculate the rate in Mb/s. 3. Send data to microcontroller from a computer where the two are connected by a USB and load data into flash memory and then load it back to the computer and compare the two

2.4 RNG Module

The HRNG or Hardware Random Number Generator is essential for this encryption device. We are hypothetically creating a hardware security device for the market, so security is top priority. The HRNG will be a big step up from the PRNG or pseudo random number generator that most programs use. In a PRNG a “seed” is passed through a black box algorithm which spits out a “random” number. Obtaining the seed used makes the random number obsolete. In our case, obtaining a seed means obtaining the AES-128 bit key for an experienced hacker. Our HRNG will not use a seed and will instead use digitized transistor noise to generate random bits.

In our HRNG a reverse biased or negative voltage applied transistor will generate noise. This noise will then be amplified by 2 more resistors. The noise is then “digitized” by a Schmitt Trigger Inverter. We now have a readable output of statistically random bits. Statistically random means that our bit stream does not produce any recognizable patterns or regularities, but is still not truly random.

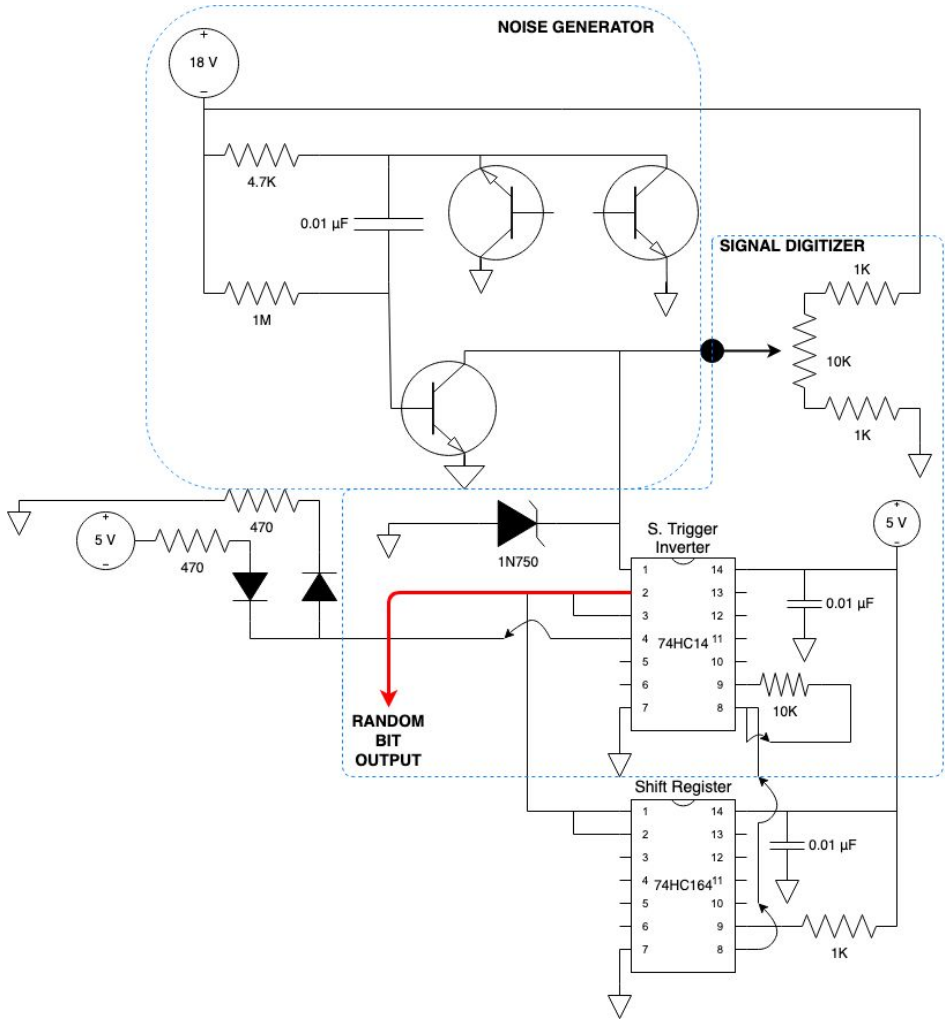

Figure 8. HRNG Schematic

Requirements	Verifications
1. Produces a statistically random bit output	1. We will use an arduino to probe our random number generator for 128 bits and feed a file with these bits into the Dieharder Test Suite which uses the diehard tests the statistical randomness of a random number generator 2. We will purchase extra LEDs that can be placed in series with resistors on the Shift Register (74HC164) to see real time random bits being generated and fed at clock speed, these random "states" can be tested for statistical randomness

2.5 Tamper Evidence Module

The tamper evidence module is what helps us to comply with most of the standards of FIPS 140-2 levels 2 and 3. Our module will zeroize the keybase in two different scenarios: when the casing has been removed or when an attacker is drilling into the case. In the scenario where the casing is being removed one or more of the pressure sensing buttons will be undone and the alert signal will become an open circuit. The microcontroller will be periodically polling this signal, as seen in figure 9, and when it inputs low the microcontroller will zeroize the keybase. In the case of drilling we have a circuit that detects an open circuit across a nichrome wire wrapped around the device using an AND gate. We are choosing to use nichrome 60 series wire because this is used as a resistance wire and will be useful in detecting the effects of a drill touching it. This will be wrapped around our epoxy covered circuit as a first line of defense against these types of attacks. The microcontroller will similarly poll the alert signal from the circuit and when an alert is sent it will also zeroize the keystore.

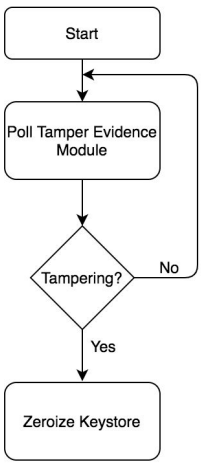

Figure 9. Tamper Evidence Software

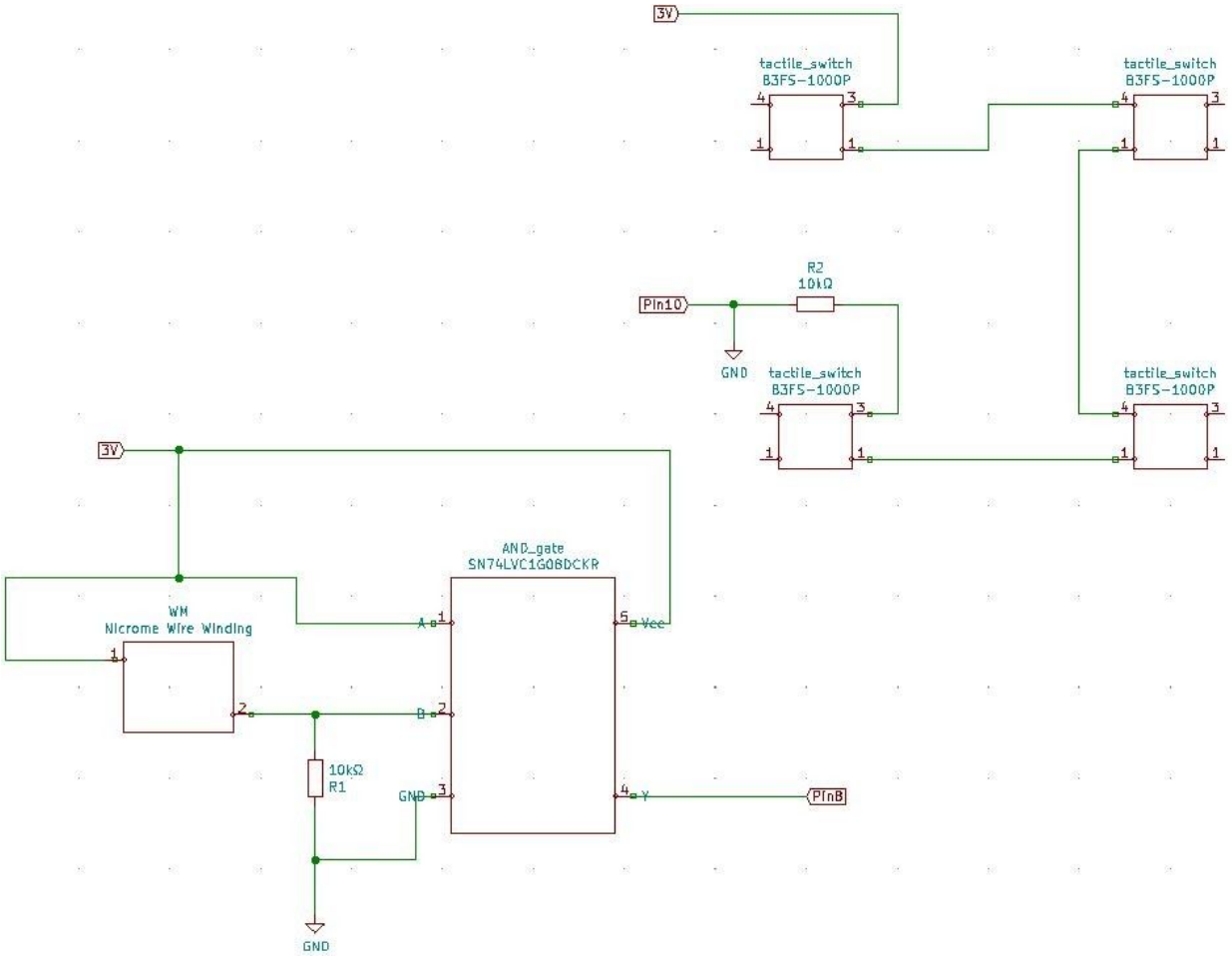

Figure 10. Tamper Evidence Module Schematics

Requirements	Verifications
1. Zeroize the keystore when the casing is removed 2. Zeroize the keystore when something penetrates the case	1. Store a key in the flash memory, remove the casing, and check the memory of the NAND flash with our microcontroller 2. Store a key in the flash memory, drill into the device, and check the memory of the NAND flash with our microcontroller

2.6 Tolerance Analysis

The main components of our device which must adhere to strict guidelines would be the HRNG module and the power supply. We suspect that our power supply must supply the desired voltages within 5% bounds. What this means is that our power supply of 18V to the HRNG module must be within 17.1V-18.9V in order to guarantee that our module gets powered correctly. The HRNG module must also output AES-128 keys that are judged to be statistically random. In order to assure this we will be using the Dieharder test suite which was designed by a professor at Duke based upon George Marsaglia's

Diehard Tests from 1995 [8]. This will take in a stream of bits or a file of bits (from our RNG) and give us a quality value to say whether or not we are creating "statistically random" bits.

3 Cost and Schedule

3.1 Cost Analysis

When determining the cost for development we assume an average salary of $80,000 per person of our 3 member team. We also assume 10 hours of work per week and a 16 week timeframe for completing the project. Using this we calculated the development cost for labor as follows.

$$\frac{\$80000}{year} \times \frac{1\ year}{52\ weeks} \times \frac{1\ week}{40\ hours} \times 3\ people \times 16\ weeks \times 10\ hours \times 2.5 = \$46,153.85 \quad (1)$$

The table below details the estimated cost for parts:

Part Name	Manufacturer	Part #	Quantity	Cost/part
Microcontroller	STMicroelectronics	STM32F730R8T6	1	$4.96
NAND Flash	Winbond	W29N029VSIAA	1	$3.82
USB Connector	CUI Devices	UJ2-MIBH-4-SMT-TR	1	$0.87
Buttons	Omron Electronics	B3FS-1000P	4	$0.65
Assorted Resistors, Capacitors, ICs, etc.	Digikey	N/A	1	~$10.00
PCB	PCBway	N/A	1	~$5.00
Plastic Housing	N/A	N/A	1	~$10.00
Lithium Battery	Panasonic	CR2032	1	$0.99
Total Cost	-	-	-	$36.29

Assuming that we will be making three prototypes for testing and demoing, the cost for parts will amount to $108.87. Adding this to the cost of labor gives us a total cost of $46,262.72.

3.2 Schedule

Week	Frankie	Calvin	Nick
3/2/2020	Research programming microcontroller and begin programming CBC AES.	Finalize parts list and verify HRNG design / statistical randomness test.	Complete tamper evidence module design
3/9/2020	Program CBC AES and key storage software	Begin PCB KiCAD design and research bit state implementation.	Order parts and begin creating the nichrome wire resistor
3/23/2020	Design PCB with parts being used	Assemble breadboard circuit / test.	Test the nichrome wire resistor and change resistor component values if necessary
3/30/2020	Assemble circuit and begin testing of software on microcontroller	Assemble circuit on PCB / verify voltage regulation.	Assemble circuit on PCB and modify the resistance in the button detection circuit for the tamper evidence module if needed
4/6/2020	Refine software and add in USB software capabilities	Test and refine PCB.	Refine all resistors and capacitors on PCB to achieve correct tolerances if necessary
4/13/2020	Assemble prototype and test each piece together	Assemble prototype / verify integration of modules.	Assemble prototype and complete final testing of design
4/20/2020	Mock demo and refine prototype	Mock demo and refine prototype	Mock demo and refine prototype
4/27/2020	Demo and begin final paper	Demo and begin final paper	Demo and begin final paper
5/4/2020	Final presentation	Final presentation	Final presentation

4 Ethics and Safety

While designing our product we have to keep in mind the ethics involved in producing a HSM with stringent security requirements. It would be highly unethical to advertise a product that is supposed to fulfill FIPS 140-2 level 3 requirements, but then fails in some aspect (perhaps some other way of tampering with our device). It would also be unethical for us to produce a faulty product; this requires us to create and administer strict tests to our device in order to assure to our customers that we have a working product.

Working in the senior design lab will require us to follow strict safety measures to prevent any injury. Using soldering irons or hot air can put us at risk of burning ourselves.To prevent this from happening will follow the correct safety precautions that were explained in our lab safety training and our soldering assignment. We will always power these devices off while not in use and use correct soldering techniques to prevent any risk of injury. Another hazard would be when we power our device with the lab power supplies. We will follow all safety guidelines to prevent risk of electrical shock or electrical shorts.

Another safety measure we will have to take into account is our use of a lithium battery. Lithium batteries can fail and cause a fire or explosion that may harm us or others during demoing and testing. This can be caused by puncture, overcharge, overheating, short circuit, internal cell failure, and manufacturing deficiencies. We will work on preventing this by operating the battery under the rated voltage, current, and temperature. We will also be careful not to damage the battery to prevent failure.

5 Citations

[1] J. Schlyter, “Hardware Security Modules,” *internetstiftelsen.se*. [Online]. Available: https://internetstiftelsen.se/docs/hsm-20090529.pdf. [Accessed: 24-Feb-2020].

[2] S. Dickinson, “HSM Buyers' Guide - Documentation Reference Material,” *OpenDNSSEC*. [Online]. Available: https://wiki.opendnssec.org/display/DOCREF/HSM Buyers' Guide. [Accessed: 24-Feb-2020].

[3] D. L. Evans, P. J. Bond, and A. L. Bement, “FIPS PUB 140-2,” 12-Mar-2002. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf. [Accessed: 24-Feb-2020].

[4] Ø. Ekelund, “Low Energy AES Hardware for Microcontroller,” *semanticscholar.org*, Jul-2009. [Online]. Available: https://pdfs.semanticscholar.org/6671/804dbe82a507cb08e8f5e392dfed668c9077.pdf. [Accessed: 03-Mar-2020].

[5] J. D. R. Arrañaga, J. A. S. Chavarin, J. J. R. Panduro, and E. C. B. Alvarez, “New S-box calculation for Rijndael-AES based on an artificial neural network,” *ReCIBE. Revista electrónica de Computación, Informática, Biomédica y Electrónica*. [Online]. Available: https://www.redalyc.org/jatsRepo/5122/512253718012/html/index.html. [Accessed: 27-Feb-2020].

[6] “Block cipher mode of operation,” *Wikipedia*, 15-Feb-2020. [Online]. Available: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation. [Accessed: 27-Feb-2020].

[7] C. Platt and A. Logue, “Really, Really Random Number Generator: Make:” *Make*, 01-May-2015. [Online]. Available: https://makezine.com/projects/really-really-random-number-generator/. [Accessed: 24-Feb-2020].

[8] R. G. Brown, D. Eddelbuettel, and D. Bauer, “Dieharder: A Random Number Test Suite,” *Robert G. Brown's General Tools Page*. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php. [Accessed: 27-Feb-2020].