National University of Sciences and Technology
School of Electrical Engineering and Computer Sciences
(SEECS)

Hardware Wallet
for Cryptocurrencies

*"keeping your digital assets **safe** and **secure**"*

Group Members:
Muhammad Fahad Baig
Haider Tariq
Safi Majid

Faculty Advisor:
Dr. Syed Taha Ali

Faculty Co-Advisor:
Dr. Osman Hassan

Mandatory Slide Requirement

- FYP Report reviewed by Advisor
- FYP Report uploaded on PMS
- FYP Demo reviewed by Advisor
- FYP Demo uploaded on PMS
- Course Feedback of all courses submitted on CMS

Results and Lockdown

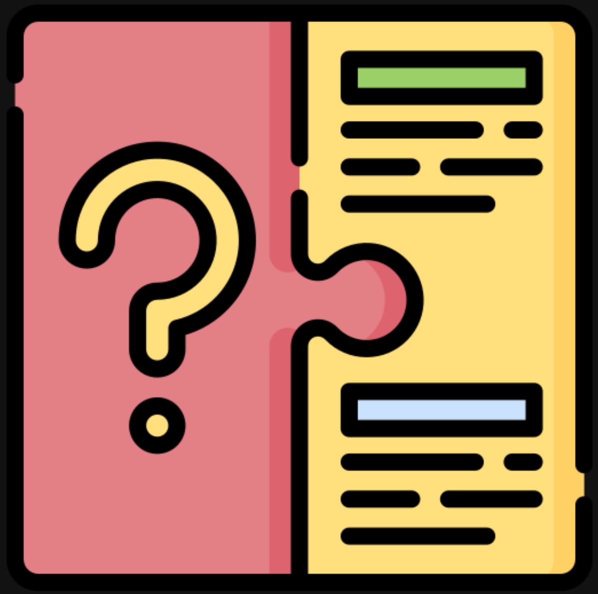
What we were able to do?

- Prototype designed on Raspberry Pi with storage encryption, wipe PIN and device spoofing protection
- Support for multiple bitcoin based cryptocurrencies with basic wallet management features
- Paired with an attractive front-end interface which is user friendly
- Device genuineness attestation using bootloader on Cortex M3/M4 architecture with MPU protection

What we were unable to do?

- ⊗ Run the bootloader and Trezor One firmware on development board (hardware unavailability). Code is implemented and algorithm wise should work.
- ⊗ Print PWB design for the development board variant (schematics are complete)
- ⊗ Print PCB design for bootloader and Trezor variant (schematics are complete)

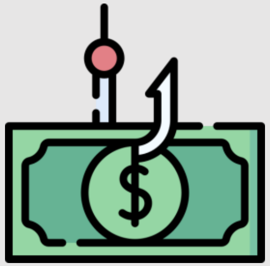
Problem Statement



Rise of threats

In 2019 alone!

50 million total global detections observed by Malwarebytes.



Webroot observed a **640%** increase in phishing attempts

55% of HTTPs hosting attacks were on crypto exchanges

300 million $ worth of cryptocurrency and **510,000** logins were stolen

Problem Statement

Cryptocurrency wallets exist to safeguard the underlying threat of theft but the safety measures of various wallets are questionable due the rise in cyber thefts of such solutions.

The possible securest solution is the use of hardware wallets, but the lack of an open framework and non empirical study has hampered the research advancement in prediction of future unknown threats on the platform.

Bitcoin and Cryptographic Primitives

What is Bitcoin?

A virtual currency, just like cash. It's just not widely accepted everywhere.

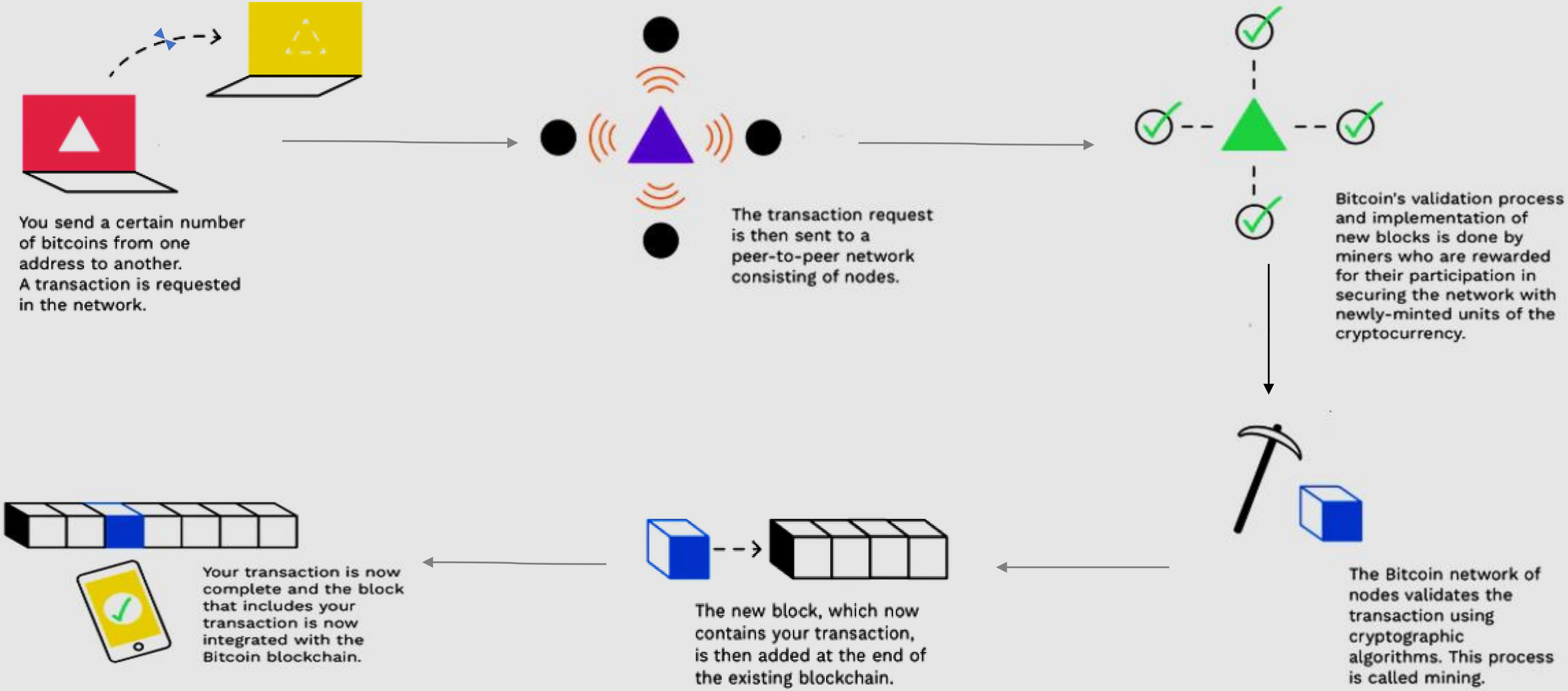
It's nothing more than a 'computer file' stored on a digital wallet.

You can send and receive payments from anywhere in the world.

Every transaction is stored in a public ledger known as blockchain.

You can buy and sell bitcoins or create it through crypto-mining!

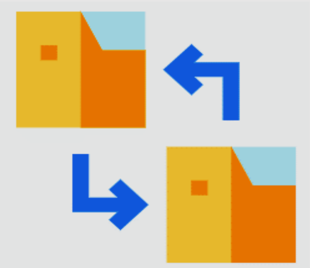
A blockchain is a special type of database. Transactions are not governed by a single party, but rather the entire transaction history is recorded in a decentralized, distributed ledger



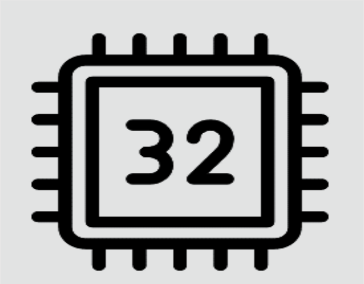
Blockchain continued...

- *Blockchain helps in reducing risks and brings transparency to the system . Furthermore, this technology helps us stamp out fraud and is therefore heralded as a promising and revolutionary technology*

- *The Chain is made up of multiple blocks and each block is made up of three basic elements:*



The data that the block consists of.



A whole number called nonce which is 32-bit.

A 256-bit number hash wedded to the nonce.

Cryptographic Primitives Examples

- ECDSA
- Cryptographic Hash Functions
- Secure Hash Algorithm or SHA
- Asymmetric Cryptography
- Hash based Message Authentication Code or HMAC
- Digital Signatures

Elliptic Curve Digital Signature Algorithm (ECDSA) Algorithm

The ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographically secure digital signature scheme, based on the elliptic-curve cryptography (ECC). ECDSA relies on the math of the cyclic groups of elliptic curves over finite field

Generation the public key and the private key

Encryption of data using generated public key

Decryption of data using generated private key

1. Public and Private Key Generation Procedure

ECDSA uses cryptographic elliptic curves (EC) over finite fields in the classical Weierstrass form. These curves are described by their EC domain parameters

For public and private keys Generation, A or/and B performs the following steps:

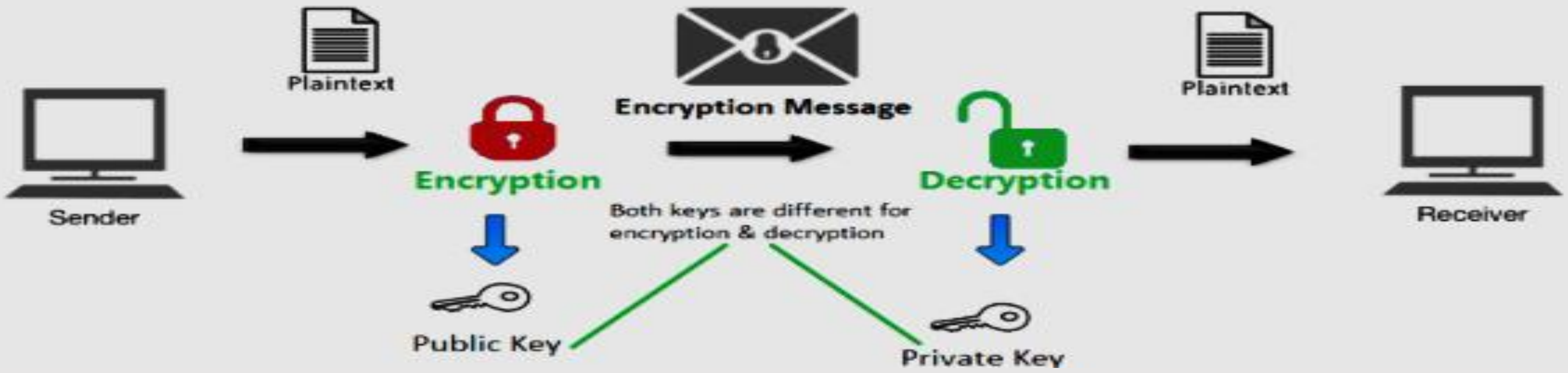
1. Select two large prime numbers, p and q. The larger the values, the more difficult it is to break RSA, but on the other hand the encoding and decoding takes longer to complete.
2. Calculate $n = pq$ and $z = (p - 1)(q - 1)$.
3. Choose a number e, less than n, that has no common factors, other than 1, with z or their greatest common divisor (gcd) equals 1, $gcd(e, z) = 1$. In this case, e and z are said to be relatively prime. e will be used in encryption.
4. Find and select a number d, such that $ed - 1$ is exactly divisible by z. In another way $ed \bmod z = 1$. d will be used in decryption.
5. The public key that B or A makes available to the world is the pair of numbers (n, e) and the private, which must be secret, is the pair of numbers (n, d).

2. Encryption of data using generated public key

Let's Suppose A wants to send B a message, which is represented by bit pattern m (plaintext message), where m < n. The encrypted value c (ciphertext) of plaintext message m is c = me(mod n). Ciphertext c will be sent to B. Note that A encrypts message using B's public key

3. Decryption of data using generated private key

To decrypt the received ciphertext c B computes m = c^d mod n which requires use of his private key (n, d). The security of ECDSA/RSA depends on the fact that there are no known algorithms for quickly factoring (prime factorization) a number. In this case the public value n into p and q.



Cryptographic Hash functions

A cryptographic hash function is an algorithm that takes an arbitrary amount of data input—a credential—and produces a fixed-size output of enciphered text called a hash value, or just "hash."

- *Non-reversibility, or one-way function.*
- *Diffusion, or avalanche effect.*
- *Determinism.*
- *Collision resistance.*
- *Non-predictable.*

Secure Hash Algorithms, also known as SHA

This are a family of cryptographic functions designed to keep data secured. It is an algorithm that consists of bitwise operations, modular additions, and compression functions.

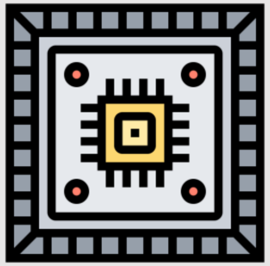
- *One-way functions*
- *Second pre-image resistance*
- *Collision resistance*
- *Resistant to length extension attacks*

Proposed Solution

NUST

Hardware Wallets

Provides more security as well as more flexibility



Transactions are made online but are stored in an offline device . The private keys are also stored in an offline device

All of Our data is encrypted and secure.

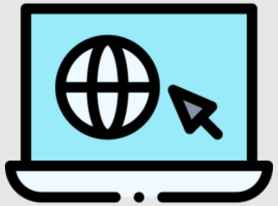
PART 4

Implementation Frameworks

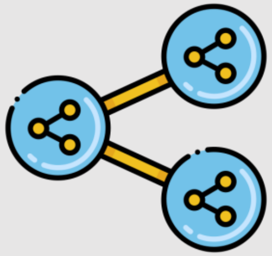
NUST

Frameworks

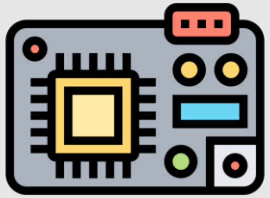
Built on Raspberry Pi, coded in Golang cross-compiled for ARMv7. Acts as backend and serves all wallet functionalities



Built on Angular and Typescript. Front-end interface that performs high level wallet interaction routines



Connector API written in Gorilla Mux that allows web interface to interact with wallet backend

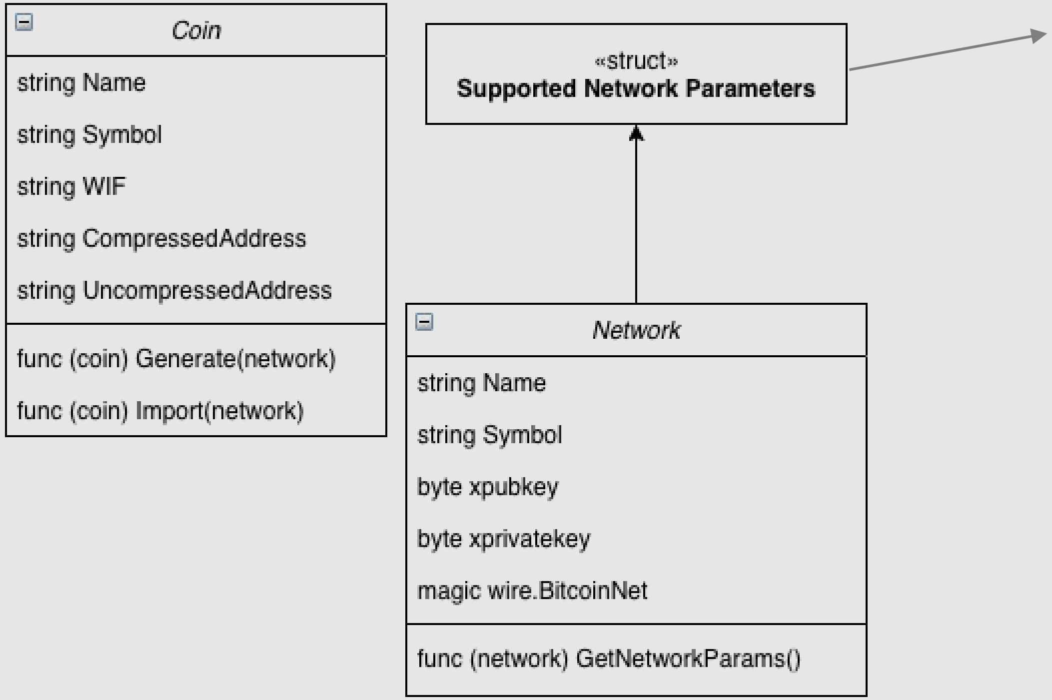


Written in pure C for Cortex M3/M4. Enables MPU and performs device attestation/genuineness checks

Algorithms Design

Main Wallet Components

Classes, Variables and Helper Functions




```
var network = map[string]Network{
  "btc":
{          name: "bitcoin",
           symbol: "btc",
           xpubkey: 0x00,
           xprivatekey: 0x80,
           magic: 0xf9beb4d9},
  "ltc":
{          name: "litecoin",
           symbol: "ltc",
           xpubkey: 0x30,
           xprivatekey: 0xb0,
           magic: 0xfbc0b6db},
  “rdd":
{          name: “rdd",
           symbol: “redcoin",
           xpubkey: 0x4c,
           xprivatekey: 0xcc,
           magic: 0xd9b4bef9},
}
```

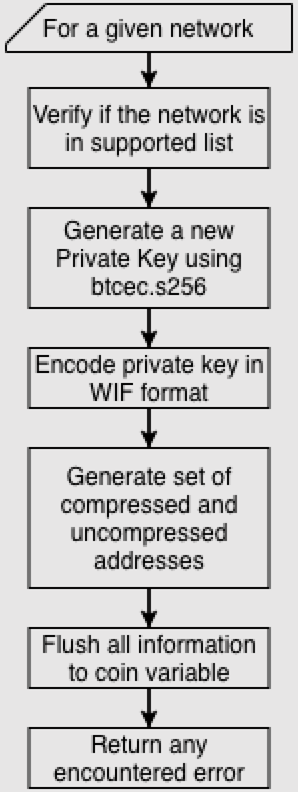
Creating Coins

func(coin) Generate(network) *func(coin) Import(network)*

Allows generating either a fresh coin or takes WIF key as in input and creates new coin based on that.

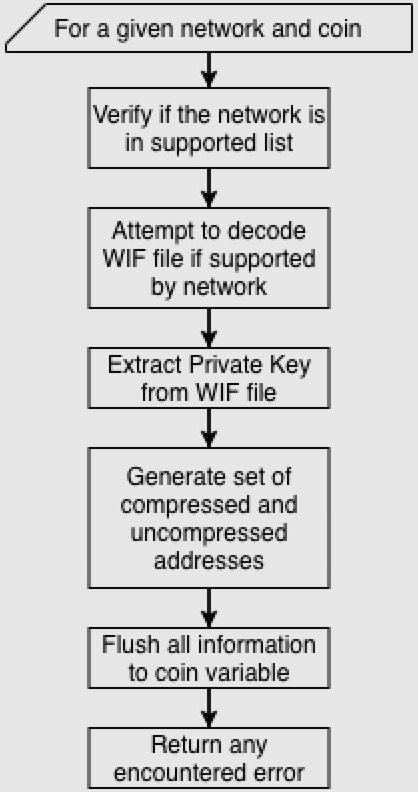
Generate

Creates new coin given network type



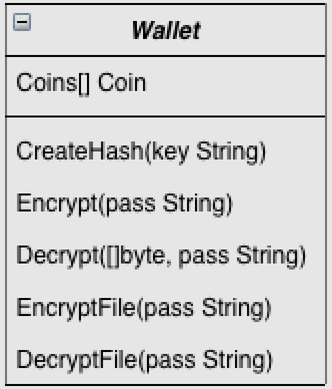
Import

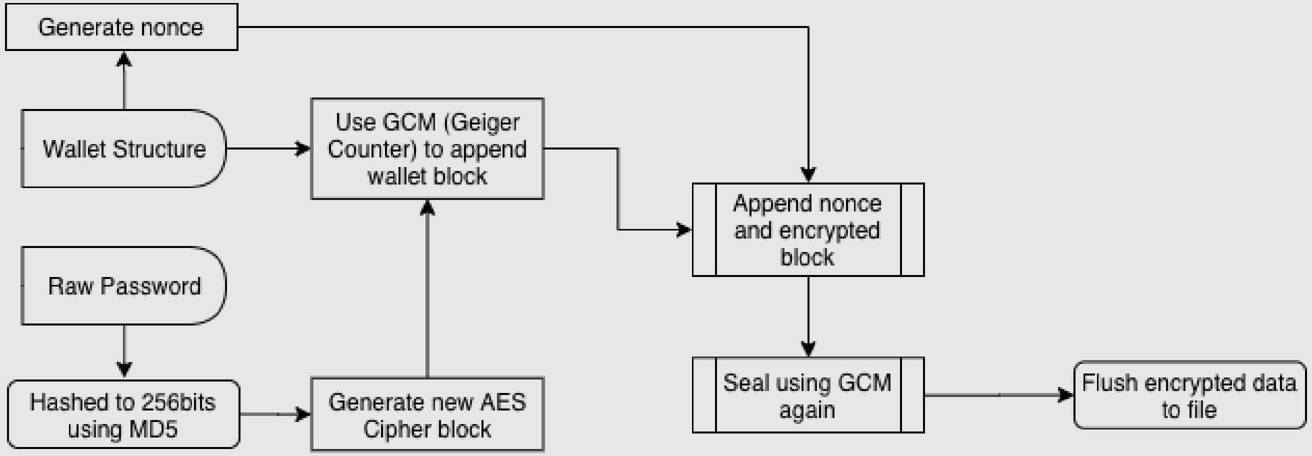
Imports existing coin to wallet



Wallet Structure and Encryption

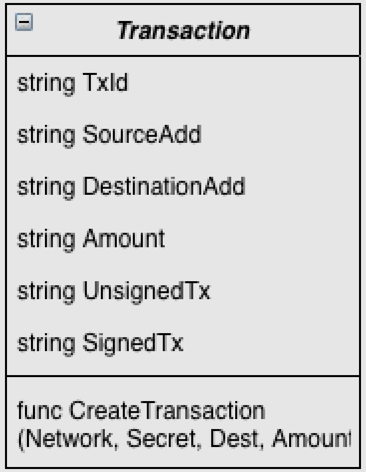
func (wallet wallet) encrypt(byte wallet_struct, string hashed_key) string cipheredText*

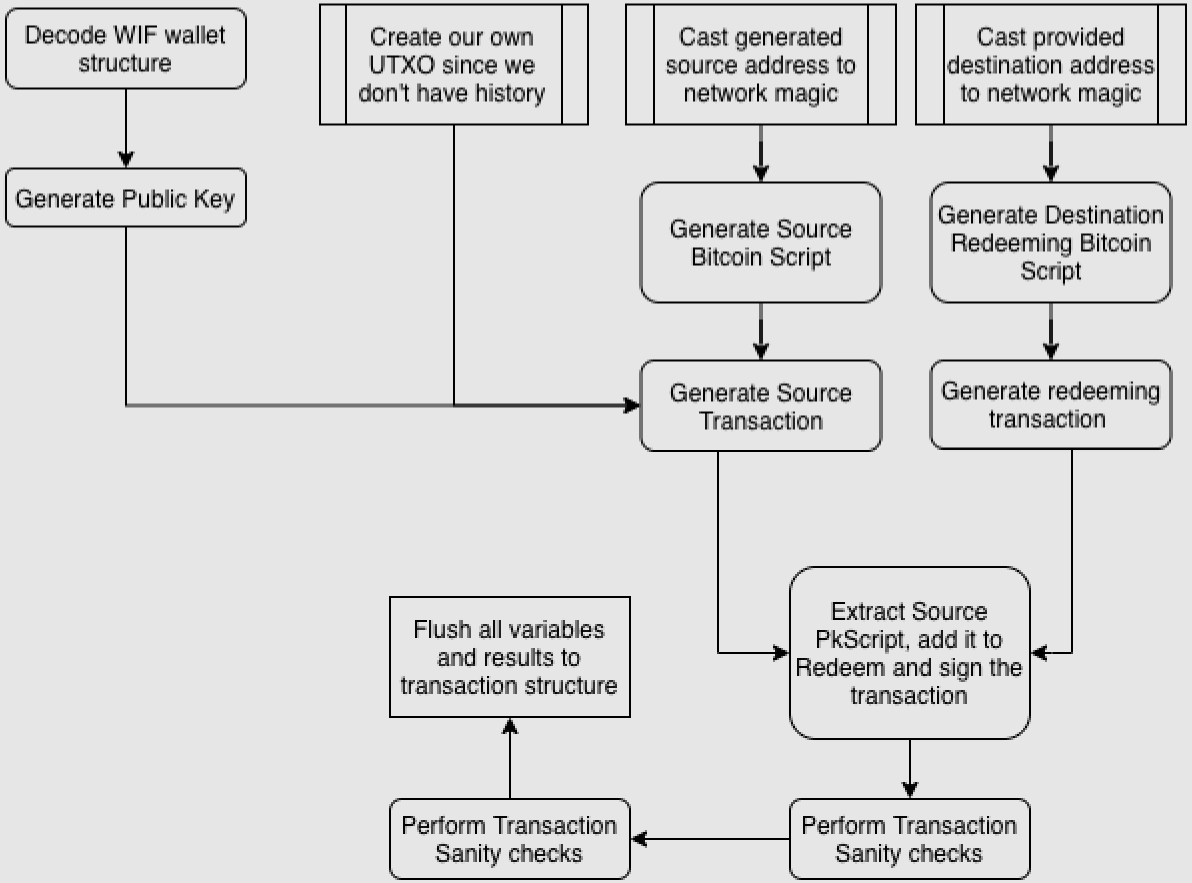




Transaction Creation and Signing

func (null) CreateTransaction(Network, Secret, Destination, Amount) json transactionData





API Endpoints

Implemented using Gorilla Mux, acts as higher level functionality for the wallet backend

/api/wallet

- CreateWallet(req POST)
- DestroyWallet(req DELETE)
- DumpWallet(req GET)
- BackupWallet(req GET)

/api/coin

- CreateCoin(req POST)
- ImportCoin(req POST)

/api/authenticate

- AuthenticateWallet(req POST)

/api/address

- GetAddresses(req GET)

/api/transaction

- CreateTransaction(req POST)

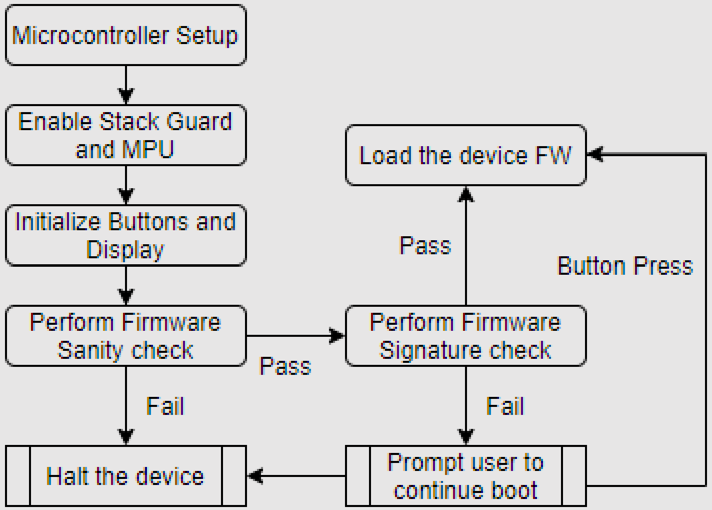
End User Interface

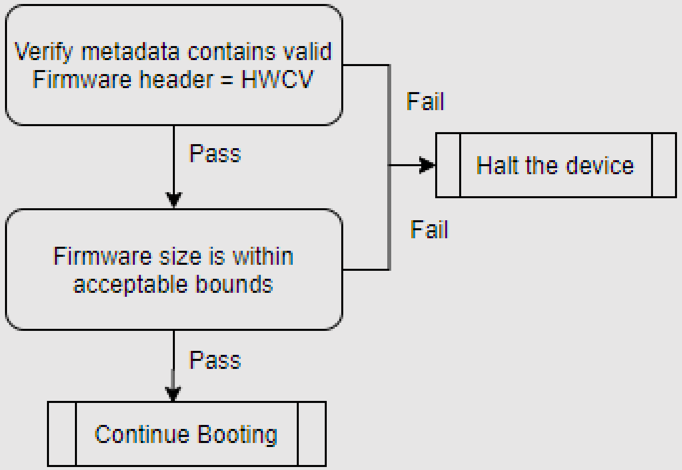
HTML, CSS, JavaScript, Angular, Node, TypeScript and a Docker/NPM build environment

It's pretty much bunch of non-interesting code and linear programming so we'll skip this.

Bootloader Functionality

Performs device genuineness attestation by verifying firmware and enables MPU to lock RW access





Enabling Memory Protection Unit

Locks all JTAG, SW and other debugging and FW write interfaces.

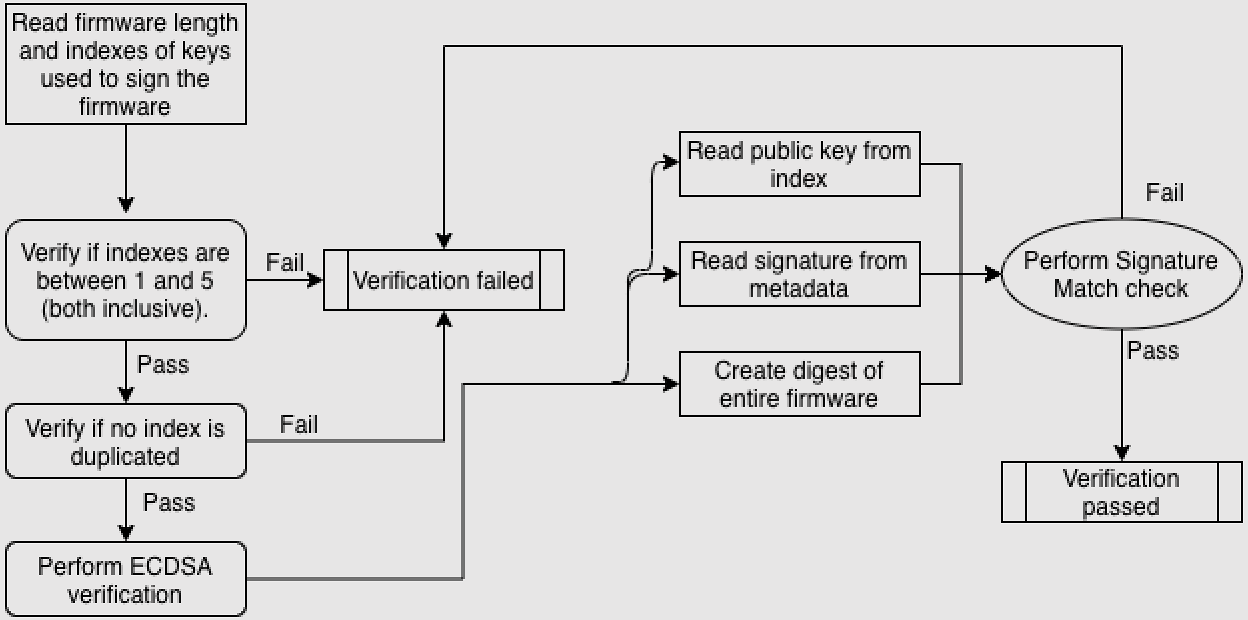
F	F	F	C	C	C	F	F

0xFFFC to the upper 2 bytes to enable Write Protection for the bootloader
0xCCFF to the lower 2 bytes to enable Read Protection Level 2.

This literally destroys a JTAG fuse on the device. It's a physical alteration and hence irreversible.

Device Firmware Verification

bool signatures_check()



Where’s the code?

```
void where_is_the_code(question code) {
          if (code == not_on_slides)
                    throw “Just request after the demo”
          else
                    throw “Sad face”
}
```

We can show the code for all algorithms above after this presentation!

PART 6

Future Work
Considerations and Improvements

Future Considerations and Improvements

Addition of a Secure Element (STM 321)

Setting attack and vulnerability vectors

Add device interaction and UI improvements

Research on a pure stateless wallet design

Possible use of a Trusted Execution Environment

Implement a Microcontroller chaining protocol

Thank You

We will now present the Demo video:

Please visit this link:
https://bit.ly/demoFYP
on your browser of your choice. The link is dropped in the messages too ☺