ECE 595: Foundations of Blockchain Systems **November 23, 2020**

Lecture 15: Proof-of-Stake Attacks

Lecturer: Sreeram Kannan *Scribe: Kelvin Lin*

Outline: So far we have checked the security performance of Ouroboros protocols with the adversarial ratio $\beta < \frac{1}{3}$ and $\beta \geq \frac{1}{2}$. And this lecture firstly shows how to use Nakamoto block to prove security for the interval between $\frac{1}{3}$ and $\frac{1}{2}$. Then this lecture answers following questions:

1. What attacks are possible on the current model of Proof-of-Stake protocols?
2. How do the Proof-of-Stake protocols, namely Ouroboros, defend or minimize these attacks?
3. What is Dynamic Stake? What are possible attacks on the Ouroboros protocols with dynamic stake and how to defend these attacks?

15.1 Proof-of-Stake Security

In our current model of a proof-of-stake protocol, a node is allowed to generate a leadership certificate (LC) if it can solve the hash function:

$$H(pk, t) < th \cdot stake(pk)$$

Previously, we showed that proof-of-stake protocols are secure when the adversarial ratio $\beta < \frac{1}{3}$, but not secure when $\beta \geq \frac{1}{2}$, but how do we solve the security for the interval $\frac{1}{3} \leq \beta < \frac{1}{2}$?

15.1.1 Nakamoto Blocks

To show the security of proof-of-stake protocols when the adversarial ratio is within the interval $\frac{1}{3} \leq \beta < \frac{1}{2}$, we introduce the concept of the Nakamoto block. First, to better understand the Nakamoto block, consider the following scenario as shown in Figure 15.1: given a blockchain tree, we decompose it into a fictitious sequence of honest blocks, denoted as $h_1, h_2, ...h_n$, that arrive at $t = \tau_1, \tau_2, ...\tau_n$ where adversaries build off each honest block.

In this scenario, the honest chain grows with a rate of λ_h, while the adversarial trees grow at a rate of λ_a. Since our hash function is only a function of the public key and timestamp, an adversary is able to work simultaneously on all honest blocks.

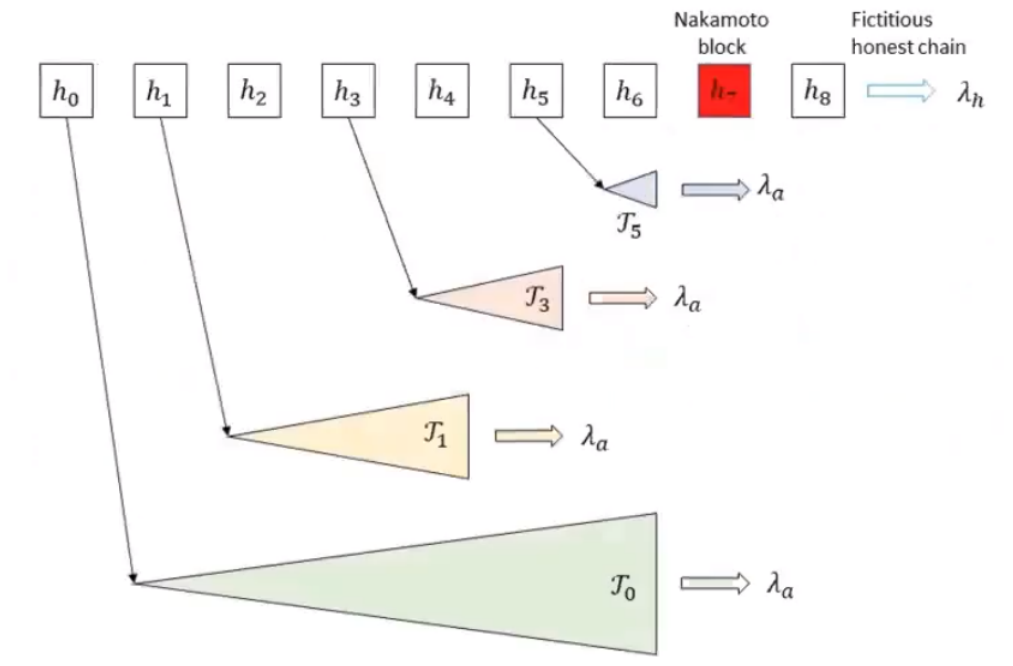

Figure 15.1: Decomposition of a blockchain tree into honest blocks and adversarial trees

We describe a honest block as a Nakamoto block if the rate of growth of the fictitious honest chain exceeds the rate of growth of all adversarial sub-trees. If there is a non-zero chance that a given block is a Nakamoto block, then when a Nakamoto block ”arrives” it can be considered as a new ”genesis” block of the blockchain and everything that occurs prior to the Nakamoto block is considered secure.

Now, how do we guarantee that there is a non-zero chance of a Nakamoto block arrival? The rate of growth of our honest chain equals λ_h and the rate of growth of any adversarial sub-tree is λ_a. As long as $\lambda_h > \lambda_a$, then the growth of the honest chain will always lead the growth of the adversarial chain and we have a non-zero probability that an honest block arrival will be a Nakamoto block. Following this conclusion, given an adversarial ratio $\beta = \frac{\lambda_h}{\lambda_h+\lambda_a}$, the blockchain is secure as long as $\beta < \frac{1}{2}$.

15.1.2 The Predictability Problem

Revisiting our hash to generate a leadership certificate,

$$H(pk, t) < th \cdot stake(pk)$$

In a proof-of-stake blockchain, the public keys are public domain information and so anyone even adversaries are able to find them. With this knowledge, any node can compute when any other node will have a chance at obtaining the leadership certificate. While the adversary cannot generate a node with the public key without signing using the private key, an adversary could utilize this information to either (1) conduct a targeted denial of service attack on nodes that can obtain leadership certificates or (2) bribe other nodes to sign blocks of its choosing.

We will demonstrate how an adversary could conduct the second attack where it bribes other nodes. Consider the scenario where we have a leadership certificate chain as shown in Figure 15.2. An adversary can bribe the nodes at time t = 7, 8, 9, and 10 to generate new blocks of the adversary's choosing. In this attack, the adversary only need to bribe k+1 leaders to have a new longest chain.

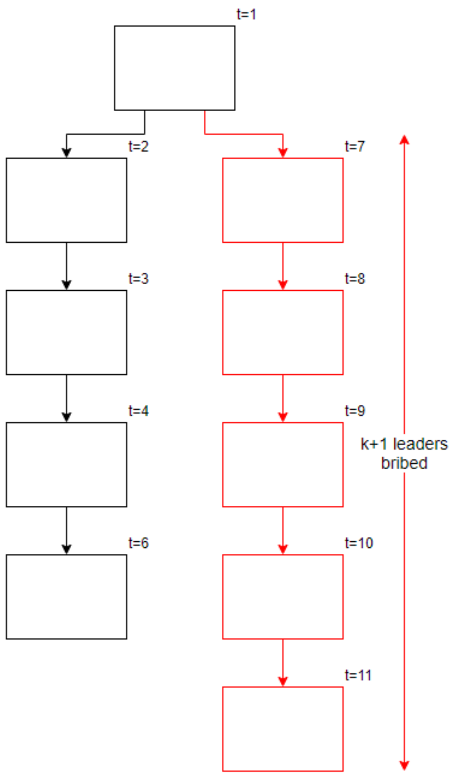

Figure 15.2: Bribery attack where the adversary generates a new longest chain by bribing leaders at time t = 7, 8, 9, 10, and 11.

15.1.3 Verifiable Random Functions

We have this predictability problem because the leadership certificate only depends on the public key. One way to avoid this is to have the leadership certificate depend on the private key which would only be known to the signing node. To do this, we introduce a construct called Verifiable Random Functions (VRF). This solution is implemented in Ouroboros Praos.

The verifiable random function has two different operations: (1) VRF Prove and (2) VRF Verify. The VRF Prove function is defined as

$$VRFProve(x, sk) = (y, \pi)$$

where,

- x : value it found
- sk : secret key
- y : an output like H(x)
- π : the proof that y is correct.

When a leadership certificate is issued, the node owning the leadership certificate also includes the outputs of the VRF Prove function. Other nodes verify this computation by running the VRF Verify function.

$$VRFVerify(x, y, \pi, pk) = true/false$$

In this scheme, the leadership computation is generated by the following equation:

$$LC : VRFProve(t, sk) < th \cdot stake(pk)$$

Now, nodes can not longer compute when other nodes will have the leadership certificate, but they can still compute it for themselves. With the new scheme, the attack described in Figure 15.2 can no longer rely on knowing who the leaders will be at a given time. Instead, an adversary could carry out a similar attack where it bribes the nodes to generate alternate leadership certificates (post-facto corruption).

15.1.4 Key Evolving Signatures

To avoid this post-facto corruption problem, we introduce another construct called Key Evolving Signatures (KES). In key evolving signatures, when a node signs a block, it "evolves" its public-secret key pair forward and forgets the previous pair. The mechanism for this evolution can be thought of as like taking the hash–computing the forward operation is very easy, but computing the reverse operation is very difficult.

$$(pk, sk) \xrightarrow[H(pk)]{Evolve} (pk_1, sk_1)$$

15.2 Dynamic Stake

So far, our analysis of proof-of-stake systems has only be with static stake where the stake of each public key is determined at the birth of the blockchain. In a dynamic stake system, the stake associated with a public key can change over time. Using the hash formula for the leadership certificate for simplicity, we have:

$$H(t, pk) < th \cdot stake_{l-1}(pk)$$

where l is the level in the blockchain.

Given dynamic stake, an adversary can conduct a key grinding attack. In this attack, an adversary can compute the hash for all the public keys in their possession to find the public keys that have the highest chance of winning the leadership certificate and invest more stake into those keys.

One idea to avoid this attack is to introduce a time lag between when an adversary invests into a public key and when the stake of the key increases. However, this only changes the window in which the adversary can conduct this attack. Suppose this delay is s, then the adversary is still able to conduct a key grinding attack after s blocks in the future.

15.2.1 Ouroboros Genesis

Suppose we have a delay of s blocks before the stake of public keys are updated. Now suppose that we use s-truncated longest chain instead of longest chain to determine which branch to take. In s-truncated longest chain, when a fork is encountered, both branches are truncated to s seconds and the longest chain after the truncation is chosen. In order for an adversary to conduct an attack in an s-truncated longest chain, it would have to compete fairly against the honest nodes within the window of s time. And so, we are able to avoid the key grinding attack.

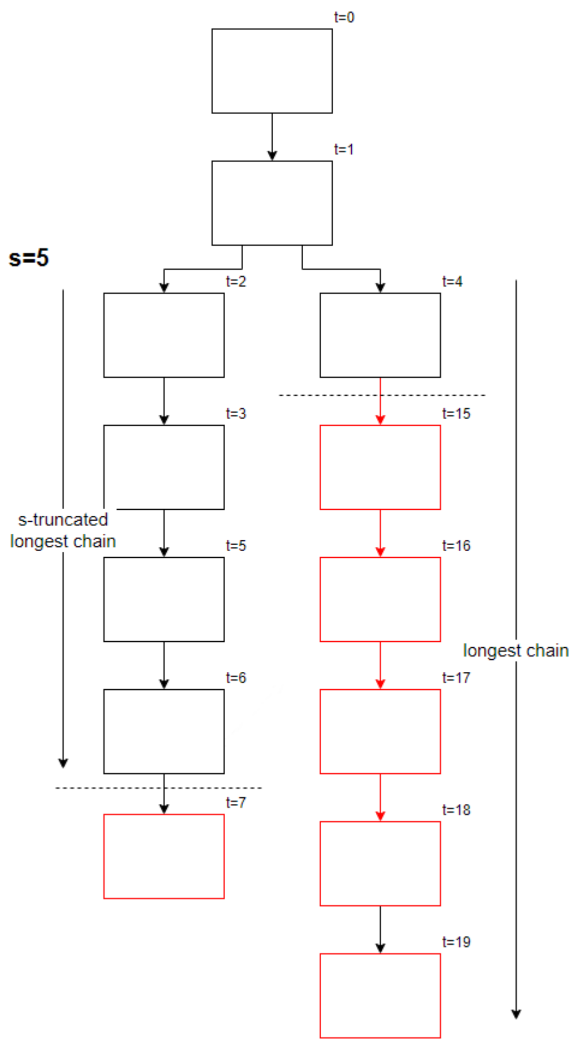

Figure 15.3: Comparison of longest chain vs. s-truncated longest chain. s = 5.