Crypto Protocols - Blockchain

Lecture Notes

Introduction

Alice owes Bob 100 swiss francs. Normally, Alice could simply go to Bob and give him a 100 bill. But, due to COVID-19 Bob is reluctant to accept a physical bank note. This is not an issue since both of them are customers at the same bank. So Alice simply instructs the bank to transfer the money to Bob's account. Later, Bob can verify with the bank that he indeed received the money.

A bit more formally, here is the ideal specification of this bank.

1. The bank keeps track of all *account balances*.
2. Users can see their balance and instruct the bank to *transfer money* from their account.
3. The bank executes valid transactions.

A transaction consists of a debit account, a target account, and an amount. If it is executed the amount is deducted from the debit account and added to the target account. A transaction issued by a user is considered *valid* if the following conditions are met:

- The debit account belongs to the user.
- The amount is non-negative.
- The debit account contains at least the amount.
- The target account is a valid account.

The Ledger Approach

In the following we discuss the ledger approach to construct a virtual bank. This approach is commonly used in cryptocurrencies.

A *ledger*[1] is an ordered list of entries with the following properties

- Anyone can append an entry to the ledger
- Anyone can read all entries on the ledger
- Nobody can modify or delete entries on the ledger

A famous ledger is the land register which lists the owner of properties. Here, entries are changes in ownership and the ledger defines the current state of ownership.
Some ledgers also have a special initial state. For the land register this means that it contains the initial state of ownership.

[1] German: Register

The plan to construct a bank is as follows. First, construct a ledger using a distributed protocol. Then construct the bank by using the ledger to store transactions.

Constructing the Bank

We start top down and show how to construct the bank given a ledger. The rough idea is that parties can post their transactions on the ledger which defines the state of the bank.

Accounts and Transactions

As anyone can post on the ledger, we need to ensure that only the owner of an account can send its money. A simple solution is to assume that each account has an associated signature public key where the account owner knows the corresponding secret key. This allows the owner to authorize transactions by signing them with the account key. Moreover, we can use the public key as an account number, i.e. as the identifier of the account. In summary, an account has the form:

acc = (public key, amount)

As seen in the introduction, a transaction must contain a debit account, a target account, and an amount. For the authorization, we additionally require that it contains the signature of the account holder. However, this is not enough. A transaction also needs to contain a nonce to prevent that the transaction is executed twice.

tx = (debit pk, target pk, amount, nonce, signature)

A transaction is valid if:

- The amount is positive and the debit account balance is greater than the amount.
- The target is a valid account, that is target pk is a valid public key.
- The transaction (debit pk, target pk, amount, nonce, signature) (as bit-string) was not executed before.
- The signature is a valid signature of (debit pk, target pk, amount, nonce) under the key of the debit account.

Note that the target account validity conditions ensures that a target account is created automatically if it did not exist before[2].

[2] The idea is that users can generate a fresh signature key pair and then use the public-key as a new account identifier. However, it is perfectly valid to send money to an account where nobody knows the secret key.

Bank Protocol

We assume that the ledger contains a list of initial accounts with initial balances. The bank protocol works as follows:

- Users can post transactions on the ledger.
- The sequence of valid transactions defines the current set of accounts and their balances.

Security Analysis

Consistency: The ordered list of transactions defines the (unique) list of valid transactions. Thus all users agree on the state of the bank.
Correctness: A transaction is only valid if it was signed by the account owner. This prevents other users from stealing the account's money. The nonce prevents replay attacks.
Privacy: We have no privacy as all information on the ledger is public. However, there is at least some sort of pseudonymity as the accounts are just identified by public-keys and users can have multiple accounts.

Nonce or Counter

Using a nonce to prevent replay attacks means that a user needs to look at the whole list of transactions to determine if that transaction was already included on the ledger. This can be optimized by using a counter instead. The idea is that each account has a corresponding counter. Each valid transaction from that account increases the counter and transactions contain the current counter value. To check validity, users just check that the last valid transaction used a strictly smaller counter.

Constructing the Ledger

The TTP Approach

We first discuss the ledger construction in the presence of a trusted party. The protocol is simple. Users send their entries to the trusted party which appends it to its ledger. Each time the ledger is extended the trusted party sends out the new ledger to all users.

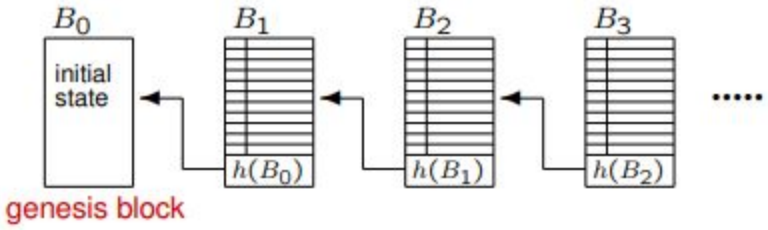
Blockchain, Blockchain, Blockchain

The above TTP protocol has the disadvantage that the trusted party sends out the whole ledger each time a new entry is added. We can improve the protocol by requiring the trusted party only to send out the new part of the ledger.

For this it makes sense to use a *blockchain* data structure for the ledger. A blockchain is a sequence of *blocks*. Each block contains data and the hash of the previous block (if it exists). The first block in the sequence is also known as genesis block.

The improved TTP protocol works as follows:

- Each user has a copy of the blockchain (which initially is just the genesis block).
- Users send their entries to the trusted party which collects them.
- The trusted party combines the entries into a block and sends the block to all users.
- Users append the new block to their copy of the blockchain. Users accept the new block if the contained hash is the hash of the last block in their current blockchain.



Permissioned Ledger

In this section, we discuss the ledger construction in the *permissioned* setting. We have worked in this setting when constructing MPC and broadcast protocols. More formally, we have

- A fixed set P_1,..,P_n of *parties* that simulate the ledger.
- *Users* that use the ledger.
- An *active adversary* that corrupts at most t parties and any number of users.

On Using MPC

In principle one could use an MPC protocol (e.g. [BGW88]) to simulate the trusted party which maintains the ledger. However, there are some caveats to this approach. First, the number of parties and users needs to be fixed and fairly small. Second, the MPC protocols we have seen so far require synchronous communication.

Ledger Protocol

It turns out that we can simulate the ledger using just broadcast. The idea is that users send their entries to the parties which then agree on a new block. We assume that each user/party holds a local copy of the blockchain. The protocol proceeds as follows.

- Users send entries to multiple parties at any time.

- Each party maintains a local pool of unposted transactions.
- Periodically, a party is chosen to be the king (e.g. round-robin). The king broadcasts a block of new (and valid) transactions to all parties. If the block is valid the parties append the block to their blockchain. A block is considered valid if it is (a) correctly formatted (syntax), (b) contains only valid transactions, and (c) contains the hash of the last valid block.
- A user asks all parties for the new block, the users stores the block received most often.

Security Analysis. The protocol satisfies the following properties for t<n/3.

- **Liveness:** (Valid entries from honest users make it onto the ledger).
  If an honest user sends its entry to an honest party, that party (once it is king) will integrate the entry in the ledger (if it has not yet been integrated), by validity of broadcast. So if (a) honest parties are elected king sometime (which is the case for round-robin) and (b) honest users send their entries to at least one honest party (which is the case if they send it to all parties) liveness follows.
- **Consistency (for parties):** Consistency follows directly from the broadcast protocol (which is consistent for t<n/3).
- **Consistency (for users):** The majority of parties are honest. From those parties the user will get the same block. Consistency follows.

Permissionless Ledger

Next, we go to the *permissionless* setting. The main difference to the permissioned setting is that anyone can be a party. We have

- Arbitrary many *parties* that simulate the ledger.
- *Users* that use the ledger.
- An *active adversary* that corrupts any number of parties and users.

Communication

We assume that parties and users communicate via *multicast channels*. These channels ensure that messages from honest parties/users are received by everyone (but does not guarantee consistency). For simplicity, we assume that messages are delivered fast (e.g. within seconds).

Proof-of-Work Ledger

In the permissionless setting the adversary can pretend to be any number of parties (Sybil attack). This makes the ledger protocol from above unsuitable as it requires ⅔ of the parties to be honest. We therefore need to base our security on a different assumption. One option is to

assume that *a majority of computing power is controlled by honest parties.* This allows us to use a *proof of work lottery* to build a ledger where the lottery winner extends the blockchain.

Block Lottery using Proof of Work

The idea of a proof of work lottery is that a participant needs to do some work to get a chance to win and create a new block. Let H be a cryptographic hash function. We also add a *nonce* to each block of our blockchain. The lottery works as follows:

- A party prepares a block. The data of the blocks consists of the transactions and the hash value of the previous block.
- The party then tries to find a nonce such that the hash H(data||nonce) has at least D zeroes.
- A block is valid if it is correctly formatted, contains valid data, contains the hash of a valid block, and has a hash value with at least D zeroes.

As H is a cryptographic hash function it acts like a random oracle. It is therefore the best strategy for a party to simply guess the nonce. The *difficulty* parameter determines how many guesses on average the party needs to make (difficulty D requires on average 2^D guesses). Each guess costs the party one hash query. Note that the verification of a valid block is easy and takes just one hash query.

Remark. Proof-of-work was originally developed to combat spam. The idea is that the sender of an email needs to do some (computational) work in order to send an email. While this is fine for normal users, it would result in great costs to spammers.

Bitcoin Protocol

The Bitcoin protocol builds a ledger using a proof-of-work lottery as seen above. Note that in this lottery it can happen that two parties win at roughly the same time. This means that we no longer have a blockchain, but rather a blocktree with the genesis block as root. The longest chain (i.e. the longest branch) represents the ledger.

The protocol works as follows

- Every user/party maintains a local copy of the blocktree.
- Users multicast entries and parties collect them in a local pool of unposted transactions.

- Each party tries to extend the longest blockchain by using the proof-of-work lottery. A winning party multicasts a new block (that extends the longest chain).
- Every user/party appends valid blocks to the blocktree.

The lottery difficulty D and the overall computing power determine the block production rate. While a fast block production is desired, it also means higher probability of two winners at the same time and thus unwanted branching. In the actual Bitcoin protocol the difficulty is therefore set such that on average 1 block per 10 minutes is produced.

Entries on the longest chain can no longer be seen as instantaneously confirmed as branching and thus rollbacks might occur. The probability that a block remains (for ever) a part of the longest chain, increases (exponentially) with its depth (i.e. the number of blocks added after it). Parties therefore need to wait until an entry is deep enough (e.g. k blocks) on the longest chain to consider it as confirmed. This makes Bitcoin a *probabilistic consensus protocol.*

Miners. In Bitcoin the parties are also known as miners as they have to do work to earn new Bitcoins.

Informal Security Analysis. We assume for simplicity that the network conditions are nice. In particular, we require that any valid block seen by an honest party, is seen by every other honest party within seconds (compared to the block production rate of 1 block per 10 minutes). This implies that honest parties have a more or less consistent view on the blocktree. It also means that honest parties will all extend the same longest chain. So without adversarial influence, the honest parties will essentially grow a chain (where branching is rare).

A potential attack the adversary can launch is the so-called reorganisation attack. Here, the adversary publishes a transaction on the chain. This transaction will be confirmed as soon as it is k blocks deep on the chain. However, in the meantime the adversary was growing a branch which is k+2 blocks long and where the money used in the transaction was spent otherwise. So as soon as the transaction got confirmed the adversary publishes the side branch. As it is now the longest chain, all honest parties will switch the new branch. Thus the original transaction never happened. To launch such an attack the adversary needs to be able to produce more blocks as all honest parties together. While this might work out of sheer luck for small k, it would require a majority in computing power for large k.

One can show (even in less ideal network conditions) that Bitcoin is secure as long as the honest parties have a strict majority of computing power.

Energy Consumption. Bitcoin is a proof-of-work protocol. Its security relies on parties using computing power. More parties joining the protocol means more computing power used to compute hash values without getting more performance. At the moment, Bitcoin requires more electricity than Switzerland. This makes Bitcoin not really a green technology.

Proof of Stake Ledger

An more environmentally friendly alternative to proof-of-work protocols are proof-of-stake protocols. Instead of using computation, money (=stake) is used as a limited resource. In particular for our bank ledger one can directly use the money on the ledger as a resource.

Proof of Stake Lottery

In a proof of stake lottery, every coin corresponds to one lottery ticket. So a party with 10 coins has twice the chance to win the lottery compared to a party with 5 coins. We will not go into details on how to construct such a lottery. However, the main challenge in doing so is actually getting the randomness (which is provided by the hash function in proof of work). The randomness problem can be solved using what is called a verifiable random function (VRF) which is essentially a PRF which also produces a proof for the outcome.

Proof of Stake Ledger

There are different approaches on how to build a ledger using proof of stake. One way is to simply take the Bitcoin protocol and replace the proof-of-work lottery by a proof-of-stake lottery.
Another option is the Byzantine fault tolerance (BFT) approach. It works similar to our permissioned ledger protocol. The main differences are (a) the king is replaced by a proof-of-stake lottery winner and (b) the broadcast is simulated by a committee which is also elected by means of a lottery.

Constructing the Bank - Part II

Minting

Until now we assume that the total balance in our bank is fixed (defined by the initial balance in the genesis block). We also assumed that honest parties act altruistically, even though maintaining the ledger can be costly.

To incentive participation in the ledger protocol, one can provide a *block reward* to the creator of a block. This reward comes out of nothing, i.e. is freshly printed money. The supply of fresh money can be capped as in Bitcoin or unlimited (which results in inflation). In Bitcoin the block reward gets smaller over time until it becomes zero when a number of Bitcoins are in circulation.

Initial Balances

In a proof-of-work ledger one can start with 0 total balance; the first money will be added with the first block to the system. A proof of stake ledger inherently needs an initial stake distribution as otherwise the lottery would not work.

Transaction Fees

Until now we assumed that users can issue any amount of transactions. However, this poses the risk that our bank system gets overwhelmed. The solution is simple: Publishing a transaction costs a *transaction fee.* The fee is deduced from the sender's account.

In Bitcoin the transaction fee goes directly to the creator of the block which contains the transaction. This incentives parties to include the transactions with the highest fees.