Hardware Security Module

Frankie Papa, Calvin Fisher, Nick Schiesl
TA: Evan Widlowski

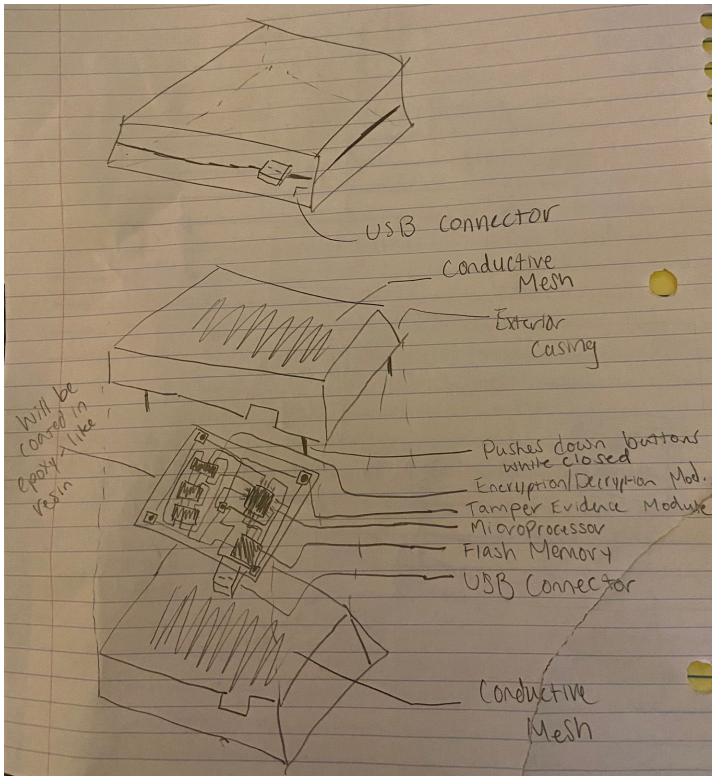
I. Introduction

A. **Problem Statement:** The Trusted Platform Module is an onboard cryptoprocessor/hardware security module on most machines and it offers a cheap way to persist asymmetric keys and do other cryptographic operations on board with a FIPS 140-2 Level 2 security. The problem is that if we want to store symmetric keys (such as AES-128 keys) we cannot store them long term (persist them) with the given windows API (CNG: Cryptography Next Generation).

B. **Proposed Solution:** My group plans to create our own small-scale and affordable hardware security module with an API that allows for persisting symmetric keys on Windows machines that satisfies the conditions for FIPS 140-2 Level 2 (or 3) security. We can then use the keys to encrypt bytes sent to the board. We plan to have a few subsystems working together which includes memory for our key storage, encryption and decryption circuits, a tamper detection and response module, and a USB connector which allows us to send and retrieve data or keys.

C. **Background:** This project was inspired by a real world problem where one of our members needed to be able to store symmetric keys on the Trusted Platform Module for a project he was working on in his internship. The provided windows API (CNG) was both difficult to work with and not able to directly store symmetric keys. The goal of this project is to design a cost effective and easy to use device which allows the user to store symmetric keys on a secure hardware device. Rather than wrapping the symmetric key with an asymmetric key, this design stores the symmetric keys directly which boosts performance as asymmetric encryption can have poor performance time. This project also will have a simplistic user interface to be much easier to use than that of the TPM. We all are interested in security and believe that in addition to the other benefits of this project it will help us to better understand what goes into designing a hardware security module.

D. Physical Design

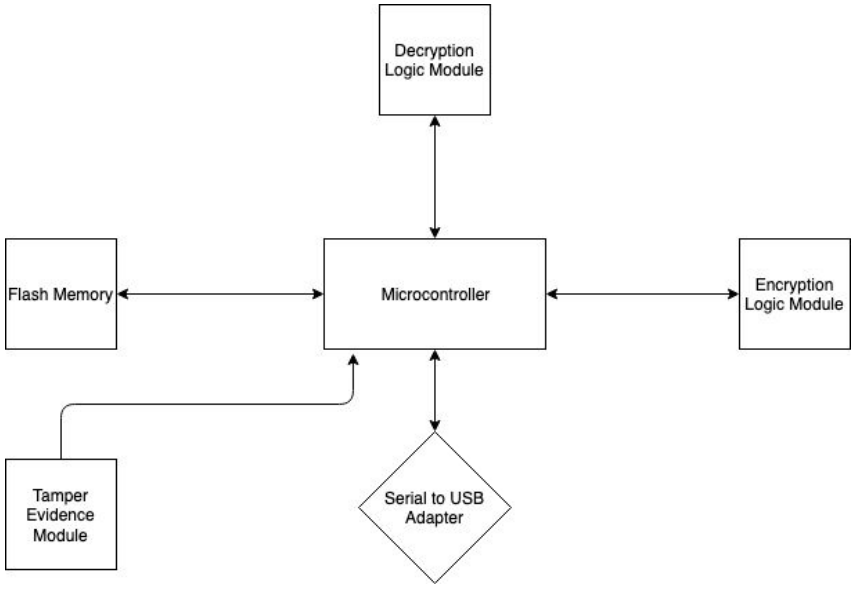


E. High-level Requirements List

1. Able to store 128-bit AES keys in nonvolatile memory.
2. Able to encrypt and decrypt a 128-bit chunk of data using a selected ket stored on device.
3. Evidence is apparent on the device in the case of an outside attacker attempting to gain access to the internal circuitry. Able to delete stored keys in certain tamper evident cases.

II. Design

A. Block Diagram



B. Functional Overview / Block Requirements

1. **Microcontroller:** This is the control room of our design and is at the center of the block diagram because it is what connects all of the individual modules and powers our device. We have been looking at different microcontrollers and feel that the ATmega32u4 may be what we are looking for. This is because it contains 32k of flash memory, an internal clock, and a serial to USB interface. The microcontroller will take in data or keys and will route it to the correct place in order to operate correctly.
2. **Flash Memory:** This is where we will store our 128-bit AES keys. We will index them so that we can select a key for use and with the capabilities of the above microprocessor we will have the ability to store up to 2,000 keys. This will interface with the microcontroller and can send and retrieve keys depending on the desired outcome.
3. **Encryption Logic Module:** This module is going to be a hardware implementation of 128-bit AES encryption. This will be created through multiple integrated circuits which will be connected as is outlined in the AES encryption algorithm. The microcontroller will send in a key and data to be encrypted and will run through our AES designed logic circuit and output the encrypted data back to the microcontroller.

4. **Decryption Logic Module:** This module is going to be a hardware implementation of 128-bit AES decryption. This will be created through multiple integrated circuits which will be connected as is outlined in the AES decryption algorithm. The microcontroller will send in a key and data to be decrypted and will run through our AES designed logic circuit and output the decrypted data back to the microcontroller.
5. **Tamper Evidence Module:** The tamper evidence module will involve buttons which are pushed down by the external casing of the device. In the event of tampering which involves the casing being removed, this module will wipe the flash memory. This is done by the module sending an alert signal to the microcontroller which will send a signal to the flash memory to wipe it. In addition to this we will be coating our finished device in an epoxy-like resin and covering it with a conductive mesh. These will likely not elicit a response in our circuit, however they will show evidence of tampering.
6. **Serial to USB Adapter:** This is the final piece which connects to the microcontroller. We will have software implemented which allows the user to send data to our device in two forms: data to be encrypted or a key pair which holds a 128-bit AES key and an index in the flash memory that it will be stored.

C. Block Requirements
    1. **Microcontroller:** We are looking for a microprocessor which has flash memory, a USB adapter, the ability to send and receive data to each module, and the ability to power all of our modules.
    2. **Flash Memory:** We are looking for a memory space capable of storing a large amount of keys. It also has to be a form of nonvolatile memory where the data will not be destroyed when the device is powered off. For this reason we chose to have a flash memory chip with at least 32kB of memory.
    3. **Encryption/Decryption Logic Modules:** These modules will largely be a combination of logic gates which will model AES encryption/decryption in hardware. This is similar to our FPGA implementation as designed in ECE 385 however we will be using a microprocessor and many integrated circuits rather than a programmable circuit such as the FPGA. The microprocessor must be able to provide clock cycles which allow the gates to properly repeat a few pieces of the algorithm multiple times.
    4. **Tamper Evidence Module:** We need to properly lay out buttons in step with our casing which will be pressed down while the device is encased. The desired outcome is that if one piece of the case is opened the module will alert the microprocessor of tampering and delete all keys.

5. **Serial to USB Adapter:** There must be software in place which allows the user to store keys and encrypt data using the other modules. We want the device to be easy to interface with on a computer so that it is easy to use.

III. Ethics & Safety

A big ethical issue to keep in mind is the security of our device / module. It would be unethical to market a hardware security module device that is unsecure as this would be highly unethical and unprofessional to make a non working product. It would also be unethical to sell a faulty product to the consumer. Therefore the security of our project will be our number one priority.

We will have to be careful when soldering to prevent injury. One of the biggest dangers of soldering is touching the tip of the soldering iron. We can mitigate this risk by not grabbing close to the iron and by using tweezers when soldering components. When it is not in use, the soldering iron will be returned to its stand and powered off. We have to be safe when testing our circuit by avoiding contact with any live wires or components that may harm us.