Cryptographic Protocols

Spring 2020

Blockchain Part 2

Repetition: A Virtual Bank (1/4)

Specification

- Bank keeps track of all balances.
- Users send transactions to bank.
- Bank executes (valid) transactions.

Alice	\$7'535
Bob	\$73'227
Clara	\$8'317
...	

`transfer(Alice,Clara,$200)`

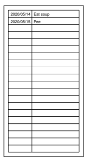
What is a Valid Transaction

- Debit account belongs to user.
- Amount of transaction is available.
- Amount is non-negative.
- Target account is valid.

Repetition: The Ledger Approach (2/4)

A Ledger

- Anybody can append an entry to the ledger.
- Anybody can read all entries on the ledger.
- Nobody can modify / delete entries on the ledger.



The Ledger Approach

1. Construct a ledger.
2. Construct a bank using the ledger.

Note 1: Privacy needs to be discussed (today).

Repetition: Constructing the Bank (3/4)

Our Bank

- ledger = initial balances, then sequence of all transactions
- transaction = (debit-pk, target-pk, amount, counter, signature)
- validity: a transaction is valid if
  - valid signature (w.r.t debit-pk)
  - amount available in debit-account
  - counter $>$ previous valid transactions with same debit-pk

Note: Target account (target-pk) is automatically created if not existing

Security Analysis

- correctness: ok if signature scheme is secure
- privacy: only pseudonymity
  $\rightarrow$ But: same customer can have multiple (many) accounts ...

Repetition: Constructing the Ledger (4/4)

Parties' Protocol (round-robin)

1. The king *broadcasts* block with unposted entries.

Users' Protocol (Post Entry)

1. Send the entry to some (presumably honest) parties.

Users' Protocol (Fetch Ledger)

1. Request block(s) from some (presumably honest) parties.
2. Majority decision (or decide according trust).
3. Check that each block is valid (including hash value).

Today: How to construct a permissionless ledger.

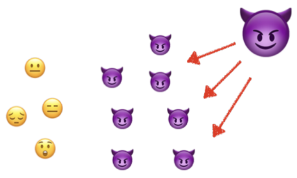
Permissioned $\longleftrightarrow$ Permissionless

Permissioned

- Users: Anyone can be a user.
- Parties: A fixed set of parties.

Permissionless

- Users: Anyone can be a user.
- Parties: Anyone can be a party.



Challenges of the Permissionless Setting

- unclear who participates
- the adversary can pretend to be many parties (Sybil attack)
  $\rightarrow$ no honest majority

And Now for Something Completely Different: Spam

The Problem

- Sending emails is cheap.
- Filtering spam is difficult (false positives).
- The receiver of spam has increased costs.

Cost-based Solution

- Incur a small cost for sending a single email.
- Normal user send small amounts of emails $\rightarrow$ small costs.
- Spammer send huge amounts of emails $\rightarrow$ huge costs.

Use Micropayments

- Sending an email costs, say 1 Rappen.
- This requires some sort of payment solution.
- No receiver actually wants to collect the payment.

Spam Spam Spam

Use Proof of Work (PoW)

- Computation cost is energy.
- Sender of email must solve a moderately-hard computation task.
- Verification of the computation is easy.

Partial Hash Inversion PoW

- Let H be a cryptographic hash function.
- Task: Given msg find nonce such that H(msg||nonce) starts with D zeroes.
- Solving task by guessing nonce (best strategy assuming H is a random oracle).
- D = k requires on average 2^k guesses
- Verification: one hash query

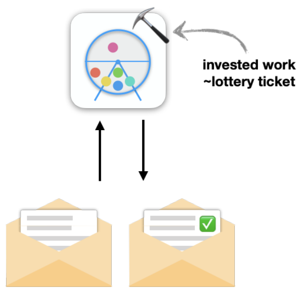
Proof of Work Lottery

Lottery

- Lottery tickets can be bought.
- Each ticket has a chance to win.
- Winner gets a prize.
- Verification of winner is easy.

PoW Lottery

- Cost ticket = one hash evaluation.
- A participant can buy many tickets.
- Winner can send email.



Idea: Use a proof-of-work lottery for a permissionless ledger.

Permissionless Setting

Parties and Users

- Users $U_1, U_2 \ldots$
- Parties $P_1, P_2, \ldots$
- Parties/users can join and leave.

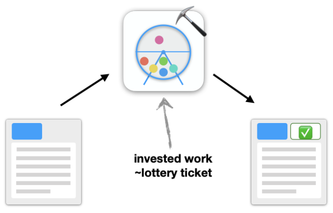
Communication

- Multicast channel
- Honest messages are received by everyone.
- Realised as a peer-to-peer network.

Constructing the Ledger: Block Lottery (1/2)

Idea

- Each party/user holds its copy of the ledger
- Lottery winner can publish a new block



Constructing the Ledger: Block Lottery (2/2)

Block consists of:

- The hash of the previous block
- Block data (the entries)
- The block nonce

Lottery

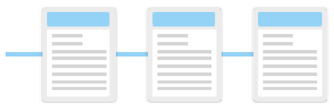
- Prepare hash and $data$ $\rightarrow$ defines what is appended where.
- Find nonce such that H(hash||data||nonce) as at least D zeroes.

Definition: A block is *valid*, if it
- is correctly formatted (syntax),
- contains only valid transactions
- contains the hash of the last valid block, and
- has a hash value with at least D zeroes.

Constructing the Ledger: Bitcoin Protocol (1/6)

Idea

- Lottery winner can publish a new block



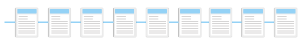
Constructing the Ledger: Bitcoin Protocol (2/6)

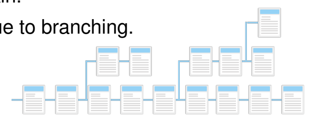
Bitcoin Protocol

1. Users multi-cast their entries, parties collect them.
2. Each party tries to extend their the longest chain by using the lottery.
3. Winner *multicasts* a block of (new) entries that extends longest chain.
4. Each party/user appends the new block if it is valid.

Observations

1. The lottery difficulty D and the total computing power determine the block production rate.
2. We have a blocktree, not a blockchain.
3. No instant confirmation of entries due to branching.





Constructing the Ledger: Bitcoin Protocol (3/6)

Setting the Difficulty

1. Low difficulty $\rightarrow$ fast block production.
2. High difficulty $\rightarrow$ less branching.
3. Ideally: The winner knows the block of the previous winner.
4. Bitcoin difficulty target:
   production rate of $\sim$ 1 block per 10 minutes.
5. Measure block production rate to estimate right difficulty.

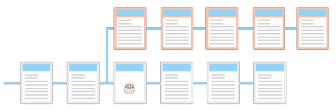
Confirmation of Entries

1. Idea: A block is confirmed if it is k blocks deep on the longest chain.
2. This is probabilistic consensus.
3. Rollbacks potentially happen.

Constructing the Ledger: Bitcoin Protocol (4/6)

Adversary

- Controls a fraction of the computing power
- Active



Example: Reorganisation Attack

1. The adversary publishes transaction to pay for Alice's cake.
2. As soon as the transaction is k blocks deep, Alice will accept it.
3. However, in the meantime the adversary created a branch:
   - which is k+2 blocks long
   - where the money for Alice is spent otherwise
4. The adversary publishes his branch.
5. As it is now the longest chain parties will accept it as the right version.
6. The payment for the cake has never happened.

Constructing the Ledger: Bitcoin Protocol (5/6)

Observation

- The attack requires the adversary to build a chain of k+2 blocks faster than the honest parties build one with k blocks.
- Computing power $\sim$ block production rate.
- $\Rightarrow$ to prevent this, the majority of computing power must be honest.

Informal Security Analysis

- Assume fast a network.
- Bitcoin is secure if 51% of the computing power is honest.
- For confirmation k must be set large enough (Bitcoin: $x = 6$).

Constructing the Ledger: Bitcoin Protocol (6/6)

Problems

- Specialized hardware skews the lottery.
- Centralisation due to electricity cost.
- Scaling: More parties = more energy consumption, but not more throughput.
- Bitcoin consumes more energy than Switzerland.

Constructing the Ledger: Proof of Stake (1/2)

Idea

- Money is a limited resource.
- Select block producer proportional to their wealth (=stake).

Approach I: Bitcoin-like

- Proof of Stake lottery
- Lottery winner proposes block

Approach II: Byzantine Fault Tolerance (BFT)

- Similar to our permissioned ledger protocol
- King replaced by winner of lottery
- Broadcast simulated by committee
- Committee also selected by lottery

Constructing the Ledger: Proof of Stake (2/2)

PoS Lottery

- Lottery tickets proportional to stake
- **Problem:** How to generate randomness for drawing?

Randomness Source

- Use an MPC protocol (expensive)
- Use a Verifiable Random Function (VRF):
  - A pseudo random function
  - Output can be computed locally
  - Output can be verified publicly

Actual Ledger Protocols

Name	Consensus	Security	
Bitcoin	PoW	$t < n/2$	
Ethereum	PoW	$t < n/2$	
Cardano	PoS	$t < n/2$	Bitcoin-like
Algorand	PoS	$t < n/3$	BFT

Constructing the Bank: Minting (1/2)

Until now:

- Total balance of all accounts is fixed.
- (Honest) parties are altruistic.

Bitcoin Approach to Minting

- Creator of each block gets a block reward.
- Block reward comes out of nothing $\rightarrow$ money printing.
- However, total number of coins is limited to 21 million.
  $\rightarrow$ Block reward gets smaller over time (until it is zero).

Constructing the Bank: Minting (2/2)

Minting in General

- Get (fresh) money for maintaining the ledger.
- Supply of money can be capped or unlimited:
  - Determines the inflation rate.

Initial Balances

- Proof of Work: Can start with no money at all.
- Proof of Stake: Needs initial stake distribution.
  - Alternatively, start with PoW and change to PoS later.

Constructing the Bank: Transaction Fees

Problem

- Users can spam transactions $\rightarrow$ system gets overwhelmed.

Solution

- Publishing a transaction costs a transaction fee.
- Fee is deducted from sender's account.
- Bitcoin: Transaction fee goes to block creator.
  $\rightarrow$ Transactions with high fee get priority.
- Bitcoin: Transaction fee will eventually replace block reward.

Constructing the Bank: Privacy (1/2)

Problem

- The ledger is public.
- Transactions reveal: amount, involved accounts.

Hiding Balances (Sketch)

- Accounts balances are encrypted (using public-key scheme).
- Transactions:
  - Amount encrypted under public-key of recipient.
  - Contains updated (encrypted) balance for sender's account.
  - Contains zero-knowledge proof for validity check.

Constructing the Bank: Privacy (2/2)

Anonymous transactions

- Idea: Hide sender and recipient of a transaction.
- Implementation: Heavy use of zero knowledge protocols.

Real World Privacy

- Privacy Coins exist, e.g. Monero or ZCash.
- Level of privacy unclear.
  - Bug in Monero allowed to link transactions.
  - ZCash privacy is opt-in.
- Used by criminals, e.g. for ransomware.

Constructing the Bank: Extended Functionality (1/2)

Goal

- Conditional transactions, e.g. Alice pays Bob iff Charlie pays Alice.
- "Automatic" contracts, e.g. an insurance policy.

Smart Contracts

- Contract expressed and executed as a program.
- Everyone agrees exactly on the interpretation.
  $\rightarrow$ Code is law!

Smart Contracts on the Ledger

- Contracts can be added to ledger.
  $\rightarrow$ initial state
- Contracts calls are added to ledger (same as transactions)
  $\rightarrow$ Calls (and their order) define current state of contract.

Constructing the Bank: Extended Functionality (2/2)

Applications

- Investment funds
- Insurance
  - Requires access to outside data $\rightarrow$ needs an oracle.
- Games and gambling

Problems

- Contracts are public.
- Bugs allow for unintended functionalities.
- Contracts are immutable (forever!).