Building and Breaking Block Chains

Merlin Corey
Pandoblox Engineer

Shellcon 2018

Who is that Merlin guy?

- Cryptography nerd
- Null Space Labs keyholder
- LayerOne Hardware Hacking Village
- Defcon Tamper Evident Village
- DC949 Alumni
- Startup Wizard at Pandoblox

Assumptions

- Cryptographic fundamentals
- Vague ideas of what a cryptocurrency is
- Passing familiarity with Python or another language

Cryptographic Fundamentals

- Hashing is most important concept
- One-way function
- Ideally
  - Large space
  - Randomized value
  - No collisions

Vague ideas of Cryptocurrency

- Magical internet currency
- The future of everything
- Really slow database
- Pyramid scheme?

Basic programming

- TrackA = Room(‘pufferfish’)
- Merlin = Speaker()
- For each Attendee in TrackA.attendees()
  - Attendee.ListenTo(Merlin)
- Print(‘Now you know Python’)

Building Basic Blockchain: Prior Art

- https://github.com/dvf/blockchain
- https://github.com/zack-bitcoin/basiccoin

Building Basic Blockchain: Challenge

- How hard could it be?

Building Basic Blockchain: Challenge

- How hard could it be?
  - Pretty hard, honestly

Building Basic Blockchain: Challenge

- How hard could it be?
  - Pretty hard, honestly
  - But we'll keep it as simple as possible

Building Basic Blockchain: High level Components

- Transaction
- Block
- Blockchain
- Node
- Network

Building Basic Blockchain: High level Components

- Transaction
  - Inputs
  - Outputs
  - Coinbase

Building Basic Blockchain: High level Components

- Block
  - Transactions
    - Merkle Root
  - Proof
  - Parent block

Building Basic Blockchain: High level Components

- Blockchain
  - Blocks connected by parent blocks
  - Block #0
    - Block #1 (Child of #0)
      - Block #2 (Child of #1)
        - Block #3 (Child of #2)
          - Block #4 (Child of #3)
            - Block #5 (Child of #4)
              - …
                - Block #N (Child of #N-1)

Building Basic Blockchain: High level Components

- Node
  - Miner
  - Wallet

Building Basic Blockchain: High level Components

- Network
  - Nodes
  - Blocks
  - Protocol

Build Basic Block Chain: Transaction Input

```
def make_transaction_input(previous_txid_hash, previous_txid_index, data):
    '''Return a transaction input with parent output hash, index, and arbitrary data.'''
    return {'type': 'txin',
            'parent': previous_txid_hash,
            'index': previous_txid_index,
            'data': data}
```

Build Basic Block Chain: Transaction Output

```
def make_transaction_output(value, data):
    '''Return a transaction output with value and arbitrary data.'''
    return {'type': 'txout',
            'value': value,
            'data': data}
```

Build Basic Block Chain: Transaction

```
def make_transaction(inputs, outputs):
    '''Return a transaction with the given inputs and outputs.'''
    return {'type': 'tx',
            'inputs': inputs,
            'outputs': outputs}
```

Build Basic Block Chain: Special Transactions

```
def make_transaction_coinbase(value, data):
    '''Coinbase transactions have no inputs, only outputs to the miner.'''
    return make_transaction(inputs=[],
                            outputs=make_transaction_output(value, data))


def make_transaction_empty():
    '''Empty transactions are used to pad the Merkle tree.'''
    return make_transaction(inputs=[], outputs=[])
```

Building Basic Blockchain: Hashing

- Exploring hashing with live code

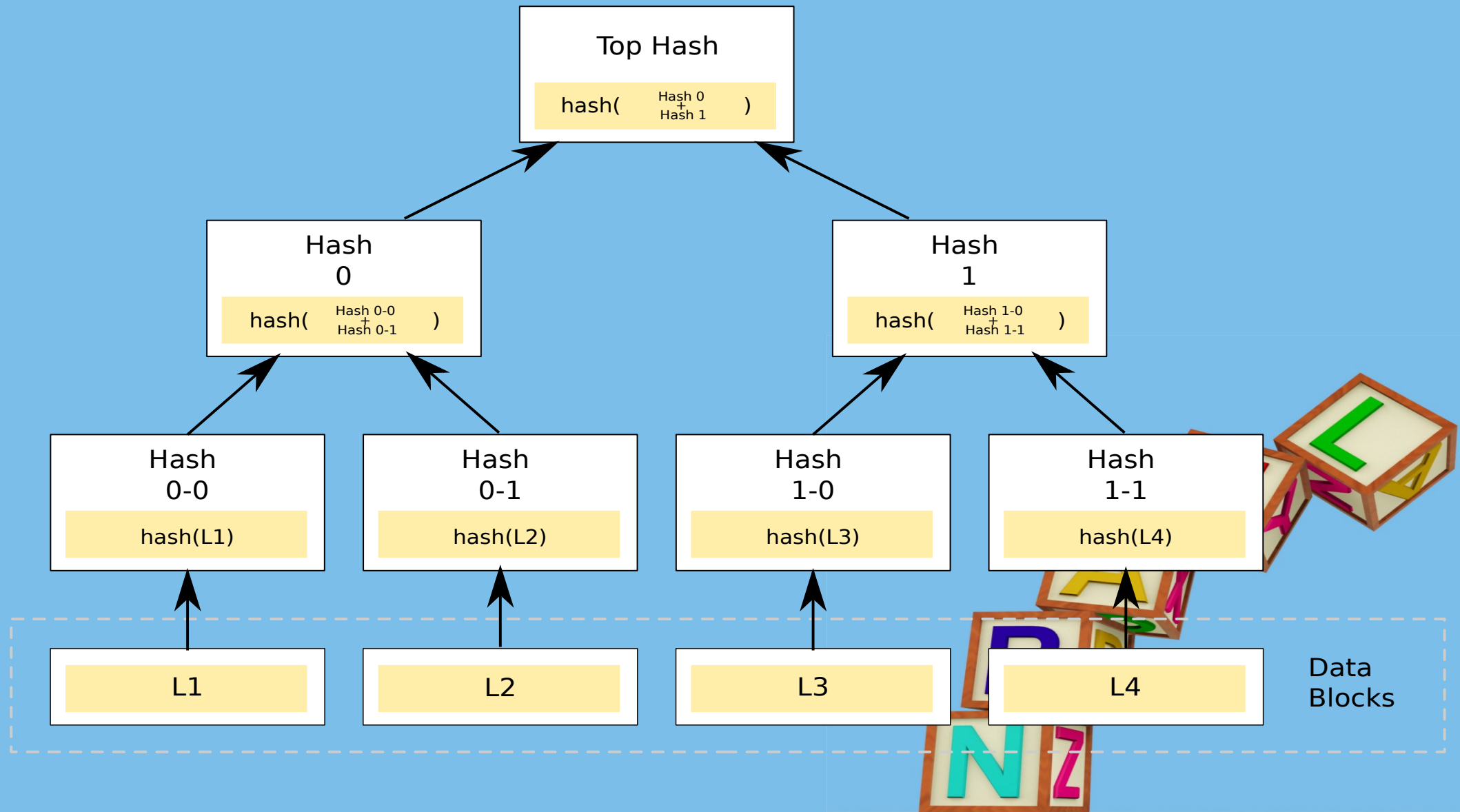
Build Basic Blockchain: Hashing

```
def make_hashable(data):
    '''Return a hashable representation of the given data.'''
    return str(data).encode('utf-8')


def sha256_digest(data):
    '''Return a SHA256 digest for the given data.'''
    return hashlib.sha256(make_hashable(data)).hexdigest()


def sha256_digest_reduce(seq):
    '''Return a SHA256 digest of a list of SHA256 digests appended to each other.'''
    return sha256_digest(''.join(seq))
```

Build Basic Blockchain: Merkle Tree



Build Basic Block Chain: Merkle Root

```
def calculate_merkle_root(transactions):
    '''Return the Merkle root of a list of transactions or None if no transactions are given.
    First pad the transactions with an empty one if necessary to make an even number.
    Then, reduce pairs of hashes to a single hash until only two remain.
    Finally, hash that to create the Merkle Root.'''
    root = None
    if len(transactions):
        transactions = list(concat(partition(2, transactions, pad=make_transaction_empty)))
        hashes = list(map(sha256_digest, transactions))
        while len(hashes) > 2:
            hashes = list(map(sha256_digest_reduce,
                              partition(2, hashes)))
        root = sha256_digest_reduce(hashes)
    return root
```

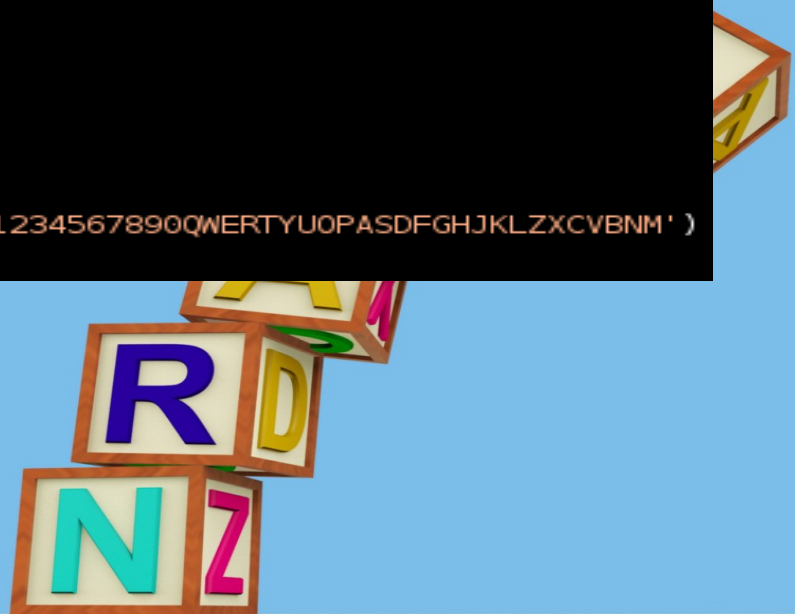
Building Basic Blockchain: Block Helpers

```
def make_block_header(previous_block_hash, merkle_root_hash, datetimestamp, difficulty, nonce):
    return {'type': 'blockhead',
            'parent': previous_block_hash,
            'root': merkle_root_hash,
            'datetimestamp': str(datetimestamp),
            'difficulty': difficulty,
            'nonce': nonce}


def hash_block(block):
    '''Return the hash of a block, excepting the 'hash' key.'''
    return sha256_digest({key: value
                          for key, value in block.items()
                          if key != 'hash'})


def count_leading_zeros(digest):
    '''Return the count of leading zeros for a block'''
    count = 0
    if digest.startswith('0'):
        count = 1
        for digit in digest[1:]:
            if '0' == digit:
                count += 1
            else:
                break
    return count


def random_chars(n):
    return ''.join([random.choice('qwertyuiopasdfghjklzxcvbnm1234567890QWERTYUOPASDFGHJKLZXCVBNM')
                    for x in range(n)])
```



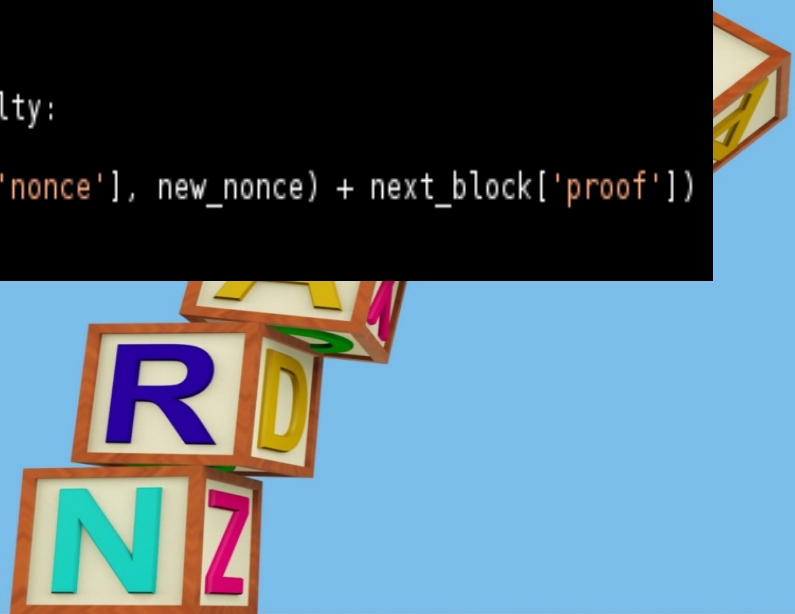
Building Basic Blockchain: Genesis Block

```
def mine_block_for_transactions(previous_block, transactions):
    '''Mine a new block given the previous block and a list of transactions. '''
    if previous_block is None:
        previous_block = {'nonce': 0,
                          'hash': '0',
                          'difficulty': 3}
    root = calculate_merkle_root(transactions)
    new_nonce = 1 + previous_block['nonce']

    min_difficulty = previous_block['difficulty']

    header = make_block_header(previous_block.get('hash', '0'),
                               root,
                               datetime.datetime.now(),
                               min_difficulty,
                               new_nonce)

    next_block = merge(header, {'transactions': transactions})
    while count_leading_zeros(next_block.get('hash', '1')) < min_difficulty:
        next_block['proof'] = random_chars(32)
        next_block['hash'] = sha256_digest('{}{}'.format(previous_block['nonce'], new_nonce) + next_block['proof'])
    return next_block
```



Building Basic Blockchain: Blockchain

- Exploring blockchain with live code

Building Basic Blockchain: Virtual Machine

- Transaction outputs are scripts

Building Basic Blockchain: Virtual Machine

- Transaction outputs are scripts
- Breathe a sigh of relief

Building Basic Blockchain: Virtual Machine

- Transaction outputs are scripts
- Breathe a sigh of relief
  - We won’t be implementing all that today!
- Listen to Merlin ramble on about it, anyway

Breaking Basic Blockchain

- What is controllable
- How is a 51% attack executed?

Breaking Production Blockchains: Smart Contracts

- There are many attacks against smart contracts

Breaking Production Blockchains: Smart Contracts

- There are many attacks against smart contracts
  - Underflow and Overflow are the most basic

```
contract OverflowUnderFlow {
    uint public zero = 0;
    uint public max = 2**256-1;

    // zero will end up at 2**256-1
    function underflow() public {
        zero -= 1;
    }
    // max will end up at 0
    function overflow() public {
        max += 1;
    }
}
```

Breaking Production Blockchains: Smart Contracts

- Understanding the DAO hack

Breaking Production Blockchains: Smart Contracts

- Understanding the DAO hack
  - Recursive function calls are dangerous

Breaking Production Blockchains: Smart Contracts

- Understanding the DAO hack
  - Recursive function calls are dangerous
  - Especially when you do work on either side of them

Breaking Production Blockchains: Smart Contracts

- Understanding the DAO hack

```
function withdraw(uint _amount) public {
  if(balances[msg.sender] >= _amount) {
    if(msg.sender.call.value(_amount)()) {
      _amount;
    }
    balances[msg.sender] -= _amount;
  }
}
```

Breaking Production Blockchains: Smart Contracts

- ERC20 Short Address Attack
  - Generate address with trailing zero
  - Send to address without trailing zero

Protecting Production Blockchains: Nodes and Wallets

- Private keys
  - Passphrases
  - Cold storage
- RTFM your configuration
- Firewalls
- Monitoring and Alerting

Protecting Production Blockchains: Network

- Economic feasibility of 51% attacks
- Like any other software: patches

Protecting Production Blockchains: Network

- Economic feasibility of 51% attacks
- Like any other software: patches
  - Bitcoin DoS [patch]
    - Bitcoin Infinite Inflation?
    - Notice of Vulnerability
    - CVE-2018-17144

Protecting Production Blockchains: Smart Contracts in Solidity

- Avoid reentrancy issues
- Be careful of overflows and underflows
- Use a library
  - Like SafeMath
- Check lengths of addresss and other data
- Use EthFiddle and test

Protecting Production Blockchains: Smart Contracts in Solidity

- Avoid reentrancy issues
- Be careful of overflows and underflows
- Use a library
  - Like SafeMath
- Check lengths of addresss and other data
- Use EthFiddle and test
  - Test

Protecting Production Blockchains: Smart Contracts in Solidity

- Avoid reentrancy issues
- Be careful of overflows and underflows
- Use a library
  - Like SafeMath
- Check lengths of addresss and other data
- Use EthFiddle and test
  - Test
    - Test!

Questions and Contact

- Any questions?
  - If you’re still awake, that is
- Want to talk to Merlin?
  - Come check out NSL 4.0 starting late October!
  - Hang out on EFNet in #NSL
  - Hand him a drink at any conference!