Blockchain

Anjali Pal

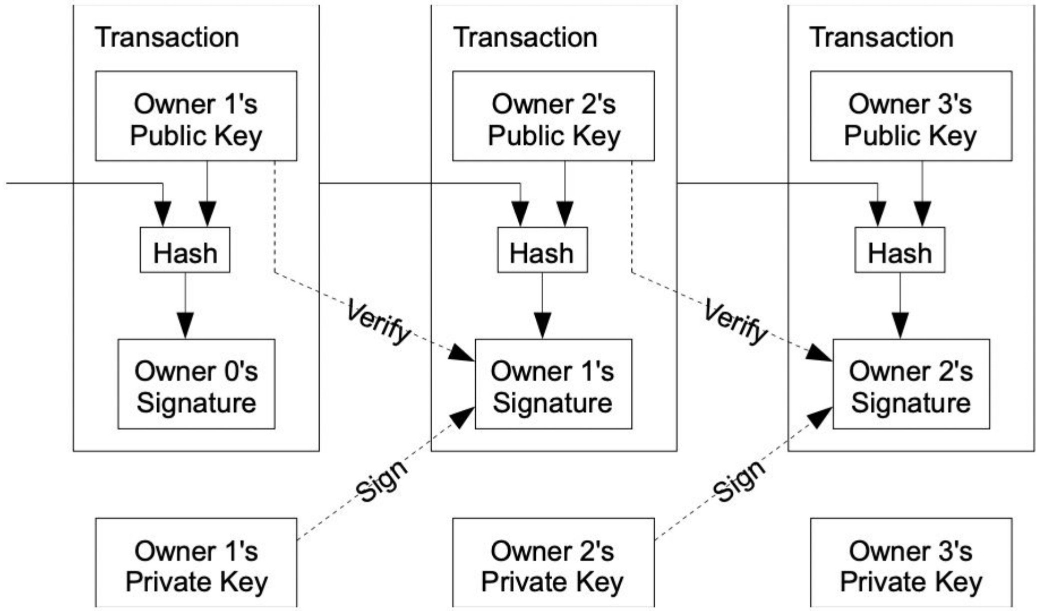
1. Introduction

System for electronic transactions that does not rely on trust

- Coins made from digital signatures give strong control of ownership, but no protection against double-spending
- Peer-to-peer network using proof-of-work to record the public transaction history
- Computationally impractical for an attacker to change the record as long as honest nodes control a majority of CPU power
- “Unstructured simplicity”
  - Nodes work with little coordination
  - Can leave and rejoin at will

2. Transactions

- Chain of digital signatures
- The owner of a coin can transfer it to the next owner by digitally signing a hash of the previous transaction and the public key of the next owner

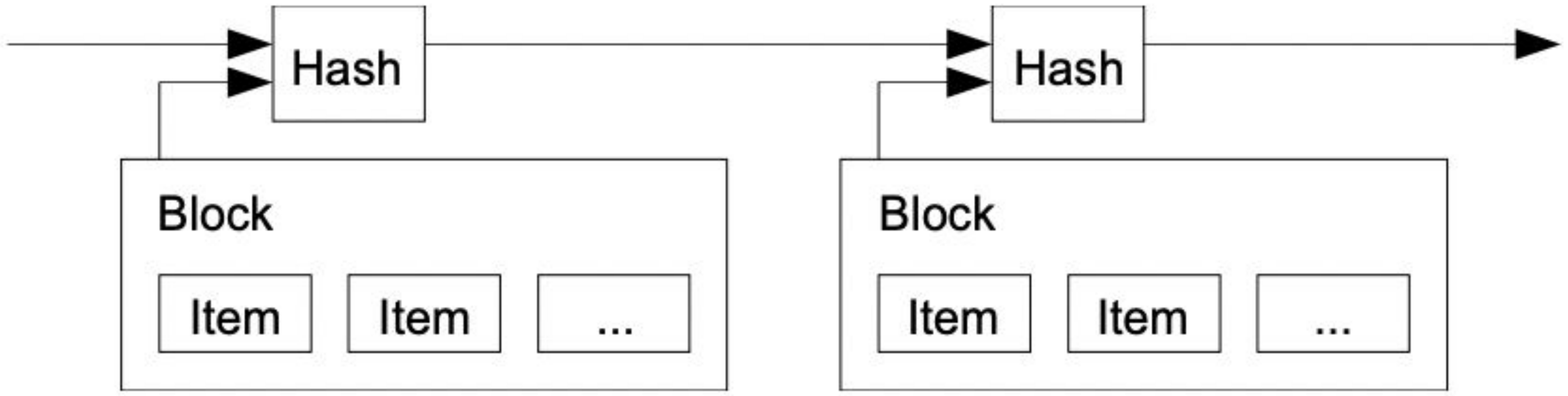


2. Transactions

- Payees can verify the signatures on a coin to verify the chain of ownership
- There’s no way to verify that the owner didn’t “double spend” the coin (transfer the same coin to two different new owners)
- Two potential solutions:
  - Add a central authority (mint) that certifies each transaction to guarantee that coins are not double-spent
  - Make every transaction public and create a system so that all participants agree on the order history of all transactions

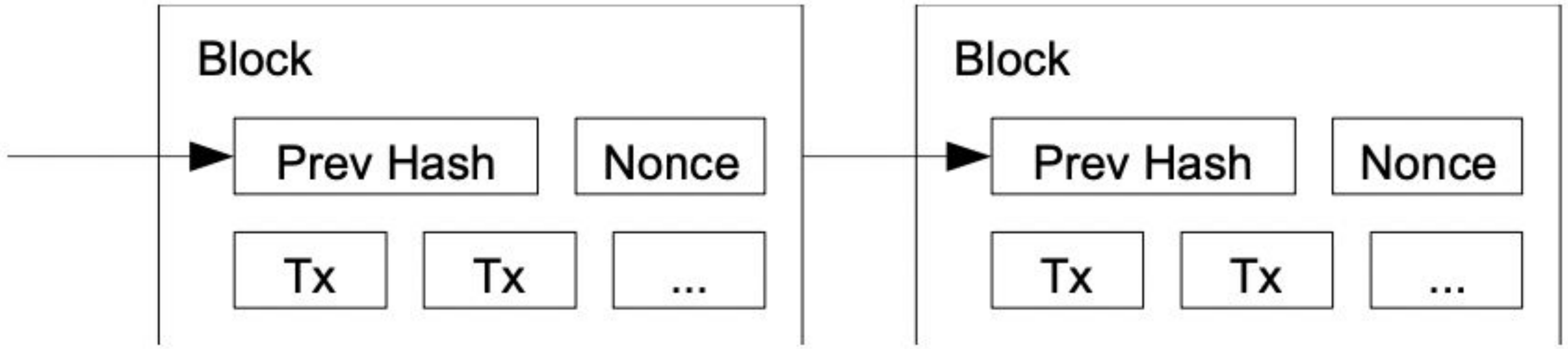
3. Timestamp Server

- Proves that data exists at a given time
- Each timestamp includes the previous timestamp in its hash, forming a chain



4. Proof-of-Work

- Scan for a value that, when hashed, begins with a specified number of zero bits
- The work to find such a value is exponential in the number of zero bits required and can be verified in constant time (calculating a single hash)
- A block cannot be changed without redoing the work for that block **and all of the blocks that follow it in the timestamp chain**



4. Proof-of-Work

- If a majority of CPU power is controlled by honest nodes, the honest chain will outpace any competing chains
- To forge a transaction, an attacker would have to redo the proof-of-work of the block it wants to modify and then catch up with and surpass the honest chain

5. Network

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. Each node works on finding a proof-of-work for its block
4. When a node finds a proof-of-work, it broadcasts the new block to all nodes
5. Nodes accept the block if all the transactions are valid
6. If the block is good, nodes will start working on the next block in the chain, using the hash of the accepted block as the previous hash

5. Network

- It’s possible that nodes might receive different values for the next block. They work on the first one they received, but save the other branch in case it becomes longer, in which case they’ll switch over to the other branch
- New transaction broadcasts don’t need to reach all nodes
    - As long as enough nodes get the message, they’ll get into a block
- Block broadcasts are tolerant of dropped messages
    - Nodes will notice they missed a block and request it

6. Incentive

- There needs to be sufficient financial incentive for nodes to behave honestly. It should be more profitable to play by the rules than to try to defraud the network
- Two profit models for nodes:
    - Creating new coins
    - Transaction fees

7. Reclaiming Disk Space

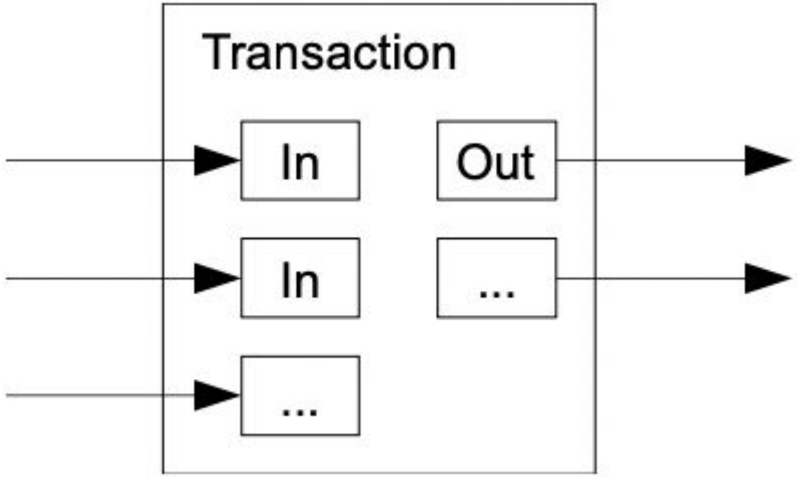
- Don’t want to store all of the transactions in the history forever
- Transactions are hashed in a Merkle Tree with only the root included in the block’s hash
- Old blocks can be compacted by stubbing branches of the tree and interior hashes do not need to be stored

8. Simplified Payment Verification

- Users can verify payments without needing to run a full node
    - Query the nodes to get the longest chain
    - Find the Merkle branch linking the transaction to the block it’s timestamped in
- This doesn’t verify the transaction itself, but it does show that the network has accepted it
    - As long as you trust the network, you can trust the transaction

9. Combining and Splitting Value

- It would be possible, but unwieldy, to make separate transactions for every cent in a payment
- Transactions accept multiple inputs to combine smaller amounts
- Transactions have two outputs: one for the payment and one to return change (if any) to the sender



10. Privacy

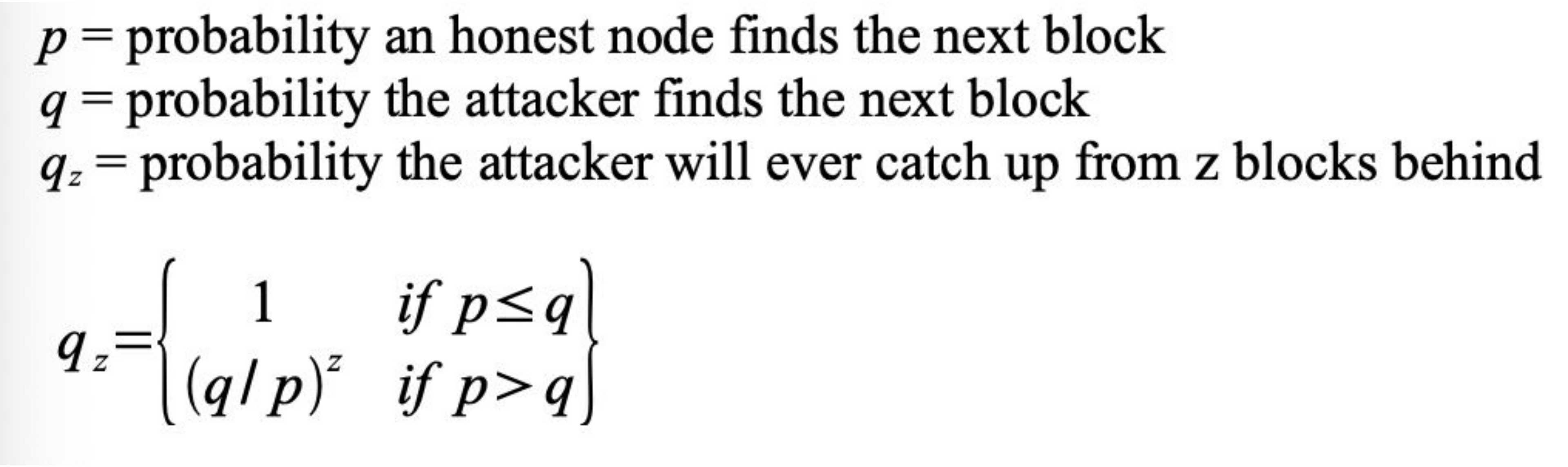
- All transactions are public, but privacy can still be achieved if transactions are anonymous
    - You can see that someone sent an amount to someone else, but without any information about who the participants are
- Use a new key pair for each transaction to keep them from being linked to a common owner
- Linking is unavoidable with multi-input transactions (all of the inputs are revealed to be owned by the same entity)

11. Calculations

- An attacker cannot create or steal value
  - Honest nodes will never accept invalid transactions or blocks containing them
- An attacker can only try to change its own previous transactions to take back money it recently spent
  - In order to do this, the attacker would need to change the block containing the transaction and make its chain longer than the honest chain

11. Calculations

The probability that an attacker catches up with the honest chain



p = probability an honest node finds the next block

q = probability the attacker finds the next block

q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

11. Calculations

- We assume p > q (honest nodes have more CPU power than dishonest nodes)
- The probability that the attacker catches up decreases exponentially as the number of blocks it is behind by increases
- If a recipient of a transaction waits until z blocks have been linked after it, then the probability of an attacker being able to change the transaction at that point is given by

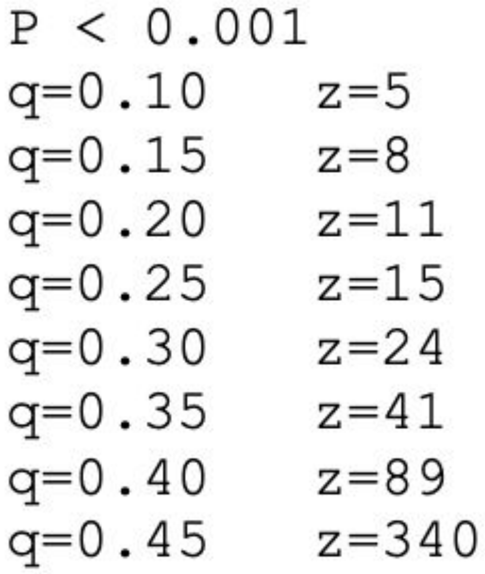
$$1-\sum_{k=0}^{z}\frac{\lambda^{k}e^{-\lambda}}{k!}\left(1-(q/p)^{(z-k)}\right)$$

z = number of blocks after the transaction
p = p(honest node finds next block)
q = p(attacker finds next block)
λ = z * q/p

11. Calculations

The main takeaway here is that it doesn’t take very many transactions to make it extremely unlikely that an attacker can catch up

As long as the majority of the CPU power in the network is controlled by honest nodes, it is very difficult for attackers to exploit the network.



```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

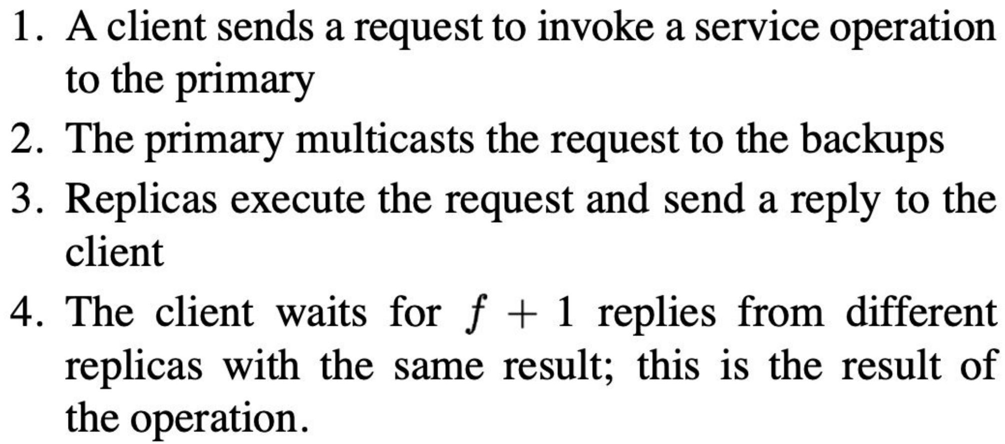
- Coins made from digital signatures don’t prevent double spending
- Peer-to-peer network using proof-of-work to record a public history of transactions
- It is computationally impractical for an attacker to change the history
- Nodes work simultaneously with little coordination
- Nodes can leave and rejoin at will and the network is robust to dropped messages
- Nodes “vote with CPU power”- they accept blocks by working on them or reject blocks by refusing to work on them

Proof-of-Stake

- Nodes become validators by locking a certain amount of coin in the network as “stake”
- One validator is randomly selected each timeslot to create a new block and send it to other nodes on the network
- There is also a committee of validators in each timeslot, whose votes determine the validity of the proposed block
- Transaction fees are the incentive for nodes to participate honestly as validators
- If validators accept fraudulent transactions, they lose a portion of their stake
- Attacker would need to control 51% of the value in the network in order to approve fraudulent transactions
- Much more energy-efficient than proof-of-work

Practical Byzantine Fault Tolerance

- Byzantine-fault-tolerant algorithms are increasingly important as malicious attacks become more common and sophisticated
- Presents a new algorithm for state machine replication that is fast enough for practical use
- Liveness and Safety are guaranteed if at most $\lfloor (n - 1) / 3 \rfloor$ replicas are faulty



1. A client sends a request to invoke a service operation to the primary
2. The primary multicasts the request to the backups
3. Replicas execute the request and send a reply to the client
4. The client waits for $f + 1$ replies from different replicas with the same result; this is the result of the operation.

From Viewstamped Replication to Byzantine Fault Tolerance

Viewstamped Replication

- 2f + 1 to tolerate up to f node crashes
- Clients send requests to Primary node, who sends a message to the other replicas
- Non-primary replicas process messages in order and responds to the Primary
- Once enough replicas respond, the Primary commits the operation and notifies all of the replicas about the commit

Byzantine Fault Tolerance

- 3f + 1 replicas to tolerate up to f faulty nodes
- Adds an extra phase to VR to prevent a malicious primary from misleading the other replicas
- Introduces certificates to ensure that all committed operations make it into the next view