6.857 Recitation 4: Bitcoin Mechanics

0. Administrivia

- PSET 2 due Monday, March 13

1. Introduction

Last week in recitation we saw a high level overview of bitcoin - we looked at the motivation behind a decentralized, electronic cash system and gave a preview of how the system works. In Monday's lecture, we went into more detail about the architecture of bitcoin and how the system was designed to achieve certain goals, such as distributed consensus.

Today, we will be reviewing some of the cryptographic primitives that are used in bitcoin and talking about some of the mechanics of bitcoin in greater detail. We'll start with a review of hash functions and the various ways they are used in bitcoin. We will then look at how transactions are made in bitcoin, making a distinction between an account-based/transaction-based ledger. We can also open up the discussion to general questions about bitcoin, such as what an adversary with 51% global hash power can do or how changes can be made to bitcoin.

2. Cryptographic Hash Functions

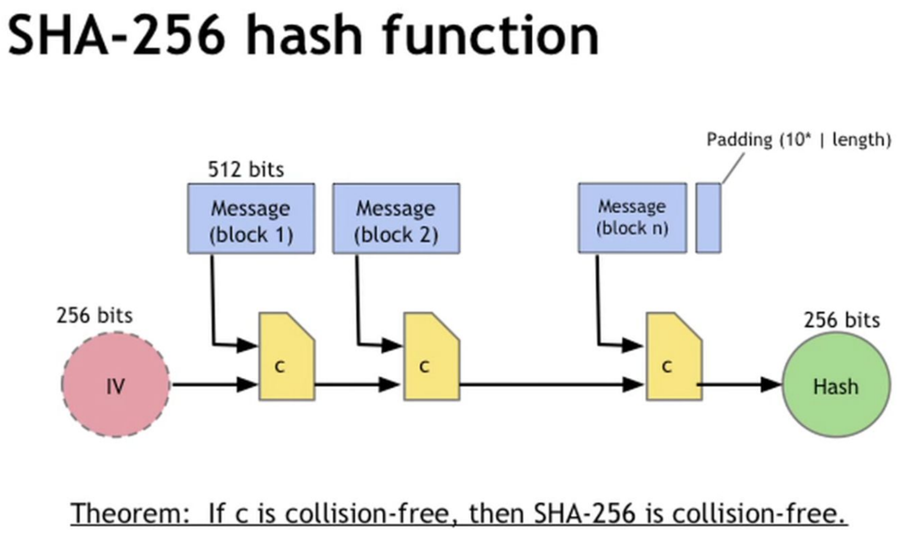
A cryptographic hash function:

- Takes any string as input
- Fixed size output (256 bits for bitcoin)
- Efficiently computable

Security properties:

- Collision free
    - Infeasible to find x != y such that H(x) = H(y)
    - Application: hash as a message digest
        - If we know H(x) = H(y), safe to assume x = y
        - To recognize a file we've seen before, just remember its hash
- Hiding / Preimage resistance
    - Given H(m), infeasible to find m
    - Note that even if hash function is preimage resistant, might be possible to find m
        - For example, if the only two possible m's were "heads" or "tails", adversary could just hash both and check if the hash matched H(m)
        - In practice, we can select r from a high min-entropy probability distribution, and use H(r || m) instead
- Puzzle friendly
    - For every possible output value y, if k is chosen from high min-entropy distribution, infeasible to find x such that H(k || x) = y
    - Important for proof of work (find nonce such that H(nonce || block header) < T)

SHA-256 (used in bitcoin)

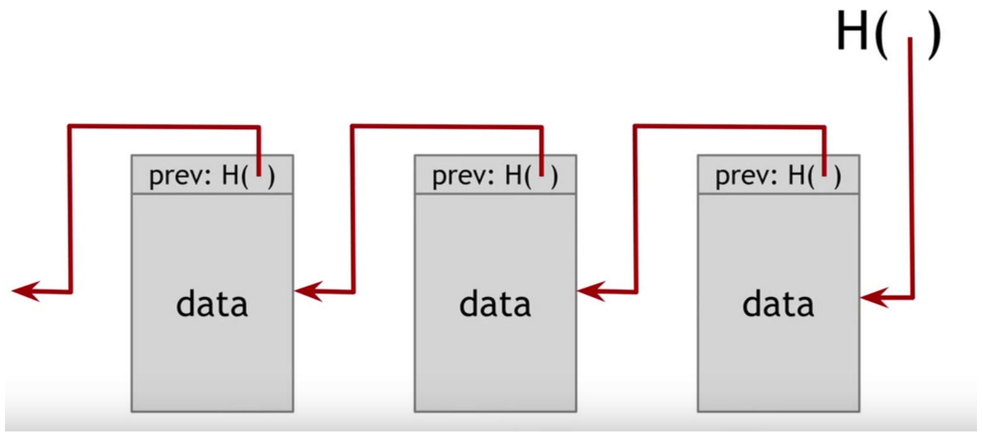


3. Hash Pointers and Data Structures

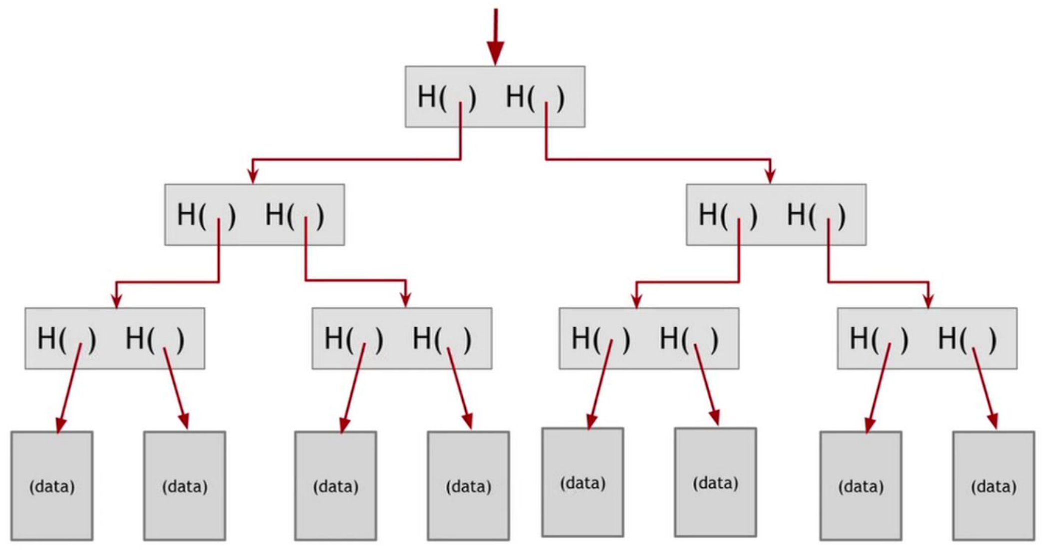
Hash pointer
- Pointer to where some info is stored
- Cryptographic hash of the info

Given a hash pointer, we can
- Ask to get the info back
- Verify that it hasn't changed

Using hash pointers, we can build data structures such as blockchains and merkle trees.



Blockchain



Merkle Tree

Blockchains

- Hash pointers prevent adversaries from tampering with block data

Merkle Trees

- Similar idea to blockchain, except in binary tree structure
- Can verify membership by showing just O(log n) blocks, rather than having to go through entire blockchain

4. Digital Signatures

Desired properties of signatures:

- Only you can sign, but anyone can verify
- Signature is tied to a particular document

API for digital signatures:

- (sk, pk) := generateKeys(keysize)
- sig := sign(sk, message)
- isValid := verify(pk, message, sig)

Requirements:

- Valid signatures verify (i.e. verify(pk, message, sign(sk, message)) == true)
- Adversary cannot forge signatures even if adversary:
    - Knows pk
    - Gets to see signatures on messages of his choice (except the one being forged of course)

Bitcoin uses Elliptic Curve DIgital Signature Algorithm (ECDSA) which uses complicated math
Signatures are used in bitcoin to sign transactions

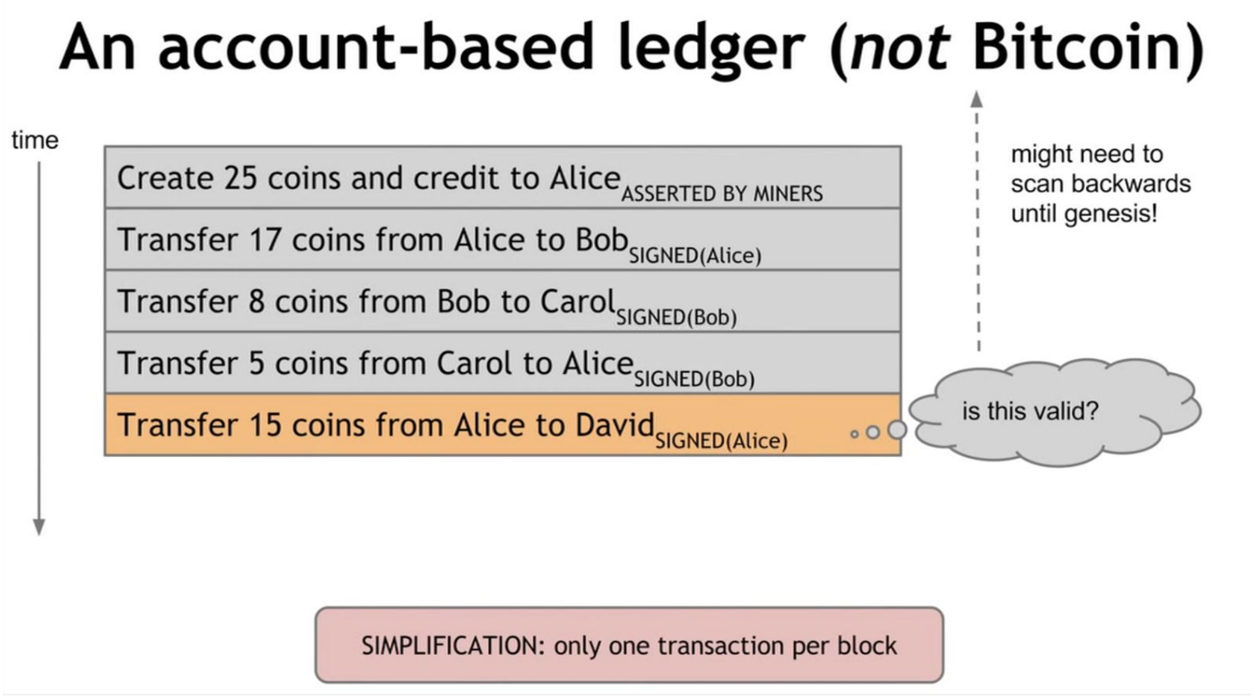
5. Public Keys as Identities

- Public keys can be thought of as identities
- verify(pk, msg, sig) == true means “pk says msg”
- In order to speak for pk, you must know sk
- Can make a new identity whenever you want
- No need for centralized identity management

6. Bitcoin Transactions

Account-based ledger (not how bitcoin works)

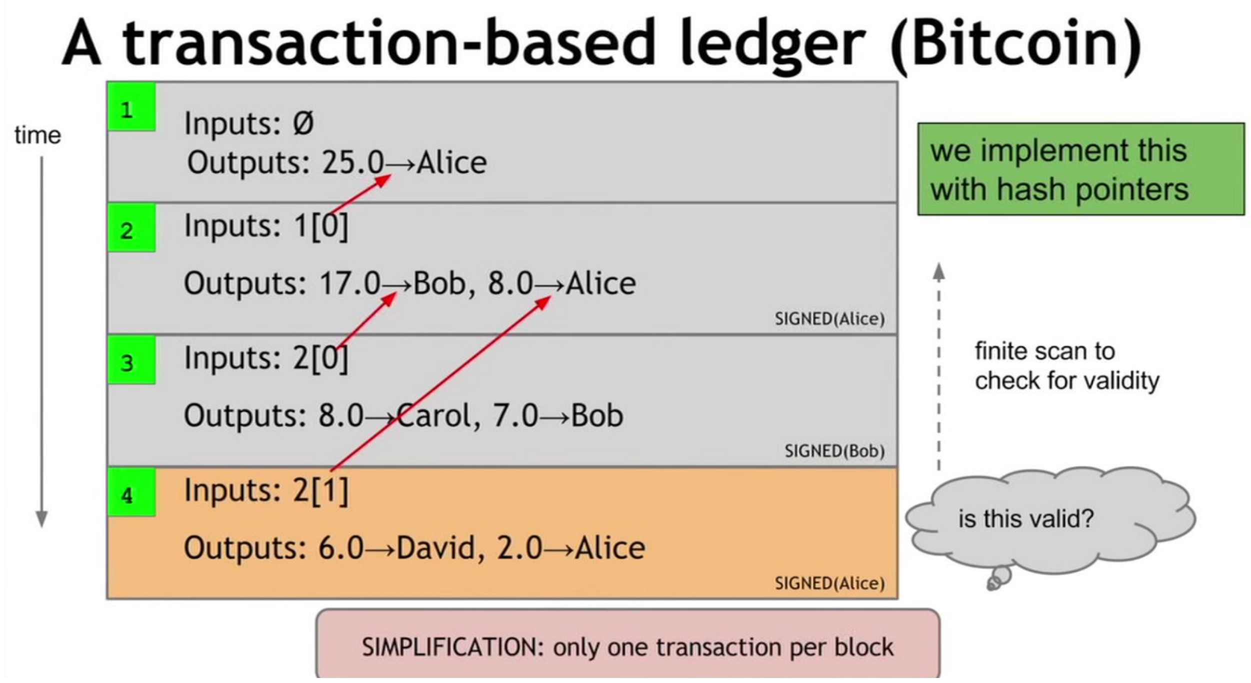
- Transactions to create or transfer coins
- Accounts have balances
- Problem: how do we know if a transaction is valid?
    - Would have to scan back all the way to the genesis block to see if Alice has enough coins for the transaction


An account-based ledger (*not* Bitcoin)
time
Create 25 coins and credit to Alice ASSERTED BY MINERS
Transfer 17 coins from Alice to Bob SIGNED(Alice)
Transfer 8 coins from Bob to Carol SIGNED(Bob)
Transfer 5 coins from Carol to Alice SIGNED(Bob)
Transfer 15 coins from Alice to David SIGNED(Alice)
might need to scan backwards until genesis!
is this valid?
SIMPLIFICATION: only one transaction per block

Transaction-based ledger (how bitcoin actually works)

- Each transaction has a unique identifier
- Transaction has inputs and outputs
    - Inputs - previous transaction where the coins came from
    - Outputs - recipients and amount

- All coins must be consumed in a transaction
    - If you are spending less than the input amount, then send yourself the change. Otherwise, it will go to the miner
- To check if a transaction valid:
    - Scan upwards to find input transaction that is referenced
    - Check that the coins from that transaction were not already spent in between


A transaction-based ledger (Bitcoin)
time
1
Inputs: Ø
Outputs: 25.0→Alice
2
Inputs: 1[0]
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)
3
Inputs: 2[0]
Outputs: 8.0→Carol, 7.0→Bob
SIGNED(Bob)
4
Inputs: 2[1]
Outputs: 6.0→David, 2.0→Alice
SIGNED(Alice)
we implement this with hash pointers
finite scan to check for validity
is this valid?
SIMPLIFICATION: only one transaction per block

Using this transaction-based ledger, it is also easy to:
- Merge values
- Make joint payments

To broadcast a transaction:
- A user submits a transaction to a node in the bitcoin P2P network
- Node tells all the other nodes about the transaction (flooding algorithm)
- Honest nodes will only include valid transactions in blocks
- (newly mined blocks are broadcasted throughout nodes in a similar fashion)

See https://blockchain.info/ to explore the blockchain and transactions