EE 374: Fundamentals of Blockchain Infrastructure **Stanford, Spring 2025**

Lecture 2: Introduction to Bitcoin

April 2, 2025

Lecturer: David Tse Scribe: Naomi Mo

Lecture Reading: "Bitcoin: A Peer-to-Peer Electronic Cash System" (`https://bitcoin.org/bitcoin.pdf`)

1 Introduction

Bitcoin is a decentralized payment system that uses a longest chain, proof-of-work consensus protocol to validate transactions and secure the network, ensuring that no single entity can manipulate the system or alter transaction history. This lecture explores the origins and motivations behind Bitcoin, the issues it aims to solve (data integrity and data agreement), and the cryptographic techniques it employs to address them (digital signatures and cryptographic hash functions).

2 Satoshi Nakamoto and the Creation of Bitcoin

On February 11, 2009, a message posted by user "Satoshi Nakamoto" appeared on the website *P2P Foundation* announcing a "new open source P2P e-cash system called Bitcoin", citing that the root problem with traditional currency is trust. For example, we must trust banks to hold and manage our money, facilitate transactions, and maintain our privacy; however, banks often breach this trust, whether by lending our money away in credit bubbles (as observed in the 2008 financial crisis) or falling victim to identity theft perpetrated by malicious actors. Nakamoto posited that strong encryption has now enabled humanity to remove trust from the loop, thus motivating the creation of Bitcoin (and subsequently the development of a 3-trillion dollar industry).

Nakamoto elaborates on this system further in his whitepaper "Bitcoin: A Peer-to-Peer Electronic Cash System", which was published online just a few months prior in October 2008. [3] The abstract of the paper states that the cash system "requires minimal structure" and is permissionless, meaning that nodes can leave and rejoin the network at will. It also delineates the use of digital signatures and a peer-to-peer network to solve data integrity and data agreement issues.

3 Peer-to-Peer Networks and Ledgers

Bitcoin is a **peer-to-peer (P2P) network**, meaning that it involves a collection of nodes maintaining (i.e., achieving agreement on) a **ledger**, or a sequence of financial transactions. Any node can join or leave the system, and each node maintains its own local copy of the sequence of transactions.

It is difficult to ensure synchronization across these local copies. For instance, say that you and three friends (Alice, Bob, and Charlie) each maintain your own transaction history (as observed in Figure 1).

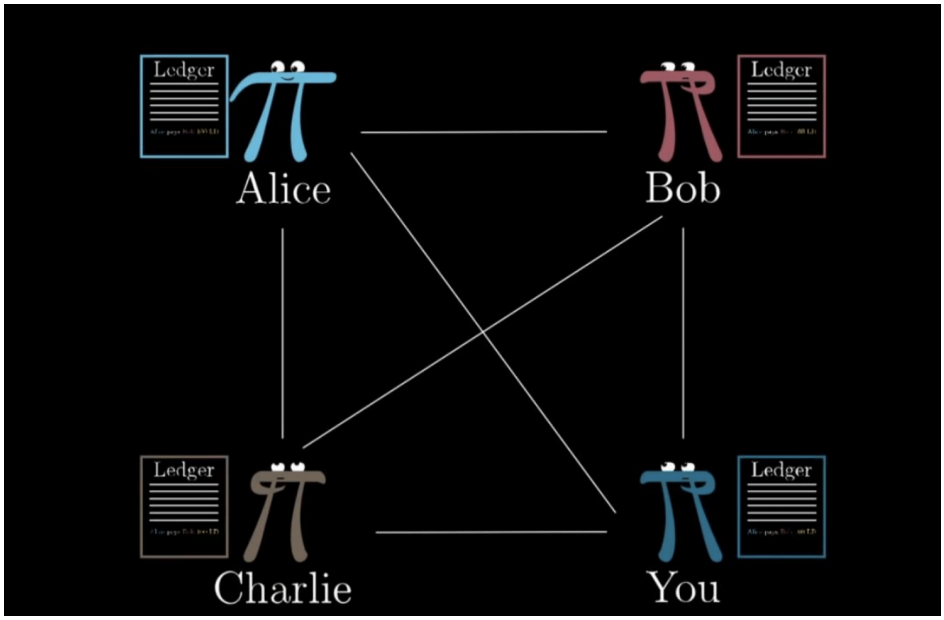

Figure 1: You, Alice, Bob, and Charlie maintain your own ledgers of the system's transactions.

Each transaction history records the change of ownership of a bitcoin from one holder to another, e.g. "Charlie pays you 1 bitcoin" or "Alice pays Bob 1 bitcoin". All ledgers in this system must record valid transactions, and they must list these transactions in the same order. Otherwise, the ledger will be incorrect or inconsistent, and the true state of the system will be lost (for example, Alice may believe you have 2 BTC while Bob may believe you have 3 BTC).

We observe that in order for this ledger system to work, it must have:

1. **Data Integrity**: Data must be legitimate. Nodes in a system should only allow good data and reject bad data.
2. **Data Agreement**: All nodes agree on data at all times; everyone maintains the same view of history. Without data agreement, there exists the potential for data/currency to be spent multiple times, known as the "double spending" problem.

4 Data Integrity and Digital Signatures

To solve the issue of data integrity, Bitcoin uses a cryptographic technique called a digital signature, which allows a user to ensure the authenticity of a transaction from a given sender.

We define a digital signature scheme as the one presented by Dan Boneh and Victor Shoup [2], involving three algorithms:

1. **a key generator G**: a probabilistic algorithm that takes no input and outputs a pair of keys (*pk, sk*) (one public, one secret).

2. **a signer S**: a probabilistic algorithm that takes the secret key sk and a message m as input and then outputs a signature σ.

3. **a verifier V**: a deterministic algorithm that takes the public key pk, message m, and signature σ as input and then outputs either **accept** or **reject**.

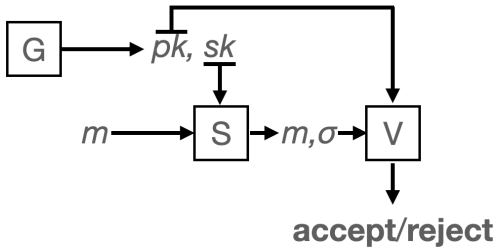

Figure 2: Block diagram of the signature scheme.

As an example of the signature scheme, recall our friends Alice, Bob, and Charlie, and observe their transactions in Figure 3.

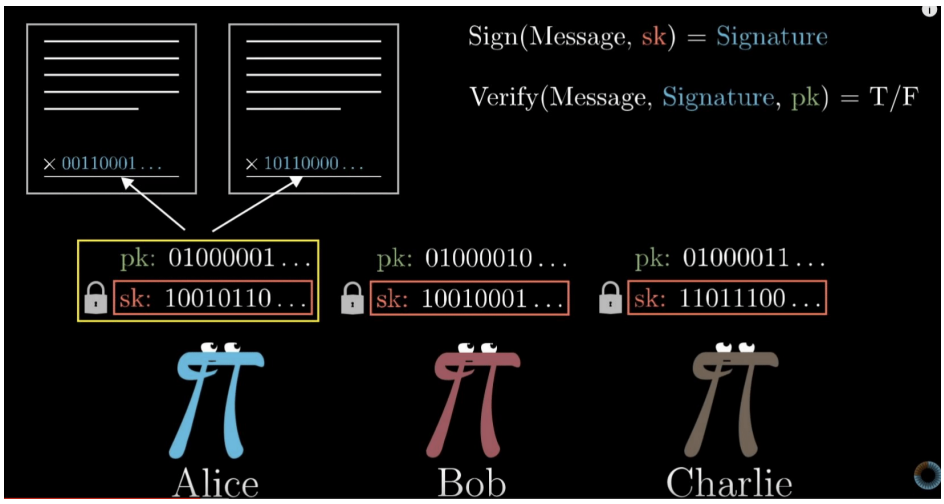

Figure 3: Alice, Bob, and Charlie and their keys.

If Alice wants to pay Bob one bitcoin, she can draft the message "Alice pays Bob 1 bitcoin" (though in reality, her message will use Bob's public key rather than his name). The sign function S will use Alice's message and secret key to create a unique signature; Alice will then send this signature with her message into the network as a complete transaction. The verifier V will then receive this message, signature, and Alice's public key, and output either **accept** or **reject**.

It is important to note that the signature generated by S should always be accepted by V; that is, for all generated keys and messages, $\mathbf{P}r[V(pk, m, S(sk, m)) = \text{accept}] = 1$. This is considered the protocol's "happy path", which means the protocol is operating under normal functioning (i.e., without attacks).

When the protocol is under attack and must reject the attack, it is considered to be on the "unhappy path." Good signature schemes will satisfy the "no-forgery" security property, meaning that no (computationally bounded) attacker can forge a signature on any chosen message with non-vanishing probability. We use a **signature attack game** (figure 4) to illustrate this concept. Consider an adversary $\mathcal{A}$ mounting a chosen message attack on another node (which we shall call the challenger), meaning $\mathcal{A}$ can request the challenger's signature and choose any message to forge a signature on.

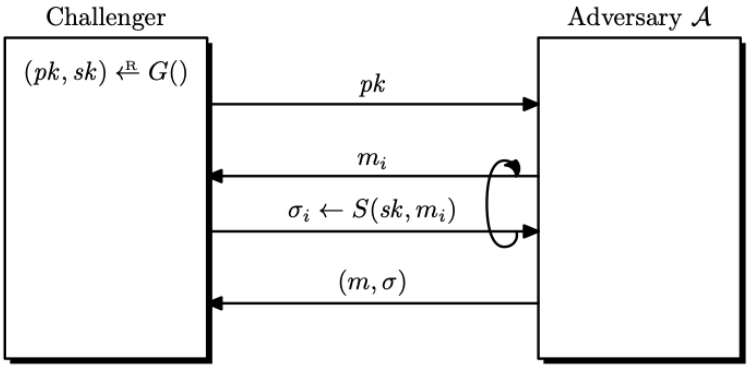

Figure 4: Signature attack game, from Boneh & Shoup 2023. [2]

Since $\mathcal{A}$ does not have access to the challenger's private key sk, they cannot directly generate the signature σ for a new message m, so they attempt to achieve a correct signature by iterating different message queries m_i. For each m_i, they are returned a signature σ_i from the challenger.

However, brute-forcing messages to find a correct (m, σ) pair would take significant computational resources given the vast nature of mappings to the signature space. Therefore, without the private key, generating a valid signature with a new message m (one not previously queried) is rendered computationally impractical, and the signature scheme is considered sound from $\mathcal{A}$'s attacks.

On this note, we conclude the section with a summary of the two properties that comprise a good signature scheme:

1. **Correctness**: "good stuff happens." The verifier accepts a proper signature.
2. **Soundness/security**: "Bad stuff doesn't happen." The protocol is impervious to attacks.

Most systems are designed to simultaneously achieve these two properties. However, there is one small caveat: we assume the adversary is not a super computer, and that they are computationally

bounded. This means that the adversary's computational power cannot be exponentially as large as the key bits; otherwise, they have the computational capability to create an existential forgery.

5 Data Agreement and Cryptographic Hash Functions

To motivate the issue of data agreement, we examine the **Unspent Transaction Output (UTXO) Model**, which is the foundation of Bitcoin's operation. In the UTXO model, each bitcoin has an associated public key that proves ownership (consider $pk1$ as a bitcoin's first ownership, $pk2$ for the second, and so on as in Figure 5). In this model, transactions have both an input and an output: they take unspent transaction outputs as input (ensuring transfers cannot be initiated from spent transactions) and then generate a new unspent transaction output, signifying a transfer.

Figure 5: These public keys refer to different reincarnations of the same bitcoin under different ownerships.

Say that the holder of $pk1$ wants to transfer their bitcoin to a new holder, so they initiate a transaction. This transaction would operate as follows:

1. **Key Generation**: The new recipient generates a public-private key pair $(pk2, sk2)$; the public key $pk2$ is shared with the original holder $pk1$, while the private key $sk2$ is kept secret.
2. **Signing the Transaction:** The current holder of $pk1$ signs a message of the form "I give this bitcoin to $pk2$" using their private key $sk1$.
3. **Verification:** The recipient verifies that the message is valid using the sender's public key $pk1$.
4. **Ownership Transfer:** If the message is accepted by the verifier, then the bitcoin is transferred to $pk2$. The holder of $pk2$ is now able to repeat the process and transfer the bitcoin to another recipient.

We observe that bitcoin ownership evolves overtime as it passes from one public key to another: $pk1 \rightarrow pk2 \rightarrow pk3 \rightarrow ...$ and so on. Now, while the digital signature scheme ensures that all transactions are valid, there remains the **double spending** problem.

Say that there are three friends: Paul, John, and Peter. Paul owns 1 bitcoin and sends two messages into the network, as illustrated in Figure 6: "Paul pays John 1 bitcoin" and "Paul pays Peter 1 bitcoin." Both of these transactions are completely valid, as they (1) come from an unspent transaction output and (2) both come with valid signatures. However, this situation creates inconsistent ledgers, as the same bitcoin is being spent twice. Nodes that first observe John receiving the

bitcoin will believe that John owns the coin, and reject Peter's ownership because it comes from a spent transaction. The opposite belief will occur for the nodes that first observe Peter receiving the bitcoin.

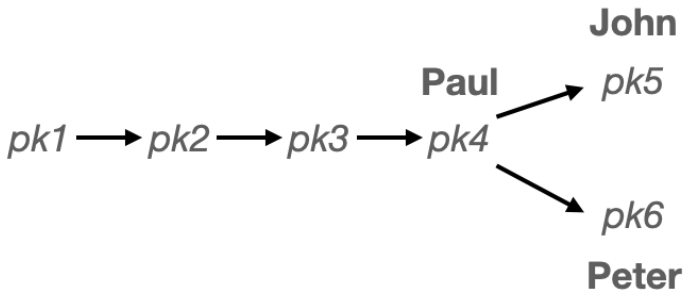

Figure 6: Two valid transactions from Paul to both John and Peter.

To prevent double spending, the network must maintain **consensus** on the sequence of events. However, we observe two primary challenges for achieving consensus:

1. **Synchronization**: Different nodes may have different transaction orderings. They may try to reconcile their views via majority voting, but network delays make it difficult to synchronize voting to happen at the same time for every node.

2. **Sybil Attacks**: In a permissionless system, anyone can join the network and create multiple identities. This has the potential to lead to unstable decision making if a majority vote is used as the basis for consensus (imagine an adversary creating many accounts and dominating the network).

To address these issues, Bitcoin uses a **proof-of-work (PoW), longest chain protocol**; we will first examine the longest chain aspect. Imagine we have a blockchain of one block in the network, which we call the **genesis block**. This is depicted in Figure 7, which also shows the network nodes across the globe.

Nodes distributed around the world engage in a process called **mining** to generate new blocks and add them to the blockchain. When a new block is mined, it is verified by the network before being appended to the chain. However, because of network delays or adversarial actions, different miners may sometimes build on different blocks simultaneously. These timing differences lead to temporary divergences, known as **forks**, in the blockchain. To resolve these forks and maintain a consistent history, the blockchain employs the longest chain protocol: nodes accept the chain with the greatest cumulative proof-of-work (the longest chain) as the valid one. Shorter chains or forks are then discarded, ensuring that a single sequence of events is maintained across the network.

Figure 7: Distributed nodes mining for a block to add to the blockchain

Next, we examine the proof-of-work concept, which is a design principle used to prevent miners from adding blocks at will to dominate the Bitcoin system. The PoW method ensures that adding a new block requires significant computational effort, which discourages malicious actors from attempting to take control of the network.

The mining process is as follows:

1. A miner intending to mine a block obtains the hash of the previous block, called *Prev*. Note that each block contains a reference to the previous block's hash, ensuring that the blocks are linked together in a chain.

2. The miner collects all validated transactions Tx from the network that they wish to put in their new block (e.g., Paul gave 1 BTC to Peter).

3. After assembling the data/transactions into a block, the miner attempts to solve a computational puzzle of the form "Hash$(Prev, Tx, nonce) <$ threshold", as seen in Figure 8. The miner must use trial and error to find a value of the nonce that satisfies the inequality.
    - The hash function used in this inequality is a SHA-256 Hash Function that takes any size input data and produces a 256-bit (32-byte) hash value output.
    - The term **nonce** is short for "number used once", and is the cryptographic name for a variable to solve.
    - The **threshold** is a target value that the computed hash must be less than in order for the block to be considered valid; this threshold is not a fixed number, and is adjusted periodically based on the cost of energy and hardware. The number of leading zeros in the threshold determines the computational difficulty (i.e., a threshold with 80 leading zeroes would require - on the average - 2^{80} attempts).

4. Once the miner finds a valid solution, their block will be added to the blockchain as it has satisfied the required proof-of-work.

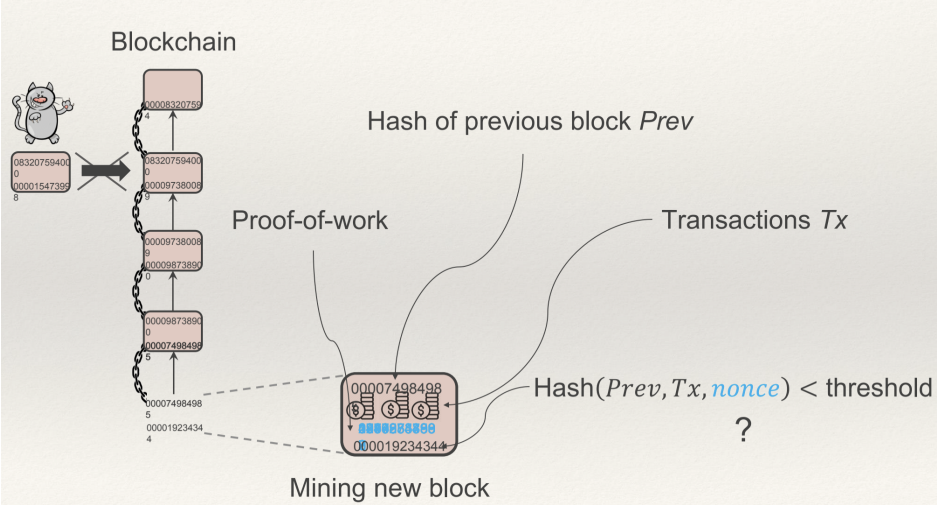

Figure 8: A miner's "proof-of-work" is solving a computational puzzle. Once this puzzle is solved, the miner will add the proof of their solution (the nonce) to the block, then add the block to the chain.

References

[1] 3Blue1Brown. But how does bitcoin actually work?, 2017.

[2] D. Boneh and V. Shoup. Definition of a digital signature. In *A Graduate Course in Applied Cryptography*, pages 528–529. 2023.

[3] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.