1 Lecture 17: Blockchain

Starting from the basics, a Blockchain is an append-only public transaction ledger.

Each identity on the blockchain is a public key for some signature scheme and is associated with some amount of initial holdings. The corresponding secret key is called the "wallet key".

Each time a transaction is made, we make a document D that states "A paid B some x amount of money". Then, this statement is signed by A (whoever paid), and we can verify their signature using the public key (which is their own identity on the blockchain). We also need to check that the transaction is valid in the sense that A had at least x dollars to spend, but we can simply do this by tracking everything A has done, and calculating how much they had based on their initial holdings and all previous transactions.

But in practice, we actually use a state machine. Each time we submit a transaction, the state machine checks if it is valid. If so, it adds the transaction and any corresponding balance changes to the state. Otherwise, it rejects the bad transaction.

Overall, the state machine keeps track of balances and transactions, and when we "get state", we get back a "state" which shows everyone's balances.

But we don't want just one person maintaining this ledger, since they will then have total control, so we go completely in the opposite direction and have multiple users collectively maintain the ledger. In order to motivate people to cooperate, we give a portion of each transaction to people who help maintain the ledger.

So, a **blockchain** is a **decentralized**, **append-only public transaction ledger**. It's called a blockchain because it's essentially a linked list of **blocks**. Each block is some bounded set of transactions. The original entry of the linked list is the set of identities and associated initial holdings, and each block points to the previous block (with the first block pointing to the original entry). The link from block B to block C is achieved by hashing the ID of block C together with the contents of block B, and keeping this as part of block B.

1.1 Adding Blocks

We run into another small issue here, similar to the issue of maintaining the state machine above. In particular, who gets to decide which blocks get added to the blockchain?

Our first possible solution is called **permissioned blockchain**. We choose a set of trusted servers who will use an MPC to determine whether blocks are valid. Depending on the security model, we only need to guarantee that some threshhold fraction of the servers are not corrupt (possible values noted in lecture were $n/2$ and $n-1$).

Another solution is called **permissionless blockchain**, where any user of the blockchain can act as a server and approve transactions. The challenge here is that we can no longer assume that we have an honest majority to our servers and that they use MPC, since an adversary could create an arbitrary amount of malicious "users" to create a dishonest

majority. So we need to deal with this. We use mechanisms such as "proof of work" and "proof of stake" to enforce users to prove that they are a real user/computer and not one of many generated by some adversary.

1.1.1 Proof of Work

For Proof of Work, we use something called a **"Puzzle-friendly hash function"**. A hash h is puzzle-friendly, if for every output y, it is "somewhat" hard to find an x such that $h(k||x) = y$, for some given k.

So, for example, we might ask a user to find an x such that $H(B||x)$ is equal a string with some prefix of n 0's, where B is a block of transactions. The number n is called the "hardness parameter".

As long as H has no obvious exploitable structure, there is no better strategy to solve this than guessing random values of x over and over.

Whichever user finds some x that "solves the puzzle" first "verifies" block B and get a reward. This is mining.

So, each block in the chain has the hashed value (B) and the value x found by the miner. In general, we increase the hardness parameter as the number of participants increases, so that it takes about 10 minutes on average to approve a new block.

A small twist here is that since we are using hashes, we can find different working x's later on, and this creates a "fork". We resolve this by always using the longest chain of valid (verified) transactions.

Putting this all together, anyone who wants to make bad blocks would also have to get them approved, and would have to get them approved faster than honest blocks are being approved. In order to do this, an adversary would need more computational power than everyone else combined, or else the honest players would create an honest chain longer than the adversarial chain and their chain will be ignored.

We choose 10 minutes for the average approval time because it takes about 1 minute for the update to propagate through the network, so we want the time to approve to be larger so that the probability of accidentally forking (2 honest solutions found at the same time) becomes very small. We don't want the time to be too large, though, since people are doing transactions and we don't want to have to wait too long for them to be verified.

Note that here, we have only enforced that, if players have equal compute power, they can agree with $t < n/2$. But in byzantine agreement, we needed $t < n/3$. The discrepancy is that the adversary had to simulate 4 players, and in doing so, did at least twice as much work as all the honest players. So, the proof doesn't apply when we are considering only equal-powered players.

1.1.2 Proof of Stake

This is used by Ethereum 2.0, Cardano, and Algorand, and it is built using unique signatures, which were discovered by Goldwasser and Ostrovsky.

As one might guess, unique signatures are any digital signature scheme with keygen, signature, and verification stages, where any signature of a document by any individual is completely unique. However, note that at least one bit of the signature must necessarily be unpredictable, otherwise it could be forged easily.

A related concept is a Verifiable Random Function (VRF), due to Rabin, Kilian, and Micali. The important thing is that we want uniqueness and unpredictability to hold even if public keys are maliciously chosen.

Proof of Stake is the following:

1. Run a lottery/raffle where the probability of winning depends on how much currency (stake) you have in the blockchain.
2. If you win the lottery, announce it to everyone
3. use Byzantine agreement among the small number of winners to agree on the next block.

The unique signatures comes in when we run the lottery. Each player signs the previous block with their secret key, where the number of 0's prefixing the signature depends directly on how much stake the player has. The winner is chosen based on how many 0's (make it more likely for large # to win by saying "winner has at least n 0's"). A player can prove that they won by publishing their signature.

The security guarantee is that if less than 1/3 of the currency is held by an adversary, then we expect fewer than 1/3 of the lottery winners to be dishonest, with high probability. So, we have a byzantine agreement less than 1/3 of the players are dishonest, and they can work to approve an honest block.

We note, but do not discuss, the fact that if more than 2/3 of the byzantine players are dishonest, the dishonest committee can actually create a fork. Then, we break the longest-chain rule, and we need something a bit more complicated.

We note that Cardano does not use the lottery exactly as described above, instead having only one winner of the lottery and having that one winner approve or deny the block, rather than using byzantine agreement.

1.2 Smart Contracts

Standard cryptocurrencies/blockchains only keep track of transactions, but what if we wanted to include conditional statements based on arbitrary programs? This is what smart contracts do.

For example, we might have something like the following:

```
If Program(input = 100 future blocks) = true:
Then: A gives B  1 Ethereum
```

The program might be anything, like checking of one of the transactions says something specific about B or A. In practice, the programs aren't totally arbitrary, since you have to pay "gas" to miners, which costs more if the program is more complex.