# Cryptographic Protocols

Spring 2020

Blockchain Part 1

## A Virtual Bank



**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**

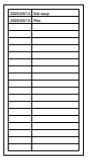
- debit account belongs to user
- amount of transaction is available
- amount is non-negative
- target account is valid

## The Ledger Approach



**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 (later)

**Note 2:** some ledgers have a fixed initial state (e.g. first $k$ entries)

## Constructing the Bank: First Attempt



**First Attempt:** Transactions on Ledger

- ledger is initialized with initial balances (starting point)
- customers post transactions on ledger
- derive balances from initial balances and sequence of transactions
- transaction = (debit-account, target-account, amount)

**Security Analysis**

- correctness: no authorization
- privacy: none (resp. pseudonymity, can track accounts)

## Constructing the Bank: Sign Transactions



**Improvement 1:** Sign Transactions

- posted transactions must be signed by account-holder
- mapping account-holder $\leftrightarrow$ public key?
- trick: account number **is** the public key
- transaction = (debit-public-key, target-public-key, amount, signature)

(with debit-secret-key)

**Security Analysis**

- correctness: replay attacks (adversary can duplicate transaction)
- privacy: pseudonymity (transactions can be *linked*)

## Constructing the Bank: Add Nonce to each Transaction



**Improvement 2:** Add Nonce to each Transaction

- transaction = (debit-pk, target-pk, amount, nonce, signature)
- each transaction (bit-string) is only valid once

**Security Analysis**

- correctness: ok if signature scheme is secure
- privacy: pseudonymity

**But:** validity of transaction is hard to check:
need to compare with all previous nonces

**Solution:** use strictly increasing counter as nonce (e.g. time-stamp)

## Constructing the Bank: Our Bank



**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 ...

## Constructing the Ledger: The TTP Approach



**Remember:** ledger = sequence of entries

**The TTP Approach**

1. each user sends his entries to TTP
2. TTP appends received entries to ledger
3. TTP sends new ledger to each user
4. goto 1.

**Caveats and Improvements**

- TTP needed
- whole ledger is sent in each iteration

## Constructing the Ledger: Chain of Blocks

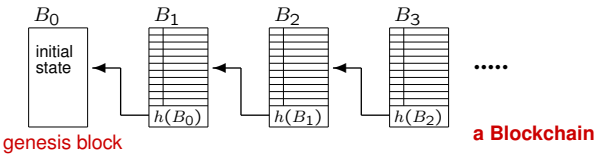


**Improvement 1:** Send only new Entries

1. each user sends his entries to TTP
2. TTP combines entries to a *block*, and sends the block to each user
3. each user appends block to his copy of the ledger
4. goto 1.

**Problem:** New users must *validate* old blocks (fetch from anywhere)

**Solution:** Add hash of previous block to new block




## Constructing the Ledger: Using MPC



**Improvement 2:** Distribute Trust with MPC (e.g. [BGW88])

**Notation**

- parties *simulate* the ledger
- users *use* (post to / read from) the ledger

**Security Analysis**

- secure if $t < n/3$

**Caveats and Improvements**

- number of parties and users needs to be small and fixed
- communication needs to be synchronous

## Constructing the Ledger: Distributing Trust (1/2)



**Improvement 3:** Distribute Trust Without MPC

- each party holds its own *copy* of the ledger
- parties append *same blocks* in the *same order*

**Protocol**

1. every user sends his entries to all parties
2. some party is chosen to be the *king* (e.g. round-robin)
3. the king *broadcasts* a block of (new) entries to all parties
4. each party stores the received block and forwards it to all users
5. each user stores the block *received most often*
6. goto 1.

**Security Analysis**

- liveness (entries make it onto the ledger): at least one honest party
- consistency: broadcast is correct, plus $t < n/2$ (for Step 5)

## Constructing the Ledger: Distributing Trust (2/2)



**Validity of Blocks**

- a malicious king can broadcast an invalid block
  (wrong format, wrong hash, etc.)
- such a block must be ignored
- honest users / parties only append valid blocks to their ledger

**Further Optimization**

- no need to store invalid transactions in ledger

**Definition:** A block is *valid*, if it
- is correctly formatted (syntax),
- contains only valid transactions, and
- contains the hash of the last valid block

## Constructing the Ledger: Decoupling Parties and Users (1/2)



**Improvement 4:** Decoupling Users

- users send entries to *one or multiple* parties *at any time*
- each party maintains a *local pool of unposted entries*
- users *request* latest block(s) from *one or multiple* parties *at any time*

**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)

## Constructing the Ledger: Decoupling Parties and Users (2/2)



**Security Analysis**

- liveness (entries make it onto the ledger)
  - entry needs to be sent to at least one honest party
  - honest parties needs to be the king sometime
- consistency among parties
  - broadcast protocol (e.g., $t < n/3$)
- consistency among users
  - majority of requested parties must be honest

## Outlook (Next Week)



**The Ledger**

- so far: permissioned (only allowed parties can participate)
- new: permissionless (anybody can participate – party set unknown)

Challenges

- broadcast not working
- round-robin not working
- no honest majority of parties (sybil attack)

**The Bank**

- so far: only transactions
- new: complex actions

Challenges

- conditioned transactions
- „smart contracts“