Bitcoin Presentation Notes

1 Motivation

- Situation: commerce on the Internet relies almost exclusively on financial institutions serving as trusted third parties to process electronic payments.
- Problem: such third parties may not be trustworthy.
- Goal: develop a decentralized version of electronic cash that would allow online payments to be sent directly from one party to another without going through a financial institution.

2 Detour: Digital Signatures

- Scenario: Alice wants to send a message to Bob over a network. How can Bob verify that the message he received was from Alice? Alice needs to sign her message.
- Desired property of signature: cannot be forged on a different message.
- Both Alice and Bob each generate a private/public key pair for themselves. Each key is some string of bits. Private keys are kept secret.
- Producing a signature: $\mathsf{sign}(\mathsf{message}, \mathsf{privateKey}) = \mathsf{signature}$.
- Verifying a signature: $\mathsf{verify}(\mathsf{message}, \mathsf{signature}, \mathsf{publicKey}) = \mathsf{true}/\mathsf{false}$.
- Only the owner of the private key can produce the signature.
- No one can copy the signature and forge it on another message.
- Signature is a 256 bit value. Hard to find a valid signature if you don't know the secret key. There is no strategy better than guessing and checking if random signatures are valid using the public key. There are 2^{256} signatures to check; this is a very large number.

3 Bitcoin

- Bitcoin network: a randomly connected overlay network of a few thousand nodes, controlled by a variety of owners. All nodes perform the same operations, i.e., it is homogeneous network and without central control.
- Users generate private/public key pairs.
- Address: an identifier derived from a user's public key to receive funds in Bitcoin.
- The Bitcoin network collaboratively tracks the balance in bitcoins of each address.

4 Transactions

- Output: a tuple consisting of an amount in bitcoins and a spending condition requiring a valid signature associated with the private key of an address.
  - Outputs exists in two states: unspent and spent.
  - Any output can be spent at most once.
  - The address balance is the sum of bitcoin amounts in unspent outputs that are associated with the address.
  - The set of unspent transaction outputs (UTXO) and some additional global parameters is the shared state of Bitcoin. Every node in the Bitcon network holds a complete replica of that state. Local replicas may temporarily diverge but consistency is eventually re-established.
- Input: a tuple consisting of a reference to a previously created output and a signature to the spending condition, proving that the transaction creator has the permission to spend the referenced output.
- Transaction: a data structure that describes the transfer of bitcoins from spenders to recipients. The transaction consists of a number of inputs and new outputs. The inputs result in the referenced outputs spent (removed from the UTXO), and the new outputs being added to the UTXO.
- A transaction from Alice to Bob:
  - Sender: Alice.
  - Recipients: Bob.
  - Inputs: one or more tuples where each tuple consists of a reference to one of Alice's previously created unspent outputs and Alice's signature using her private key.

  - Output: a tuple consisting of the amount in bitcoins for Bob and a spending condition requiring a valid signature associated with the private key of Bob's address.

- Transactions are broadcast in the Bitcoin network and processed by every node that receives them.
- The outputs of a transaction may assign less than the sum of inputs, in which case the difference is called the transaction's fee. The fee is used to incentivize other participants in the system.
- Transactions are in one of two states: unconfirmed or confirmed. In coming transactions from the broad cast are unconfirmed and added to a pool of transactions called the memory pool.

5 Doublespends

- Problem: nothing prevents nodes to locally accept different transactions that spend the same output. For example, the transaction from Alice to Bob could have spent the same output as the transaction from Alice to Charlie.
- Doublespend: a situation in which multiple transactions attempt to spend the same output. Only one transaction can be valid since outputs can only be spent once. When nodes accept different transactions in a doublespend, the shared state becomes inconsistent.
- Doublespends can result in an inconsistent state since the validity of transactions depends on the order in which they arrive. If two conflicting transactions are seen by a node, the node considers the first to be valid. The second transaction is invalid since it tries to spend an output that is already present. The order in which transactions are seen may not be the same for all node, hence the inconsistent state.
- If doublespends are not resolved, the shared state diverges. Therefore a conflict resolution mechanism is needed to decide which of the conflicting transactions is to be confirmed (accepted by everybody), to achieve eventual consistency.

6 Detour: Cryptographic Hash Functions

- $\mathsf{SHA256}(\mathsf{message}) = \mathsf{hash}$.
- Input: message.
- Output: String of 256 bits, known as the hash of the message.

- The output looks random even though it isn't (i.e., the function always gives the same output for a given input).
- Slightly changing the input results in the output to change completely in an unpredictable way.
- Infeasible to compute in the reverse direction. Given only the output, finding the input that hashes to the output is extremely difficult. There is no better method than to guess and check. Need to go through 2^{256} guesses.
- However, given the input message, the hash is fast to compute.

7 Proof-of-Work and Blocks

- Proof-of-Work (PoW): a mechanism that allows a party to prove to another party that a certain amount of computational resource has been utilized for a period of time.
- Block: a data structure used to communicate incremental changes to the local state of a node. A block consists of a list of transactions, a reference to a previous block and a nonce. A block lists some transactions the block creator ("miner") has accepted to its memory-pool since the previous block. A node finds and broadcasts a block when it finds a valid nonce.
- A valid nonce is a number such that the SHA256 hash of the block begins with a required number of zero bits.
- A node finds a valid nonce for a block by incrementing the nonce from zero and checking if the SHA256 hash begins with the required number of zero bits.
- Once a node finds a valid nonce for a block, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.
- The PoW also solves the problem of determining representation in majority decision making. The majority decision is represented by the longest chain, which has the greatest PoW effort invested in it.

8 Network

- The network executes as follows:
  1. New transactions are broadcast to all nodes.
  2. Each node collects new transactions into a block.
  3. Each node works on finding a difficult proof-of-work for its block.

4. When a node finds a proof-of-work, it broadcasts the block to all nodes.
5. Nodes accept the block only if all transactions in it are valid and not already spent.
6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

- Nodes always consider the longest chain to be the correct one and will keep working on extending it.
- If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received but save the other branch in case it becomes longer; the nodes that were working on the other branch will then switch to the longer one.
- New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long.
- Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block.