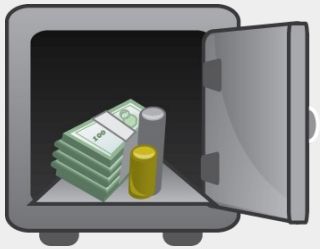
Building a Microcontroller Bitcoin Address Generator

A Little About Me...

- Software Engineer @ Microsoft in Pittsburgh
- Run https://chaintuts.com creating cryptocurrency & blockchain related tutorials
  - Articles, videos, and code projects
  - On YouTube, Twitter, Github
  - Support: Patreon, Crypto, Spreadshirt Apparel
- Focus is on understanding & teaching core concepts

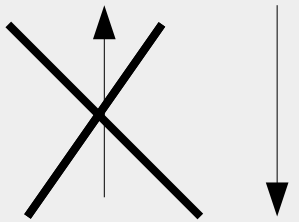
First, A Quick Disclaimer

- I’m not a cryptography/security expert, but I am a software engineer
- Security is a Complex, evolving topic
- This is a proof-of-concept for building open hardware, open source cryptocurrency tools



The Core of Crypto Ownership: Private Keys

0x12351bc143badf2348fe38e8f8b785b...

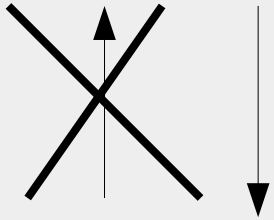


Elliptic Curve
(secp256k1)

PRIVATE KEY

0x04135981abcd7f7a7d7b7c720….

PUBLIC KEY



“Double hash” (SHA-256
and RIPEMD160)
And Base58check encoding

1MT3uNoFLP82j2aSD5Qtibm2kXJ7RWumAM

ADDRESS
(PUBLIC KEY
HASH)

The Core of Crypto Ownership: Private Keys

- Private keys prove ownership of funds using digital signatures
- Anyone with the keys *can spend the funds*
- It’s absolutely critical that keys are both:
  - Securely generated
  - Securely stored

Offline Wallets

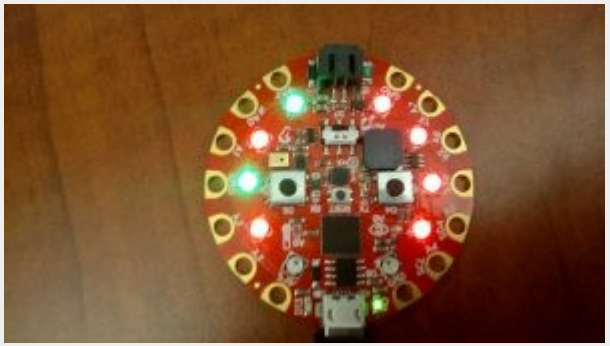
- Ex: Hardware Wallets (Keepkey, Trezor, etc.) and Paper/Metal/etc. Wallets
- Keys are generated and stored on a device not connected to a network
  - Could be from a seed phrase, or random
  - Generated on dedicated hardware, or an offline PC

Offline Wallets

- Typical solutions are commercial devices, with open source firmware
  - Trezor, Ledger, KeepKey
  - I asked, why can’t I build one of these myself on cheap hardware??
  - Thus, uBitAddr was born!

The Idea

- It would be super cool to generate real, offline keys with my own code
- I had some basic microcontroller experience from another project (visualizing proof-of-work)
- Let’s build something with readily available, “easy” to code products



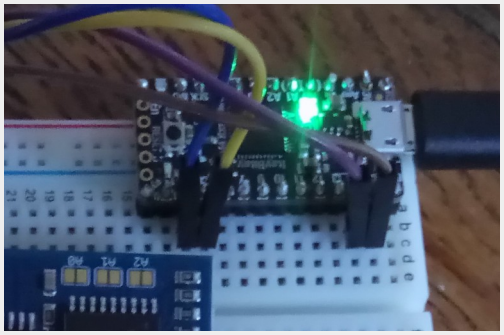
Proof-of-work simulator
On the Circuit Playground

The Challenges

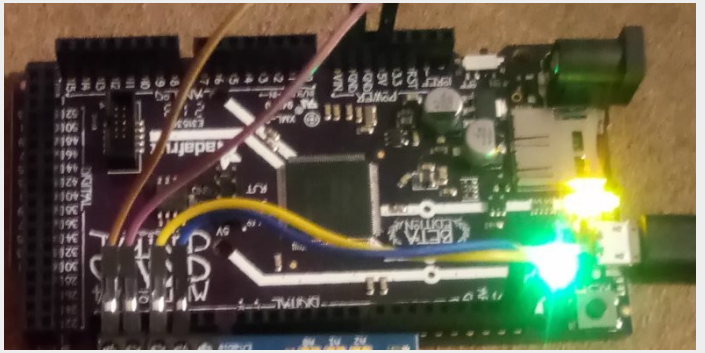
- Need a platform with enough memory to store, run code
  - First experiments shows the M0 platform was insufficient
- Cryptographic primitives – need:
  - SHA-256, RIPEMD160, Keccak (for ETH) hashes
  - Secp256k1 ECDSA algorithm
- MUST have cryptographically secure RNG

Answering the Challenges

- Memory, power
  - Decided on the Adafruit M4 platforms
  - Grand Central, ItsyBitsy tested
  - Runs on Atmel SAMD51, 256KB+ flash and 192KB+ RAM



ItsyBitsy M4



Grand Central M4

Answering The Challenges

- How to get Crypto Primitives? - Use the Open Source!
  - Standalone Python libs are hard to find and run on micro platform
  - Trezor firmware is open source, standalone C code
  - Ported Trezor code (more on that later)

Answering The Challenges

- Need for true random number generation
  - Platform choice fixed this for us!
  - M4 microcontrollers used all have built in true random number generation built in
  - Other option *could* have been to read data from an accelerometer or used other suitable environmental noise

Putting It Together

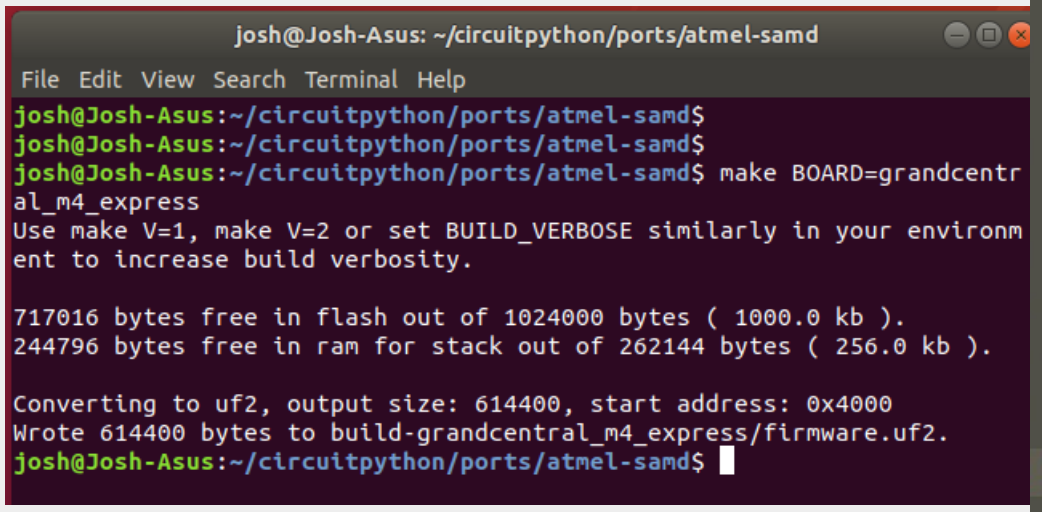
- The stack:
  - Custom CircuitPython module, C
  - Application, CircuitPython
  - Device: M4 processor with I2C Character LCD screen (mostly used) or a receipt printer
  - Supported currencies: BTC, BCH, ETH, LTC, DGB

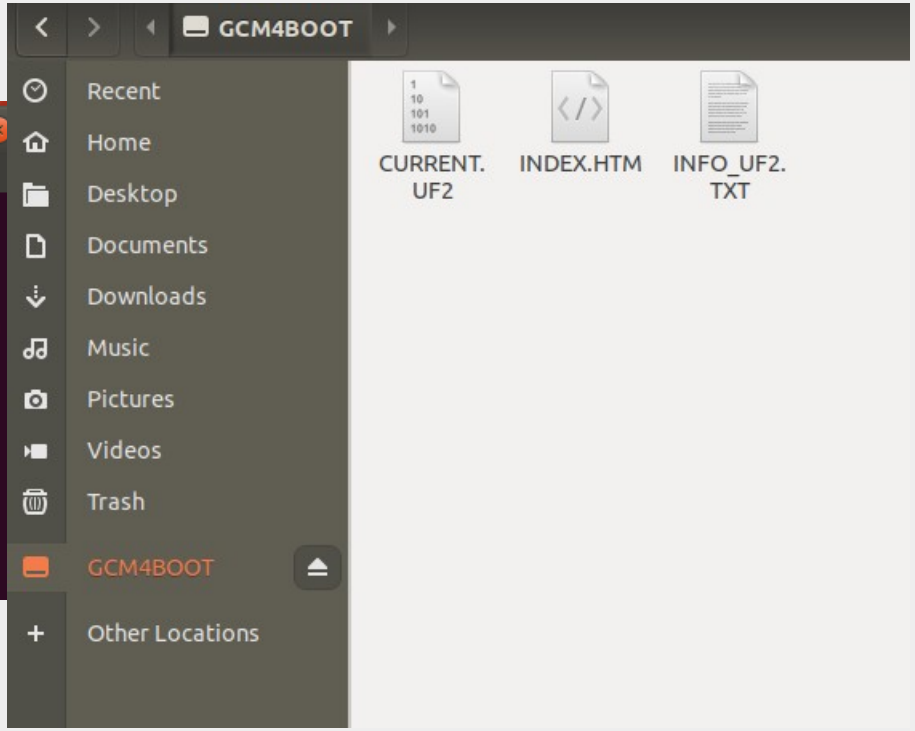
Putting It Together

- Custom CircuitPython Module
  - Followed “Extending CircuitPython” by Dave Astels
  - Where the crypto happens (thanks to Trezor)
  - Gets CRNG seed passed in from Python layer
  - Hashes seed into private key, generates pubkey, returns encoded keypair
    - Chain-appropriate address encoding
    - WIF or HEX privkey

Putting It Together

- Custom CircuitPython Module
  - Have to build firmware and load on the device (UF2 format)





Putting It Together

- Custom CircuitPython Module
  - Trezor crypto-primitives and encoding functions are pretty standalone
  - Made simple CLI testing possible

```
josh@Josh-Asus:~/circuitpython/shared-module/bitaddr$ ll
total 428
drwxr-xr-x  2 josh josh   4096 Aug  5 09:38 ./
drwxr-xr-x 28 josh josh   4096 Aug  3  2019 ../
-rw-rw-r--  1 josh josh   2817 Aug  7  2019 base58.c
-rw-rw-r--  1 josh josh   1460 Aug  7  2019 base58.h
-rw-rw-r--  1 josh josh  32293 Aug  7  2019 bignum.c
-rw-rw-r--  1 josh josh   5200 Aug  7  2019 bignum.h
-rw-r--r--  1 josh josh   6319 Sep 11  2019 cash_addr.c
-rw-r--r--  1 josh josh   3429 Sep 10  2019 cash_addr.h
-rw-rw-r--  1 josh josh  21541 Aug  7  2019 ecdsa.c
-rw-rw-r--  1 josh josh   3814 Aug  7  2019 ecdsa.h
-rw-r--r--  1 josh josh  12153 Mar 14 20:16 __init__.c
-rw-r--r--  1 josh josh    497 Aug  7  2019 memzero.c
-rw-r--r--  1 josh josh    123 Aug  7  2019 memzero.h
-rw-rw-r--  1 josh josh   2570 Aug  7  2019 options.h
-rw-rw-r--  1 josh josh   2678 Aug  7  2019 rand.c
-rw-rw-r--  1 josh josh   1494 Aug  7  2019 rand.h
-rw-r--r--  1 josh josh  10195 Aug  7  2019 ripemd160.c
-rw-r--r--  1 josh josh    762 Aug  7  2019 ripemd160.h
-rw-rw-r--  1 josh josh   2215 Aug  7  2019 secp256k1.c
-rw-rw-r--  1 josh josh   1300 Aug  7  2019 secp256k1.h
-rw-rw-r--  1 josh josh 124720 Aug  7  2019 secp256k1.table
-rw-r--r--  1 josh josh  16501 Aug  7  2019 sha2.c
-rw-r--r--  1 josh josh   4703 Aug  7  2019 sha2.h
-rw-rw-r--  1 josh josh  11338 Sep 19  2019 sha3.c
-rw-rw-r--  1 josh josh   3068 Sep 19  2019 sha3.h
-rwxr-xr-x  1 josh josh 110088 Mar 14 19:52 test*
```

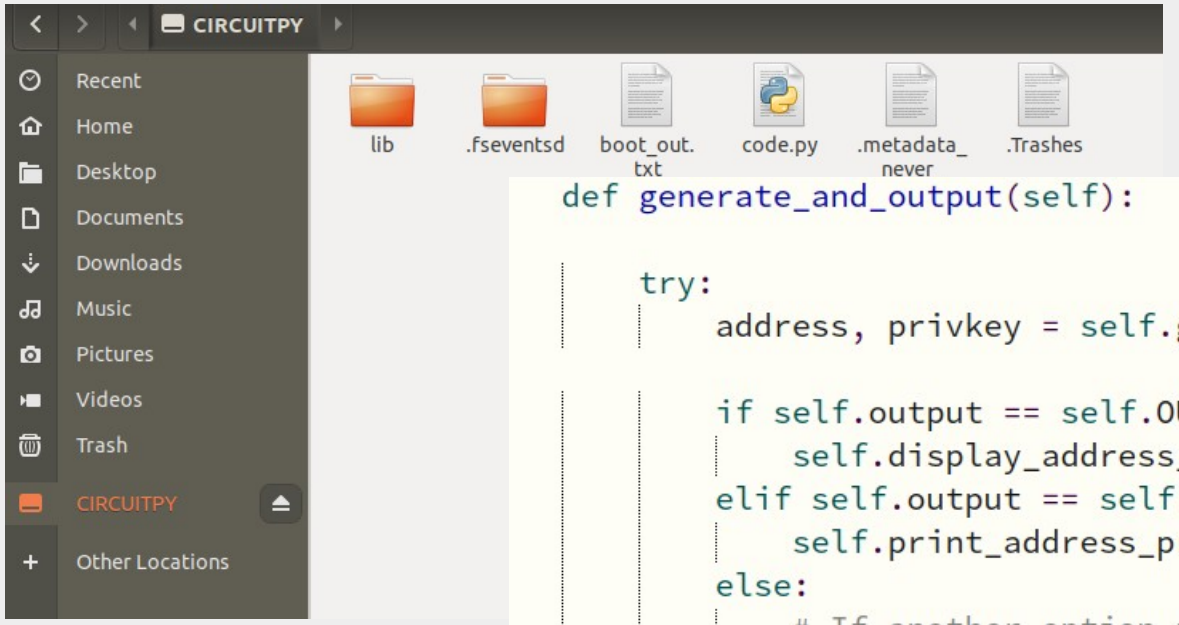
```
josh@Josh-Asus:~/circuitpython/shared-module/bitaddr$ ./test
DSu9C81NQCqmM2X4Ga99ULWarn8yH7TNKV
5JCtUcQv5w1Jj3SEeX1J8PpM2ByWPxzTexXHXoBdFoYKEWEERoi
```

Putting It Together

- CircuitPython layer
  - Generates entropy (Python os.urandom, which uses the built in CSRNG)
  - Calls custom module code to get privkey, address
  - Displays on LCD screen or prints to receipt printer, etc. using Adafruit libraries

Putting It Together

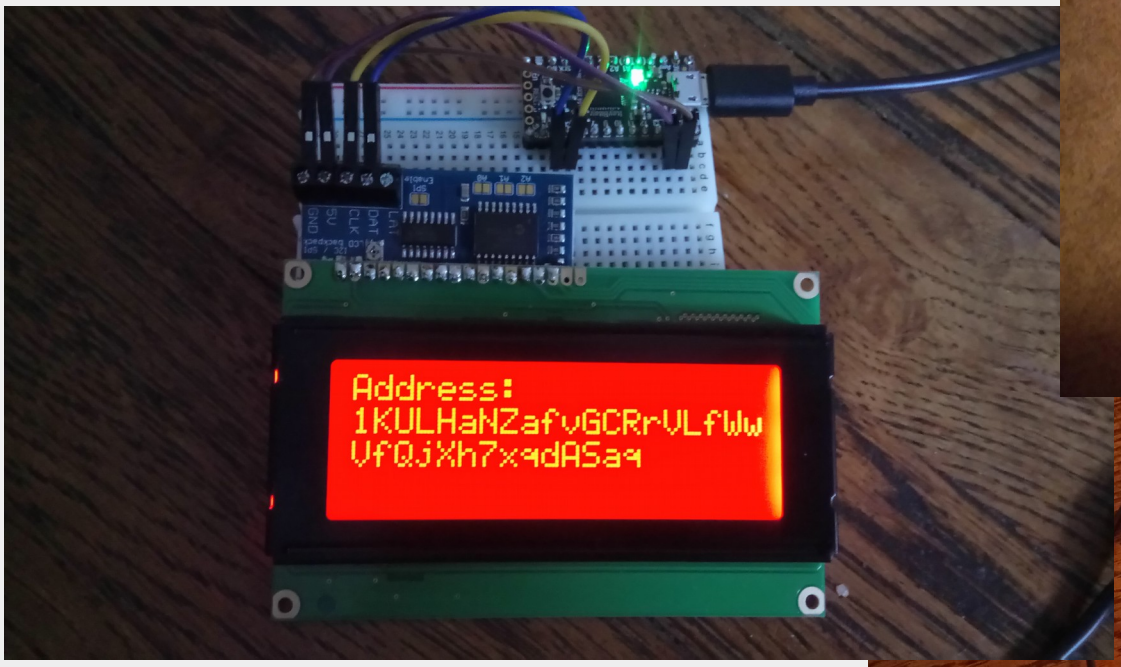
- CircuitPython layer
  - Load code into CIRCUITPY drive
  - Usage is then simple – each reset generates and shows a new keypair!

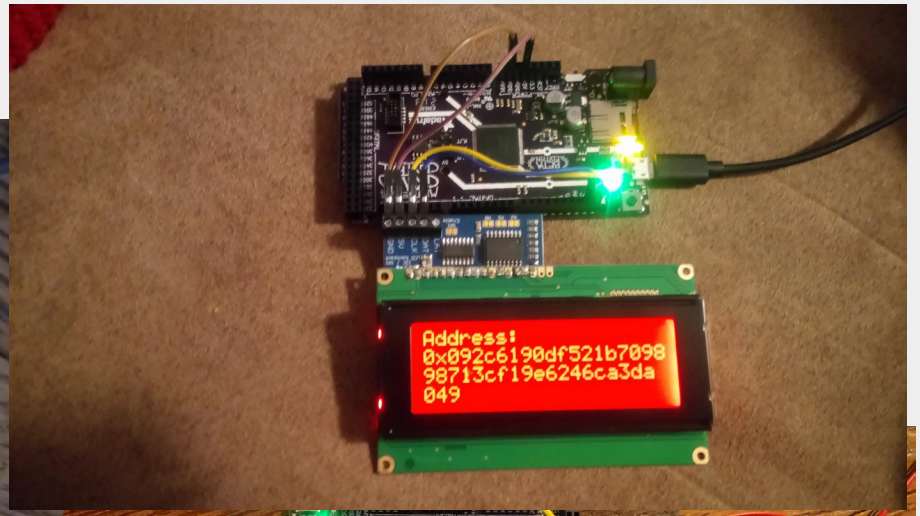



```
def generate_and_output(self):

    try:
        address, privkey = self.generate_address_privkey()

        if self.output == self.OUTPUT_DISPLAY:
            self.display_address_privkey(address, privkey)
        elif self.output == self.OUTPUT_PRINTER:
            self.print_address_privkey(address, privkey)
        else:
            # If another option isn't available, print to serial
```

uBitAddr

https://github.com/chaintuts/ubitaddr/tree/development





Potential Pitfalls

- Cryptography Foot-Guns
  - Again, I’m not an expert here
  - There’s always hidden dangers with implementations
    - Used trusted primitives
    - Kept things simple
    - Tested, tested, tested manually against known-working implementations

Potential Pitfalls

- Single keypairs are out-of-favor/legacy
  - MUST make sure to copy correctly, double check before sending funds
  - Encoding is not human friendly
  - Data easily corrupted (water damage, bad handwriting, character distinctions)
  - As is, this code only shows keypair ONCE

Potential Future Improvements

- Biggest: Upgrade to BIP39/BIP44 compatible seed phrases instead of single keypairs
  - Give user a seed phrase for safe backup
  - Generate new addresses as needed
  - Even if we use it for one address from tree (similar to old impl), backup is easier and safer
- UI/UX improvements
  - Warnings, store keypairs?, etc.

Potential Future Improvements

- Upgrade to CircuitPython5 – latest builds
  - Downside: build process was tricky
- Device powerful enough for pure-Python implementation?
  - Easier development, less complexity
  - Downside: Python crypto-security pitfalls?
- Unit testing to catch basic & edge cases with generation
- Open to ideas – always trying to improve!

Questions?