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

Lecture 3: PoW Longest Chain Protocol: Attacks and Security Analysis

April 7, 2025

Lecturer: Prof. David Tse Scribe: Nathan Kuo

1 Introduction

This lecture explores the properties of proof-of-work, focusing on the role of cryptographic hash functions and the security of the longest chain protocol. Two foundational cryptographic primitives are involved: **Digital signatures**, which ensure **data integrity**, and **hash functions**, which support **data agreement** by enabling consensus on a single, tamper-evident version of the blockchain.

2 Why are hash functions a good cryptographic scheme for PoW?

In summary, miners create new blocks through the following process:

1. **Gather transactions:** Miners collect pending transactions from the mempool (the pool of unconfirmed transactions).
2. **Construct the block:** They assemble a candidate block, which includes:
   - The hash of the previous block
   - A selection of transactions (usually prioritized by fee)
   - A nonce (an arbitrary number to vary the input to the hash function)
3. **Solve the proof-of-work puzzle:** Miners repeatedly vary the nonce until they find one that satisfies the condition:

$$\text{Hash}(\text{Prev}, \text{Txs}, \text{nonce}) < \text{threshold}$$

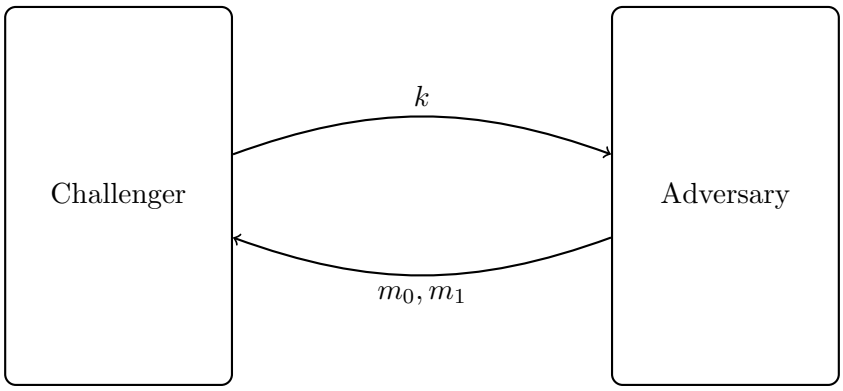
   - The hash function used is SHA-256, which outputs a 256-bit value.
   - The threshold is a target value that determines the mining difficulty.
   - A lower threshold means more leading zeros are required in the hash output, making it harder to find a valid nonce.
   - As of now, the target hash has 80 leading zeros in binary.
4. **Fee prioritization:** Transactions with higher fees are more likely to be included in a block sooner. Those with lower fees may experience delays before being confirmed.

We can define a hash function as the one presented by Dan Boneh and Victor Shoup [1]:

Definition 7.1 (Keyed hash functions). *A **keyed hash function** H is a deterministic algorithm that takes two inputs, a **key** k and a **message** m; its output $t := H(k, x)$ is called a **digest**. As usual, there are associated spaces: the keyspace $\mathcal{K}$, in which k lies, a message space $\mathcal{M}$, in which m lies, and the digest space $\mathcal{T}$, in which t lies. We say that the hash function H is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$.*

In the context of proof of work, the **key** to the hash function can be thought of as the combination of the **previous block hash** and the **set of transactions**, while the **message** is the **nonce**. Together, these inputs are used to compute the hash.
An important property that hash functions should satisfy is the **collision resistance property**.

Collision resistance is the property that, given a key k, it is computationally infeasible to find two distinct messages m_0 and m_1 such that: $H(k_1, m_0) = H(k_1, m_1)$. Collision resistance can be expressed in terms of a game:



$H(k_1, m_0) = H(k_1, m_1) \Rightarrow$ A wins. The probability that the adversary wins is negligible and the adversary is computationally bound.

Intuitively, collision resistance means it is hard to find two different inputs that produce the same hash output. This is because the hash function maps a very large input space (potentially infinite-length messages) into a fixed-size output space—in this case, a 256-bit digest space.

However, **this property itself is not sufficient for ensuring the difficultyt of PoW**.

The random oracle model is an idealized abstraction of a hash function. In this model, when given an input pair (k_n, m), the hash function H produces an output that is uniformly random over the digest space T. Upon receiving the same inputs, the hash function outputs the previously sampled value.

The random oracle model implies collision resistance. If a hash function behaves like a

random oracle, then the probability of two distinct inputs producing the same output is extremely low. For a 256-bit output space, the chance of a random collision is about $1/2^{256}$, which is negligibly small.

Because random oracle implies collision resistance, **collision resistance is a weaker property.** To support the fact that proof of work is difficult, we need the random oracle model. There is no shortcut to solving the proof-of-work puzzle: one must repeatedly evaluate the hash function with different inputs (nonces) until a valid output is found. The function is non-invertible, and the puzzle cannot be solved any quicker.

3 Consensus Protocol Security and the Private Double-spend attack

Consensus protocols are often built upon cryptographic primitives. To evaluate the security of a consensus protocol, we must ask: what is the security level of the protocol given security of the underlying cryptographic primitive. In the case of Bitcoin, we are interested in analyzing the risk of **double spending** assuming that the underlying hash function satisfies the random oracle model.

Here's an example of a **double spend attack**:

Suppose Alice, who is also a miner, wants to buy a car. She initiates a payment by signing the message "Alice pays Bob one BTC," which is then broadcast to the network. A miner, "M", includes this transaction in a new block, which is successfully added to the blockchain.

However, Alice is malicious. She secretly mines two new blocks in private that exclude the transaction to Bob. These blocks are not part of the *public* longest chain, so no one is initially aware of them.

Bob, seeing the transaction confirmed on the public blockchain, believes the payment is legitimate and hands over the car to Alice.

At this point, Alice reveals her private chain—now two blocks longer than the public one. Because Bitcoin nodes follow the *longest valid chain rule*, the network accepts Alice's version of the blockchain as the new canonical history. As a result, the transaction between Alice and Bob is erased from the blockchain, and Alice gets both the car and her Bitcoin back.

This attack hinges on the assumption that an individual can privately mine a longer chain and later release it to replace the publicly accepted version.
A potential solution to the double spend attack is to reject any future fork that suddenly appears—especially if it overrides the current longest chain and removes previously confirmed transactions. However, this approach introduces several challenges:

- **Susceptibility to timing uncertainty:** It is difficult to determine whether a newly revealed fork is the result of malicious behavior or simply due to network delays.
- Late-joining nodes: Nakamoto wanted the system to work even when nodes leave and join the system at arbitrary times. But a node that joins the system after the new fork arrives cannot

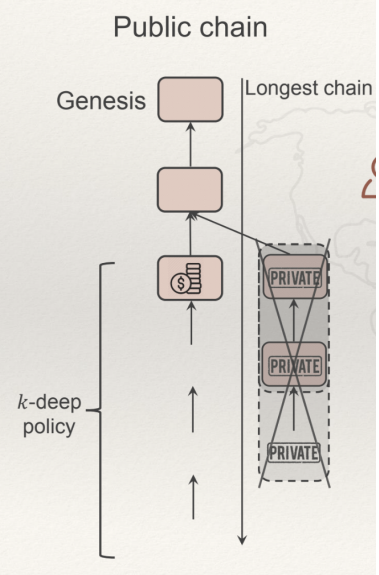


Figure 1: Alice's Private Double-spend attack

tell which fork is new and which fork is old. Even though each block includes a timestamp, adversaries can manipulate them. Thus, timestamps cannot be trusted to resolve disputes about which chain is legitimate.

As a result, the price is **latency**. We do not immediately confirm and instead modify our definition of a transaction.

4 K-deep Transactions

A transaction is only considered confirmed and part of the ledger if it is **at least k blocks deep** in the longest chain. In other words, we take the longest chain and ignore (or "chop off") the most recent $k-1$ blocks. Only the portion before that is treated as the definitive ledger. This approach acknowledges that the tail end of the chain is still vulnerable—transactions near the tip can potentially be invalidated due to chain reorganizations (forks).

However, as illustrated in the earlier example, a determined adversary like Alice could still attempt to build a private chain that overtakes the public chain by at least k blocks, thereby reversing a transaction that was initially considered secure. Nakamoto analyzed this attack scenario and derived the probability that an adversary could succeed, based on the number of confirmations k and the relative mining power of the adversary.

The key insight is that as k increases, the probability of a successful double-spend attack drops exponentially. For example, when $k = 50$, the probability of success becomes extremely low—on the order of 10^{-7}.

Suppose the adversary controls 30% of the total mining power, while the remaining 70% is controlled by honest miners. The following is Nakamoto's table, showing the probability of a successful double-spend as a function of k in this scenario:

$\beta = 30\%$	Adversarial power
k=0	$\varepsilon = 1.0000000$
k=5	$\varepsilon = 0.1773523$
k=10	$\varepsilon = 0.0416605$
k=15	$\varepsilon = 0.0101008$
k=20	$\varepsilon = 0.0024804$
k=25	$\varepsilon = 0.0006132$
k=30	$\varepsilon = 0.0001522$
k=35	$\varepsilon = 0.0000379$
k=40	$\varepsilon = 0.0000095$
k=45	$\varepsilon = 0.0000024$
k=50	$\varepsilon = 0.0000006$

Figure 2: Nakamoto's Table

Effectively, Nakamoto's protocol is:

1. Longest chain to choose the correct fork
2. Choose the transaction that is k-deep into the block

5 Relationship of Epsilon and k

If a log-scale plot of Nakamoto's confirmation probability versus depth k **is linear**, then the probability **decreases exponentially** with some constant c. See the following:

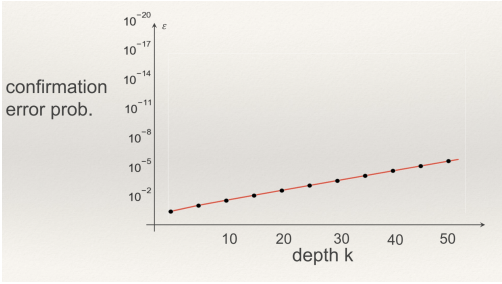


Figure 3: Error vs. Confirmation Depth

6 Nakamoto Race Analysis

To analyze this event, we introduce random variables to model the behavior of the system. By understanding their distribution, we can estimate the probability of the adversary succeeding.

The undesirable event occurs when the adversary is able to mine k blocks before the honest miners do. While the honest miners are working to extend the public longest chain, the adversary is secretly working on an alternate fork in an attempt to overtake the main chain.

In our analysis, we assume that ties are always resolved in favor of the adversary. This is a conservative modeling choice that strengthens our system design: by assuming the worst-case scenario, we ensure that the security guarantees remain valid even under adversarial conditions.

This competition between the honest majority and the adversary attempting to build k blocks in secret is referred to as the **Nakamoto Race Analysis**.

We can define the following time variables:

$$\text{Sum of the time for the adversary: } S_k^a = T_1^a + T_2^a + \ldots + T_k^a$$
$$\text{Sum of the time for the honest: } S_k^h = T_1^h + T_2^h + \ldots + T_k^h$$

where T_i^a and T_i^h are the block inter-arrival times, the time it takes the adversary and honest miners to mine the i-th block from completion of the $(i-1)$-th block.

From this, we can define the event as:

$$\begin{aligned} E &= \text{event that adversary wins the race} \\ &= \{S_k^a \leq S_k^h\} \end{aligned}$$

This means the adversary completes building their k blocks before or at the same time as the honest miners. The probability we are trying to calculate is $Pr[E]$, which represents the likelihood of a successful attack at confirmation depth k. To calculate this, we must think of the mining process, which involves computing a hash value and comparing it to the threshold.

7 Distribution of Mining Time

The distribution of mining time is **geometric.**

$$\begin{aligned} T &= \text{time for miner to get a block} \\ &\sim \text{Geom}(p) \quad p = \Pr[\text{solving the puzzle}] \\ &\text{(unit: \# of hashes)} \\ &\approx \exp\left(\frac{1}{\lambda}\right) \end{aligned}$$

Here,

$$\lambda = \text{rate of mining} \qquad \frac{1}{\lambda} = \text{Expected time to get a block}$$

Due to the extreme difficulty of the mining puzzle (requiring approximately 10^{80} hash attempts), the geometric distribution can be well-approximated by an exponential distribution measured in

time units. As a result, the process of a miner generating blocks follows a Poisson process with exponential interarrival times. This simplifies the analysis. Suppose again that an adversary controls 30% of the mining power while honest miners control 70%. **The ratio of their mining rates can be approximated as** $\lambda_a/\lambda_h = 3/7$.

We can now express the following:

$$T_1^a \sim \exp\left(\frac{1}{\lambda_a}\right), \quad T_1^h \sim \exp\left(\frac{1}{\lambda_h}\right)$$

Here, we have a geometric distribution approximated by an exponential distribution. Each time the miner solves the puzzle, due to the random oracle model, a random number is being generated. As a result, we can say that these are all **independent random variables**.

We can now write this event in terms of the sum:

$$\begin{aligned}\mathbf{Pr}(E) &= \mathbf{Pr}\left(\sum_{i=1}^{k} T_i^h > \sum_{i=1}^{k} T_i^a\right) \\ &= \mathbf{Pr}\left(\sum_{i=1}^{k} (T_i^h - T_i^a) \geq 0\right)\end{aligned}$$

Then, we can compute the expectation of the random variable:

$$E[T_i^h - T_i^a] = \frac{1}{\lambda_h} - \frac{1}{\lambda_a}$$

We see that when $\lambda_h > \lambda_a$, the honest miners have more mining power, while when $\lambda_h < \lambda_a$, the adversary has more mining power. When this is the case, the protocol cannot be secure because the error probability is too high to ensure security.

Additionally, we can apply the law of large numbers (as k increases):

$$\frac{1}{k}\sum_{i=1}^{k}(T_i^h - T_i^a) \to E[T_i^h - T_i^a] < 0 \text{ as } k \to \infty$$

Here, we can see that:

- Since $T_i^h - T_i^a$ are i.i.d. random variables for $i = 1, \ldots, k$, as confirmation depth k increases, the probability of a successful attack decreases exponentially by the Chernoff bound.
- This shows Nakamoto's analysis that security improves exponentially with confirmation depth

References

[1] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography.* 2023.