IT486: Blockchains and Cryptocurrencies

Key management: wallets and exchanges

Digression: Quantum Resistance

- Signature schemes based on the integer factorization problem, the discrete logarithm problem, or the elliptic curve discrete logarithm problem
- Can be solved with Shor's algorithm with an enough powerful quantum computer
- Hash functions are considered to be relatively secure against quantum computers

Digression: Quantum Resistance

- If public key unknown, then bitcoins cannot be stolen. (If public, bitcoins are stolen)
- How do you prevent them from being stolen when you issue a transaction?
- If the quantum computer takes longer than 1-2 minutes to hack your private key, then you can transfer bitcoins if you always use a new address to transfer (to transfer, but also as a return address)
- In Bitcoin it is considered bad hygeine to reuse addresses. In a post-quantum world, it will get your funds stolen.

Storing bitcoins

- Storing bitcoin is ALL about managing your secret keys
- Different approaches, with various trade-offs:

Storing bitcoins

- Storing bitcoin is ALL about managing your secret keys
- Different approaches, with various trade-offs:
  - Availability: Being able to spend your coins when you want to

Storing bitcoins

- Storing bitcoin is ALL about managing your secret keys
- Different approaches, with various trade-offs:
  - Availability: Being able to spend your coins when you want to
  - Security: Making sure nobody else can spend your coins

Storing bitcoins

- Storing bitcoin is ALL about managing your secret keys
- Different approaches, with various trade-offs:
  - Availability: Being able to spend your coins when you want to
  - Security: Making sure nobody else can spend your coins
  - Convenience: Managing your keys (and thus your coins)

Bitcoin wallets

- Provide a practical user interface for the generation and storage of private keys
- Keyring would have been a better name, since wallets don't store money directly!

Hot vs. Cold Storage

- Hot storage (online) is convenient
  - coins can directly be spent on the Bitcoin network (i.e. node or node-proxy is online)

Hot vs. Cold Storage

- Hot storage (online) is convenient
  - coins can directly be spent on the Bitcoin network (i.e. node or node-proxy is online)
- Cold storage (offline) offers better security, but comes with some inconvenience
  - coins cannot directly be spent on the Bitcoin network (i.e. secret keys are offline)

Hot vs. Cold Storage

- Hot storage (online) is convenient
  - coins can directly be spent on the Bitcoin network (i.e. node or node-proxy is online)
- Cold storage (offline) offers better security, but comes with some inconvenience
  - coins cannot directly be spent on the Bitcoin network (i.e. secret keys are offline)
  - Note: cold storage retains the possibility of sending coins to it even if offline, however, as long as a valid address is known

Desktop Wallets (hot)

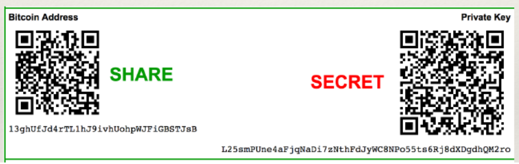
- Software on your device that keeps track of your keys, coins, makes transactions, etc.
- Think of it as the wallet in your pocket
  - simple to use and understand, but pickpockets are rife!

Desktop Wallets (hot)

- Software on your device that keeps track of your keys, coins, makes transactions, etc.
- Think of it as the wallet in your pocket
  - simple to use and understand, but pickpockets are rife!
- Availability: HIGH, Convenience: HIGH, Security: LOW

Paper Wallets (Cold)

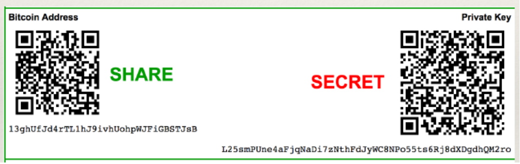
- Writing secret keys on paper instead of device gives some protection
- QR codes for easy use



- Anyone who can see it can steal it!

Paper Wallets (Cold)

- Writing secret keys on paper instead of device gives some protection
- QR codes for easy use



- Anyone who can see it can steal it!
- Availability: LOW, Convenience: MEDIUM, Security: MEDIUM/HIGH

Hardware Wallets (Hybrid hot/cold)

- Simple devices (not full computers) which generate keys but secret keys never leave device
- Only public keys/addresses/signatures allowed to escape device

Hardware Wallets (Hybrid hot/cold)

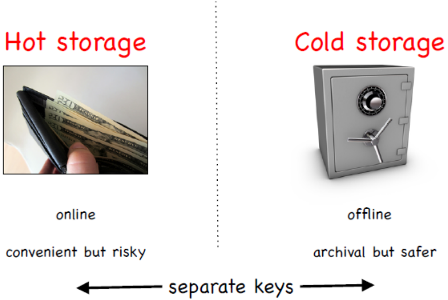
- Simple devices (not full computers) which generate keys but secret keys never leave device
- Only public keys/addresses/signatures allowed to escape device
- Examples: Trezor or Nano

Hardware Wallets (Hybrid hot/cold)

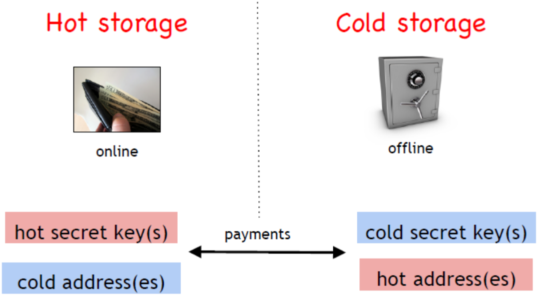
- Simple devices (not full computers) which generate keys but secret keys never leave device
- Only public keys/addresses/signatures allowed to escape device
- Examples: Trezor or Nano
- Availability: MEDIUM, Convenience: MEDIUM, Security: HIGH

How to Store Bitcoins Safely

- Useful principle: keep limited amount of bitcoins in hot storage and majority of the reserve in cold storage

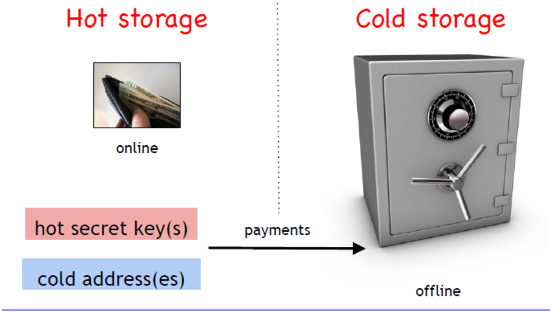


Transfers from hot storage to cold storage



- To move coins back and forth between the two sides, each side will need to know the other's addresses

Transfers from hot storage to cold storage



- Want to use a new address (and key) for each coin sent to cold storage. But, how can hot wallet learn new addresses if cold wallet is offline?

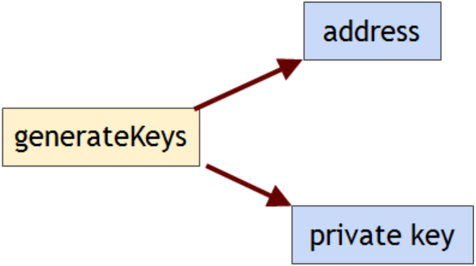
Straw-man solution

- Generate a big batch of addresses/keys, transfer addresses to hot beforehand
- This approach has the drawback that the two sides would need to communicate once the key pool is exhausted

Straw-man solution

- Generate a big batch of addresses/keys, transfer addresses to hot beforehand
- This approach has the drawback that the two sides would need to communicate once the key pool is exhausted
- Better solution: Hierarchical wallet

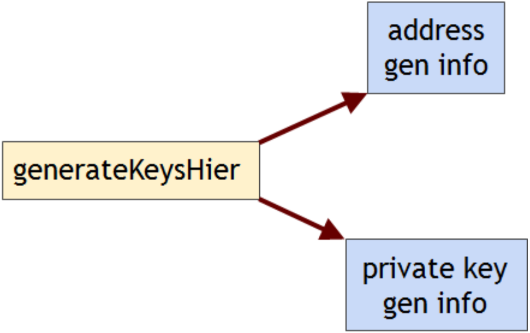
Recall: Regular key generation



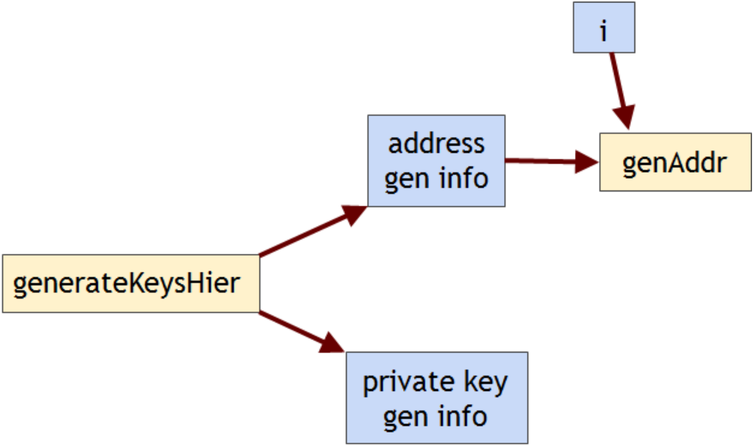
Hierarchical key generation

generateKeysHier

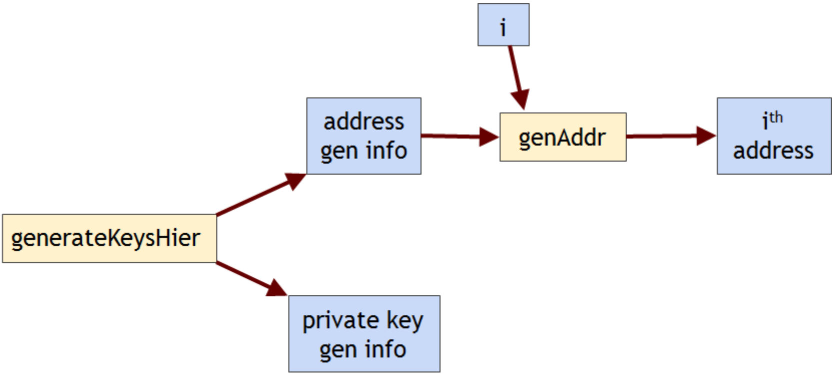
Hierarchical key generation



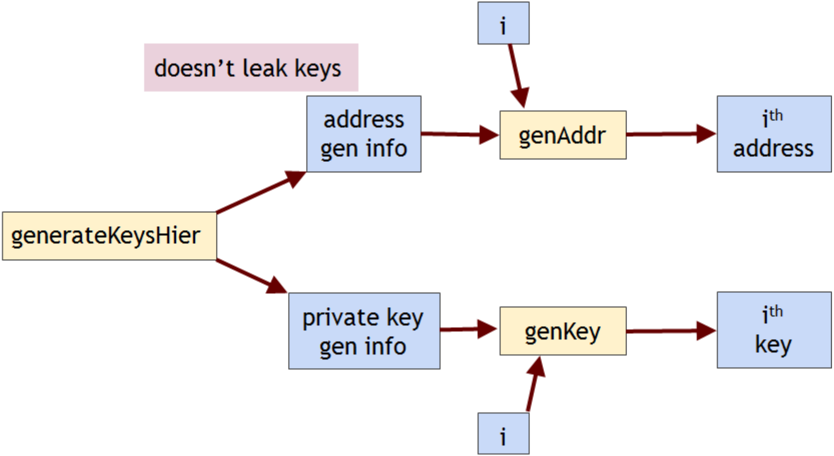
Hierarchical key generation



Hierarchical key generation



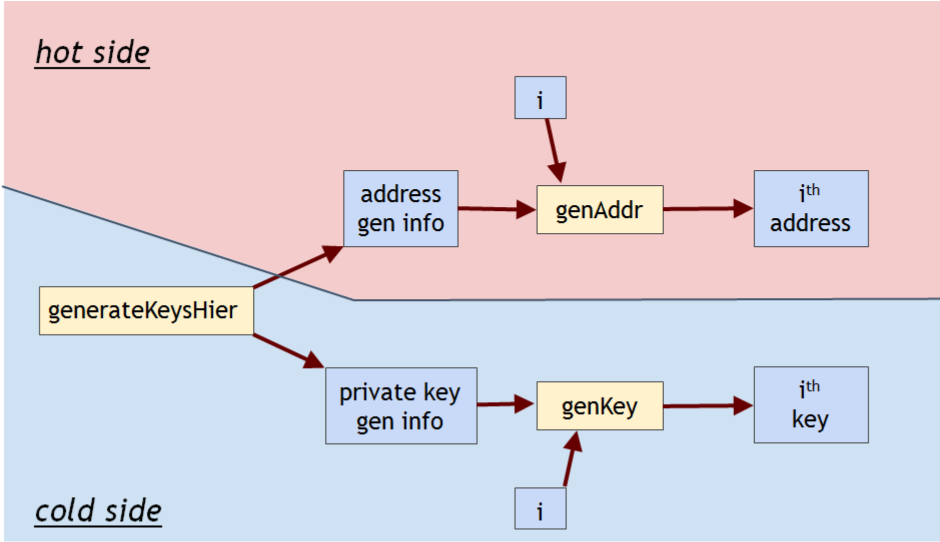
Hierarchical key generation



Properties

- Given an initial address generation info, there is a function that generates a sequence of public and private keys
  - For any integer i, the function generates the i-th address
  - and the i-private key in the sequence
  - Knowing the list of public keys does not reveal any private key

Hierarchical wallet



Implementation using ECDSA

- Let G denote the base point
- Setup

Implementation using ECDSA

- Let G denote the base point
- Setup
  - The master secret key e is stored on the cold wallet

Implementation using ECDSA

- Let G denote the base point
- Setup
  - The master secret key e is stored on the cold wallet
  - The corresponding master public key $P = eG$ is kept on the corresponding hot wallet

Implementation using ECDSA

- Let G denote the base point
- Setup
  - The master secret key e is stored on the cold wallet
  - The corresponding master public key $P = eG$ is kept on the corresponding hot wallet
  - The hot and the cold wallets both keep a common secret w

Implementation using ECDSA

- Let G denote the base point
- Setup
  - The master secret key e is stored on the cold wallet
  - The corresponding master public key $P = eG$ is kept on the corresponding hot wallet
  - The hot and the cold wallets both keep a common secret w
- ith public key:

$$P_i = P + H(w, i)G$$

Implementation using ECDSA

- Let G denote the base point
- Setup
  - The master secret key e is stored on the cold wallet
  - The corresponding master public key $P = eG$ is kept on the corresponding hot wallet
  - The hot and the cold wallets both keep a common secret w
- ith public key:

$$P_i = P + H(w, i)G$$

- ith secret key:

$$e_i = e + H(w, i)$$

Fractional Reserve Banking

- You give the bank money (a "deposit")
- Bank promises to pay you back later, on demand

Fractional Reserve Banking

- You give the bank money (a "deposit")
- Bank promises to pay you back later, on demand
- Bank doesn't actually keep your money in the back room

Fractional Reserve Banking

- You give the bank money (a "deposit")
- Bank promises to pay you back later, on demand
- Bank doesn't actually keep your money in the back room
  - typically, bank invests the money

Fractional Reserve Banking

- You give the bank money (a "deposit")
- Bank promises to pay you back later, on demand
- Bank doesn't actually keep your money in the back room
  - typically, bank invests the money
  - keeps some around to meet withdrawals ("fractional reserve")

Bitcoin Exchanges

- Online wallet plus place to buy/sell bitcoin (like a stock exchange)
- Help connect BTC economy to fiat currency economy
- Exchange knows private keys for your assets
- Just like real banks, exchanges promise to give you your bitcoin when you ask for it, but may not actually have it!

What happens when you buy BTC

- Suppose my account at Exchange holds \$5000 + 3 BTC
- I use Exchange to buy 2 BTC for \$580 each
- Result: my account holds \$3840 + 5 BTC

What happens when you buy BTC

- Suppose my account at Exchange holds $5000 + 3 BTC
- I use Exchange to buy 2 BTC for $580 each
- Result: my account holds $3840 + 5 BTC
- Note:
  - No BTC transaction appears on the blockchain!

What happens when you buy BTC

- Suppose my account at Exchange holds \$5000 + 3 BTC
- I use Exchange to buy 2 BTC for \$580 each
- Result: my account holds \$3840 + 5 BTC
- Note:
    - No BTC transaction appears on the blockchain!
    - Only effect: Exchange is making a different promise now

Risk 1: Bank/Exchange Runs

- Many people ask for their money back at a bank; bank does not have enough cash on hand

Risk 1: Bank/Exchange Runs

- Many people ask for their money back at a bank; bank does not have enough cash on hand
- Rumors spread that bank is insolvent

Risk 1: Bank/Exchange Runs

- Many people ask for their money back at a bank; bank does not have enough cash on hand
- Rumors spread that bank is insolvent
- More people try to cash out, exacerbating problem

Risk 2: Counterparty Risk

- Bank/exchange is not really planning on giving people the bitcoin that is rightfully theirs

Risk 2: Counterparty Risk

- Bank/exchange is not really planning on giving people the bitcoin that is rightfully theirs
- "Exit scam"
- Ponzi scheme: paying people who ask for bitcoin with newcomer's bitcoin

Risk 3: Security Breach

- Exchanges hold big amounts of coins and are a popular target for hackers
- A hacker who breaks into the exchange's software can move all of their bitcoin
- Many bitcoin thefts can be attributed to a security breach of exchanges
  - Mt. Gox roughly lost US$450M

Bank Regulation

- Traditional banks are required to comply with government regulations so that losses don't happen ...
- Government typically:
  - Imposes minimum reserve requirements
    - Must hold some fraction of deposits in reserve

Bank Regulation

- Traditional banks are required to comply with government regulations so that losses don't happen . . .
- Government typically:
    - Imposes minimum reserve requirements
        - Must hold some fraction of deposits in reserve
    - Regulates behavior, investments
        - Insures depositors against losses

Bank Regulation

- Traditional banks are required to comply with government regulations so that losses don't happen . . .
- Government typically:
  - Imposes minimum reserve requirements
    - Must hold some fraction of deposits in reserve
  - Regulates behavior, investments
    - Insures depositors against losses
  - Acts as lender of last resort

Bank Regulation

- Traditional banks are required to comply with government regulations so that losses don't happen . . .
- Government typically:
  - Imposes minimum reserve requirements
    - Must hold some fraction of deposits in reserve
  - Regulates behavior, investments
    - Insures depositors against losses
  - Acts as lender of last resort
- Bitcoin is not regulated like this!

What is Solvency?

- Solvency is the ability of an exchange to meet its financial commitments (i.e. assets $\geq$ obligations)
- Unlike conventional banks, bitcoin exchanges are expected to be fully solvent at all times

Proof of Solvency

- Proof-of-reserve
  - this proves that exchange holds at least X amount of bitcoins (assets)
- Proof-of-liabilities
  - this proves that exchange owes its customers no more that Y amount of bitcoins (obligations)
- Solvency is proved if X $\geq$ Y

Proof-of-Reserve Problem

- Goal: Prove how much reserve you're holding
    - Publish a valid payment-to-self of claimed amount
    - Sign challenge string with same private key

Proof-of-Reserve Problem

- Goal: Prove how much reserve you're holding
  - Publish a valid payment-to-self of claimed amount
  - Sign challenge string with same private key
- Proof of reserve $\neq$ willingness to pay

Proof of Liabilities

- Vanilla approach
  - Publish list of amounts and usernames of all accounts!

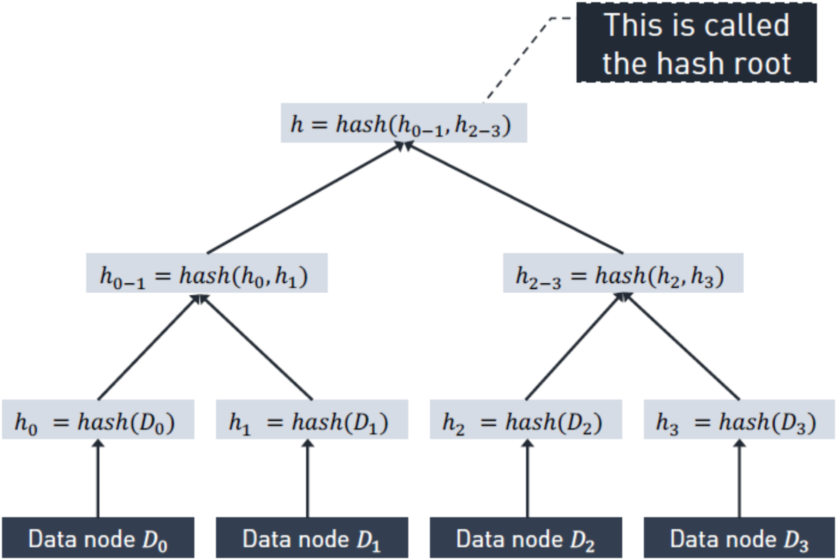
Proof of Liabilities

- Vanilla approach
  - Publish list of amounts and usernames of all accounts!
  - Users can complain if their accounts are missing or amounts are wrong

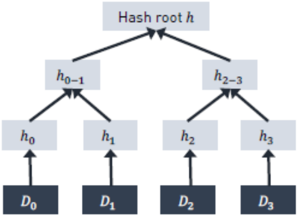
Proof of Liabilities

- Vanilla approach
  - Publish list of amounts and usernames of all accounts!
  - Users can complain if their accounts are missing or amounts are wrong
- Exchange may create fake users, but this only overstates liabilities

Merkle trees



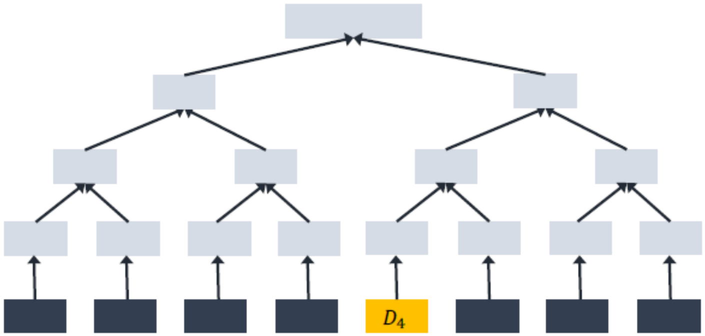
Uses of Merkle trees



Data integrity verification

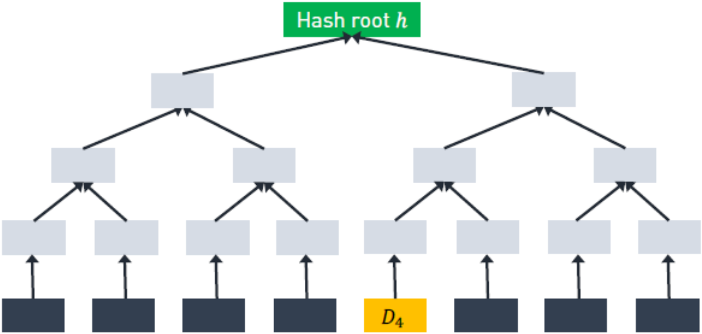
- The hash root is typically obtained from a trusted source
- One can verify the integrity of the data elements D_0, D_1, D_2, D_3 by reconstructing the hash root and comparing it to the trusted root

Membership verification



Verifying whether D_4 is a member (i.e., a leaf):

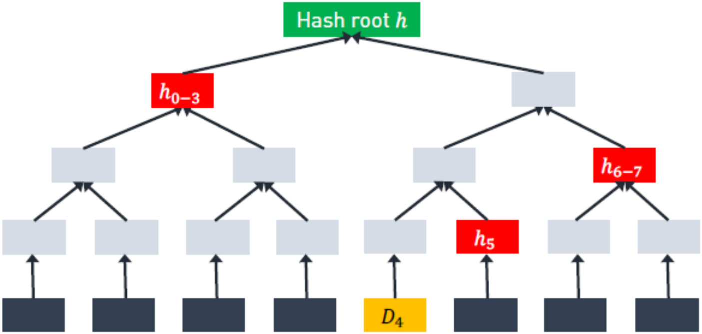
Membership verification



Verifying whether D_4 is a member (i.e., a leaf):

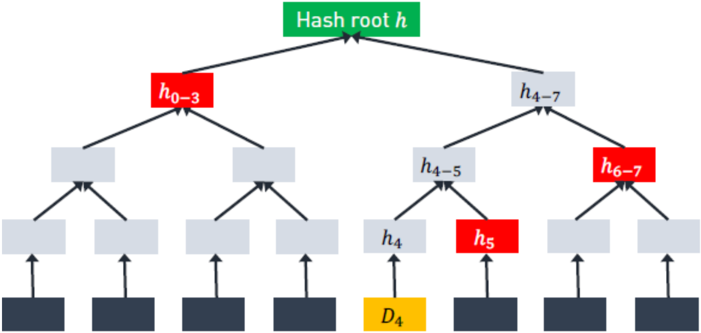
- Obtain the hash root h from a trusted source

Membership verification



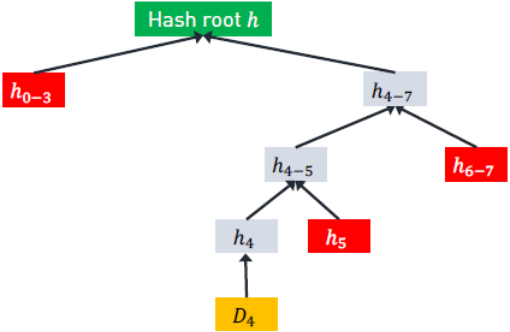
- Obtain the hash root h from a trusted source
- Request the nodes h_5, h_{6-7}, h_{0-3} (from possibly untrusted source)

Membership verification



- Obtain the hash root h from a trusted source
- Request the nodes h_5, h_{6-7}, h_{0-3} (from possibly untrusted source)
- Compute h_4, h_{4-5}, h_{4-7}, h' and check whether $h = h'$

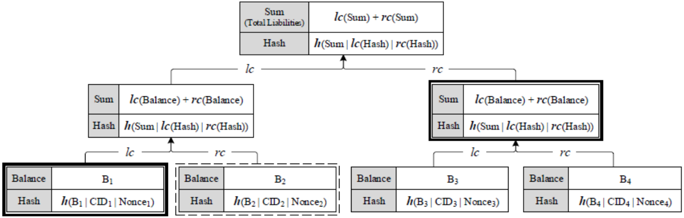
Membership verification



- Membership verification requires $\log n$ elements
- Useful when the set of data elements is large

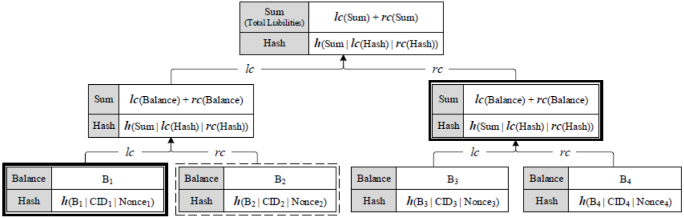
Maxwell protocol

- The exchange privately computes a modified Merkle tree that accumulates the account balances of depositors
- The tree below records four accounts:



Maxwell protocol

- The exchange privately computes a modified Merkle tree that accumulates the account balances of depositors
- The tree below records four accounts:



- The root node is publicly broadcast by exchange as its total liabilities

Maxwell protocol

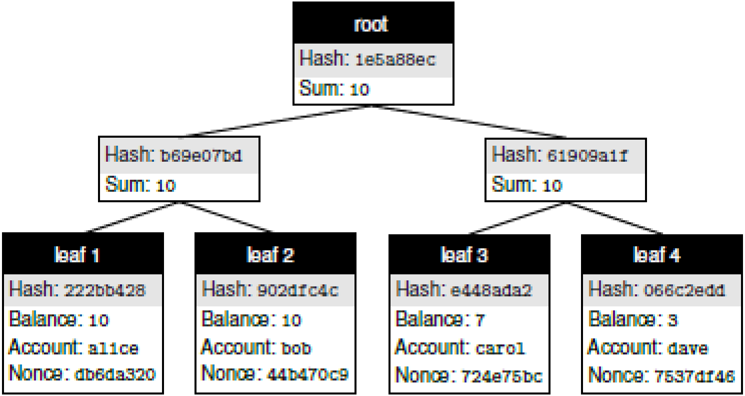
- Balance verification:
  - A customer requests the exchange to send a proof that her balance is included in the total liabilities
  - The exchange sends to the customer her nonce and the sibling node of each node on the unique path from the customer's leaf node to the root node
  - The customer accepts that their balance is included iff their path terminates with the same root broadcast by the exchange

Maxwell protocol

- Balance verification:
  - A customer requests the exchange to send a proof that her balance is included in the total liabilities
  - The exchange sends to the customer her nonce and the sibling node of each node on the unique path from the customer's leaf node to the root node
  - The customer accepts that their balance is included iff their path terminates with the same root broadcast by the exchange
- It is required that each customer performs the verification regularly

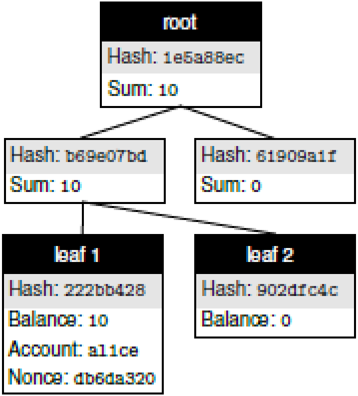
Flaw in Maxwell protocol

- Recorded by malicious exchange



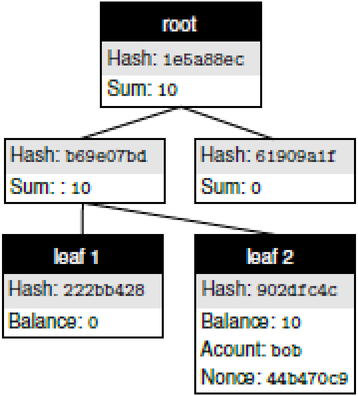
Flaw in Maxwell protocol

- The exchange gives Alice this tree:



Flaw in Maxwell protocol

- The exchange gives Bob this tree:



Attack result

- Both users are assured their balance is declared in the tree yet the exchange only needs to prove assets of 10 (instead of 20)

Privacy issues

- The value of exchange's total liabilities becomes public information

Privacy issues

- The value of exchange's total liabilities becomes public information
- Leaks partial information about other customers' balances
  - Each customer's proof reveals the exact balance of the sibling account in the tree

Privacy issues

- The value of exchange's total liabilities becomes public information
- Leaks partial information about other customers' balances
  - Each customer's proof reveals the exact balance of the sibling account in the tree
  - Each sibling node revealed in a given users' path to the root node reveals the total holdings of each customer in that neighboring subtree.

ZeroLedge

- ZeroLedge: Proving Solvency with Privacy
  - Jack Doerner, Abhi Shelat, David Evans [unpublished]
- no account can be fraudulently reduced or omitted
- Proof of solvency by inequality
  - proves only that the the total liability is less than some publicly stated value