EE 374: Scaling Blockchains **Stanford University, Spring 2020**

Lecture 18: Bribery and stake grinding attacks

04-June-2020

Lecturer: David Tse Scribe: Ashwin Sekhari

Update: This is the last lecture, and next week there will be student presentations. The presentation schedule is uploaded on the course website.

1 Introduction

This lecture finishes our discussion about Proof of Stake (PoS), PoS is a complex problem because changing from Proof of Work (PoW) to PoS opens up the avenue for many attacks. In the beginning, we talked about Nothing at Stake (NaS) attack and showed that it could be solved to some extent, and by a more careful security analysis, we can confirm that we get the security of threshold as $\frac{1}{1+e}$. To prevent an adversary from gaining an unfair advantage by grinding on the randomness of the block, we introduced correlation among the randomness of blocks where c defines the degree of correlation, and by increasing this c correlation, our security threshold approaches 0.5. Still, we are worried that this randomness correlation will open the door to another type of attack, which is a prediction attack i.e., a third party can predict who wins in advance. Having high correlation results in a large prediction window, and a third party can easily simulate who wins the lottery and at what time slot, using which an adversary can selectively go and bribe the nodes to help him make a double-spend attack.

The way we solved this in the last lecture is by using a cryptographic device called Verifiable Random Functions (VRF) [4], which allows the miners/stakeholders to generate the randomness such that no one can generate it other than the stakeholder itself using its private key. However, it can be verified by the public that the generated randomness is correct. Now, we come back to the protocol to see whether the VRF solves the problem, or there is some residual leftover issues. Finally, we will look at the stake grinding attack.

2 Back to $c = 0$

We are back to $c = 0$, where the randomness updates every block. To prevent a third party from predicting the lottery outcome, instead of using a regular hash function, we use $\text{VRFHash}(R_0, t_0, sk)$, his secret key in place of the public key, and the block producer provides a proof

$$\text{VRFProof}(R_0, t_0, sk) \to \pi$$

which can be verified by $\text{VRFVerify}(R_0, t_0, \text{VRFHash}, \pi, pk)$. In terms of prediction aspect, we are starting at the genesis block; an adversary tries to predict who is going to win the slots in the future so that he can selectively bribe people. Unlike before adversary cannot figure out who wins and when because he does not know the secret keys of the stakeholders. Therefore, he is at a total loss. Is there anybody who can predict who wins?

VRFHash can only be computed by the private key. However, let's think about it from the point of view of the stakeholder. He can predict at what particular time slot he is going to win. He may not be the next block winner, but now he has some information. Because, for example, if he knew that he is the winner on the first time slot. Then he knows that with a very high probability, he will get the next block. Therefore although VRF hides information from another party, it still does not solve the issue of having the information leaked to someone, in this particular case, the stakeholder. Each stakeholder has some estimate on the probability that he is going to win! Note that this is not the case in PoW: even the eventual winner himself does not have any indication that he will more or less likely to be the winner beforehand.

However, this information leakage is relatively small, and also this window that he knows information is short. For example, if one block is generated every 10 seconds, an individual stakeholder has almost no information on who is going to win beyond the time scale of 10 seconds because it is hard to predict which stakeholder is going to win the block after next. This event depends on the randomness of the next block, and that in turn depends on who is going to win the next block.

In contrast, in c-correlation protocol, the larger the c is, the more information the stakeholders have and the more time they have to predict which time slots they will win. They can now predict c blocks ahead.

3 Prediction bribery attack

Ouroboros Praos[3] is a PoS protocol similar to the c-correlation protocol introduced, with c very large, roughly like five days. Ouroboros protocol did in fact, include VRF in their proposal. The adversary, as a third party, still cannot predict. There is a genesis block having randomness R_0, and throughout epoch (5 days), the randomness stays the same. Then everybody knows which time slot they are going to win. Time slot for a miner i is determined by

$$\text{VRFHash}(R_0, t_i, sk_i) < threshold$$

A third party cannot predict who wins because it does not know the secret key, but the node itself can predict within these five days at what time slot he is going to win the lottery. When the randomness was updating every block, i.e., $c = 0$, our prediction window was small around 10 seconds because of which it was not a big deal. However, when we have a large correlation among the blocks' randomness, our prediction window increases to 5 days as a result of which the nodes have ample time to predict their winning slots. Nodes can be selfish. These selfish nodes know in these five days, which time slot in the future they are going to win. An adversary can publish a website to request nodes for their winning timeslots along with a verifiable proof to take advantage of this information. He will then go and bribe nodes to help him build an alternate chain to perform a double-spend attack.

This is a pretty dangerous attack, in order to bribe he will have to pay the nodes more than the reward. However, the crime that he is attempting is double spend which can earn him a lot. For a successful attack, the adversary needs to bribe at least k nodes because the confirmation rule that we are using is k-deep, and for anything more than k, he can produce a longer chain and reverse a transaction This is a PoS system, so k nodes are only a tiny subset of the stakeholders, the total nodes in the network can be a million, but k can be around 20-30 stakeholders.

3.1 NaS-Prediction Tradeoff?

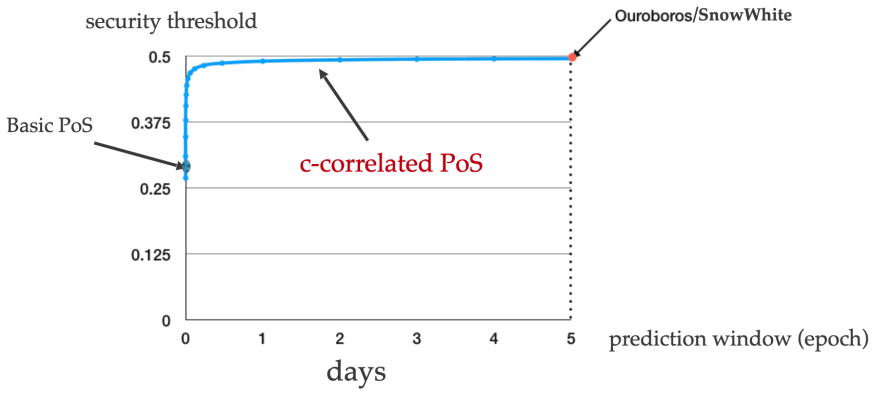

Figure 1: The graph represents security threshold as a function of prediction window; when we decrease the prediction window, security threshold decreases because of Nothing at Stake attacks, but when we increase the prediction window by introducing more correlation, the security threshold approaches 0.5, but we are at risk of bribery attacks.

The current situation is represented as a graph in figure 1; there is a tradeoff between the predictability, which enables bribing and the NaS attack. If we choose the prediction window to be very short, the threshold drops to 0.27, i.e. (1/1+e). As we increase c, the security threshold approaches 0.5, which is good, but we are also increasing the prediction window, which makes it prone to bribery attacks. There is a question mark here because we only analyzed this one particular family of protocol, this c correlation protocol. Ouroboros and Snow White [2] are one extreme, and PoS with randomness update every block is another extreme. At this moment, we do not know whether there is another family of protocol that does not have this tradeoff, i.e., we can get to 0.5 as the security threshold without having to open up to prediction attacks.

4 Block Content

All this time, we have been thinking about blocks $R_0, R_1, ...$, the whole reason we want to have a blockchain is not just to win the lotteries. The goal of blockchain is to carry information and to carry transactions because we want to update and maintain the ledger. We can think of $R_0, R_1, ...$ as the block headers, and there should also be content (genesis block may have no content) in the blocks (figure 2). The content C_i is the sequence of transactions denoted by $\mathsf{Tx_i}$ in the i^{th} block. Despite the transactions being signed by the stakeholder, we cannot put them directly in the block because an adversary can replace an old block with a new block with the same header, but different contents, which violates consistency. To prevent this from happening, we require the winner to sign the transaction set. Signing the transaction requires the secret key $C_i = \text{sign}(\mathsf{Tx_i}, \mathsf{sk_i})$ of the stakeholder that mined the block. Using the public key, $\text{verify}(C_i, pk_i)$, people can verify the content. As a result of which a third party cannot forge the signatures and change the contents afterward. Are there any other possible attacks?

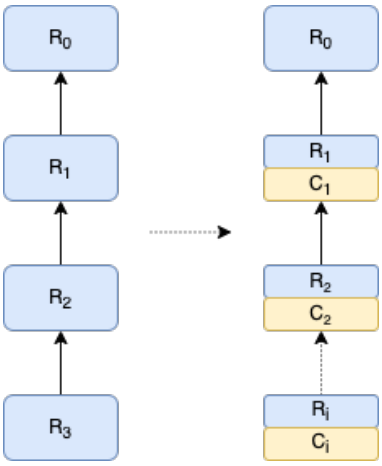

Figure 2: Block Content

Let us compare it with PoW. In PoW the transactions are sealed by the hash inequality,

$$\text{Hash}(R_{i-1}, \mathsf{Tx_i}, nonce) < threshold$$

As a result, we cannot just replace $\mathsf{Tx_i}$ with $\widetilde{\mathsf{Tx_i}}$ because it will not satisfy the PoW puzzle anymore and will be considered invalid. In our case, the hash inequality does not secure the transactions. Therefore there is plenty of time for an adversary to bribe the node and make him change the contents, sign it, and generate the block again. Other miners cannot reject the block because it contains valid signatures. Moreover, any minor who is the winner of a block can go back later on and publish another block with a different set of content, signed by itself. How can an adversary use this to his advantage?

The most severe offense that an adversary can commit on a blockchain is double-spending. We have a sequence of blocks, and we are k-deep in it. Bribing immediately is not what an adversary is going to do; he always wants to bribe after the confirmation happens bribing immediately is useless. Now the adversary can go back and declare that whoever won that block can claim a prize provided he is willing to re-sign his transactions. Let us define two types of attacks:-

- **Post-bribery attack:** We are attacking/bribing the system after the block is generated.
- **Pre-bribery attack:** We are bribing people even before the blocks get generated because we know ahead of the time who is going to win.

Now, a pre-bribery attack is more severe than a post-bribery attack because this guy is publishing a second block, i.e., he is double signing two different contents using his signature. Pre-bribery attacks are not attributable because a third party knows in advance who is going to win, and using which he can bribe the nodes to change the contents before publishing the block. As a result of which there is no trace in the network of any malpractice, and therefore it cannot be attributed to any identity. Whereas, because of double signing, post-bribery attacks are attributable; you can attribute the attack to an identity. With the identity, you can do a crime, but also with the identity, I can try to catch you. In this case, it is trouble because we know a particular node with a particular identity has a double sign. However, we can still try to make this a little more secure,

and the way to do this is by creating a chain, Nakamoto always teaches us that creating a chain is a good thing, right now we have a chain, but the chain is only connecting the header information. It does not connect the transactions so we can create a separate chain or a virtually separate chain of the transactions through the signing process,

$$C_i = \text{sign}(C_{i-1}, \mathsf{Tx_i}, sk_i)$$

So then it is not so simple to replace the content as changing the content of a block means we will have to change the content of all the next blocks; otherwise, the verification, i.e., $\text{verify}(C_i, \mathsf{Tx_i}, pk_i)$ will fail. With this provision, the attacker will have to bribe all the k nodes, which is hard.

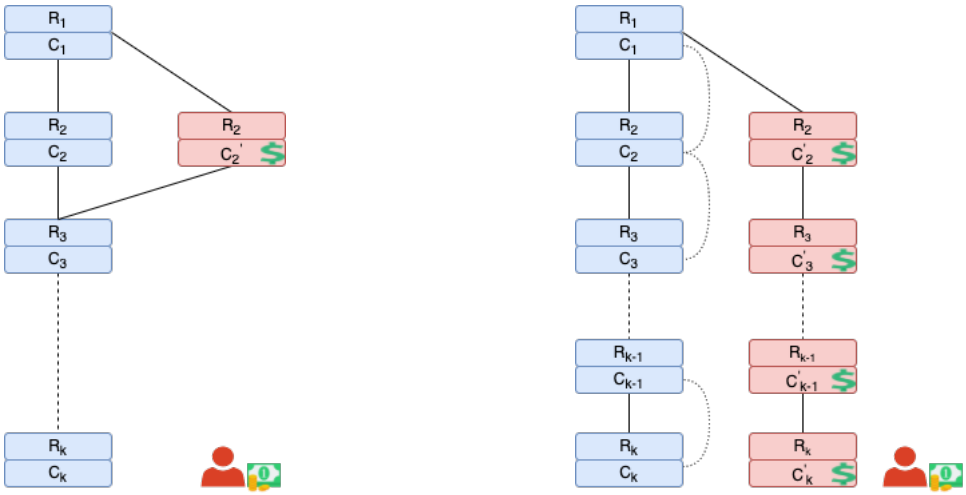

Figure 3: (left) The adversary only has to bribe one stakeholder to perform a double-spend attack. (right) The block contents form a virtual chain because of which the adversary has to bribe all subsequent k blocks to deliver a successful attack.

5 Stake Grinding

Almost all the problems that arise in PoS design is because of the opportunity to do grinding. In other words, an attacker can try to play around with the input to the lottery in order to gain an unfair advantage over an honest node. PoW avoids that problem by putting the grinding explicitly into the competition; whoever grinds fastest will win, and that costs the use of resources. In PoS, we do not want to use physical resources, and we need to be content and therefore need to deal with grinding at a protocol level. In Nothing at Stake, we are talking about randomness grinding, and we are trying out different randomness of different blocks to see where we can mine a faster chain. This can be reduced by introducing correlation, but that brings us the prediction attacks, and we are stuck at a trade-off there. There is another source of randomness which we have not discussed yet, which is the secret key of the stakeholder.

In PoS systems, the hash input,

$$\text{VRFHash}(R_{i-1}, t_i, sk_i) < threshold$$

uses randomness of the previous block R_{i-1}, the time t_i, and secret key of the node. Therefore, NaS is grinding this R_{i-1} by trying out different blocks, and stake grinding is trying out different

secret keys. One main thing we want to do with a blockchain is to maintain the ledger, the set of all the transactions that have been committed to this blockchain. There is a population of users $(pk_1, pk_2, ..., pk_{1000000})$, defined by the public key, which is their identity. These are the set of coins out there and associated with each of these public keys; there is a secret key. This set of coins is the ledger state, which essentially tells us about stake ownership.

A transaction is nothing but a transfer from one identity to another identity, one public key to another public key or a pair to another pair because behind a public key; there is always a secret key. As a result, the old pair disappears from the system, and the new pair appears in the system ledger state. Therefore, grinding on the secret key requires changing the ledger state. There is a little bit of chicken-egg problem here. As the blockchain evolves, we have $state_0, state_1, state_2, ...$, the state evolves, a state is nothing but the stake distribution, keys in the system, and as we execute a transaction, the identity of the coin, secret key and public key, changes.

Nevertheless, the keys themselves are the ones who determine who is going to win next. The chicken-egg problem arises because if we are the winner of a block, we can decide what content goes in that block. An adversary can create a new key pair of the coin that he already has, and he can transfer it to himself. Moreover, using this new coin, he can try whether his chance of winning the next block is higher as the next block's winning is based on the randomness of his generating block. He knows the lottery of the next block, and unlike everyone in the system, he can control the state's evolution. He can transfer the same coin to himself repeatedly until he finds the winning secret key. Initially, he has a 1 in a million chance of winning, but by allowing himself to transfer the coin once, he can double his identity to become effectively two coin holders from the point of view of winning the lottery, and make his probable winnings double.

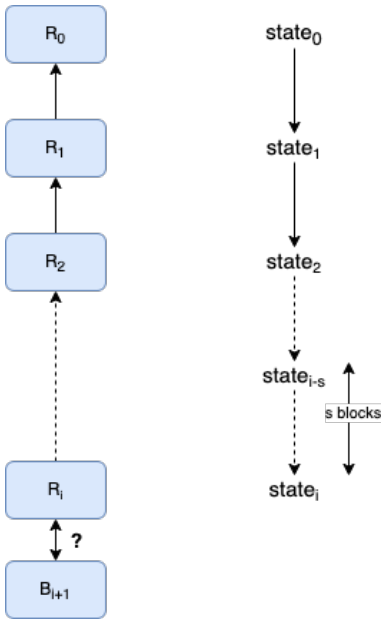

Figure 4: Whenever a new block is mined, it incorporates transactions that bring about a change in the system's stake distribution, and as the stake distribution changes, the state evolves. For mining the block B_{i+1}, we consider the stake distribution as defined by $state_{i-s}$.

One way to prevent stake grinding attacks is to use old state (stake distribution): instead of using the most recent stake distribution ($state_i$), we will use the stake distribution described by the block s blocks behind ($state_{i-s}$), and only the stakeholders of the coins in $state_{i-s}$ can participate in the lottery. This is a solution, and there are some drawbacks here. When you use s very large, you are using an outdated stake; the whole point of PoS is that you want people who have the stake in the system to have a voice. It is like saying, Can I vote on a company that ten years ago I had shares in, but now I do not, this is unavoidable in a proof of stake system, we have to do something like this, but we want to keep s kind of minimum. Let us try to understand how small s can be. To answer this question, we need to first understand that even this protocol has another attack.

How can the attacker bias the selection of block B_{i+1}(figure 4)? He needs to affect whatever determines the lottery i.e, $state_{i-s}$. He can go back and say I need to change the state, this is a game similar to post-bribery attack you go back and change the content, and if he was the block miner, he could easily change its content. However, he will also have to change all the blocks following it to make it consistent because the content is linked through signatures; we used the hash of previous blocks' content. So he needs to be able to control all s blocks, which is hard when s is very large. But is it really that hard? Here is an attack (figure 5). We have a public longest chain which is maintaining a ledger, and the adversary is growing a Private chain to attack, it tries to mine the block, but it is hard for him because he has less than 5% of the stake. Since the system is live eventually, he will get a block, but it is taking a lot of time. Because he is using the state s blocks behind, he is not controlling the state, but once he gets s blocks, he is the king! Now he can create his world his own stake; as he can now go back and change the content of the blocks such that in the next lottery his chance of winning is very high, he will grind the keys which are s blocks behind and does not have to compete with anyone else. After successful grinding, he will also have to re-sign all the subsequent s-blocks' transactions to maintain consistency. So initially, the chain grows very slowly because he has $< 5\%$ of the stake, but suddenly, after s blocks, it becomes really fast and will eventually catch up with the public longest chain.

This attack is not attributable because this was done in private, unlike bribery, which was in public; no one knows that this guy is grinding all his content to have a high chance of winning. So what you see in the chain will be indeed many blocks infrequently, and suddenly you get a quick succession of blocks. Moreover, when we overtake the longest chain, we know that the double-spend attack has succeeded. How can we prevent this attack?

We can think of the longest chain rule as a way to order the chains according to their length. In this example (figure 6), chain C_2 is longer than chain C_1, semantics is that C_2 has more stakes and thus it should win. However, it could also be possible that many blocks generated at the end are due to stake grinding, and that is unfair i.e., it is not a reflection of the fact that C_2 has a larger stake than C_1, it is a reflection of him cheating, and we need to prevent that. To prevent that, we will use a better fork choice rule called the s-truncation rule.

s-truncation rule: Whenever a node receives conflicting chains instead of comparing the length of those chains directly, as in the Nakamoto's protocol, we will not count the extra length of the chain and chop anything after s. When comparing two chains, we only apply this truncation when at least one of them has more than s blocks after forking. If both of the chains have less than s blocks after forking, we use the longest chain rule to determine which chain to mine. That is, we will not count more than s because it is not reliable anymore due to stake grinding. The new

lengths after this truncation effect are given by,

$$\tilde{l_i} = l_0 + \min(l_i - l_0, s)$$

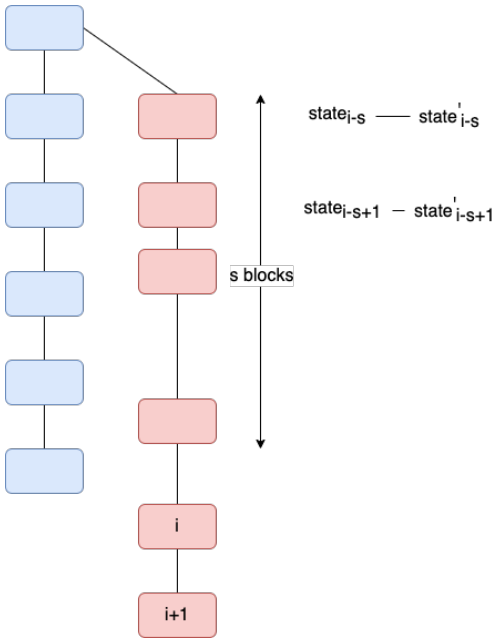

Figure 5: Honest users mine the blue chain, and the adversary is building the red chain in private. Initially, the growth of the chain is slow, but as soon as the chain reaches s blocks, the growth becomes very fast because he controls the chain and can, therefore quickly go back to change the content of the blocks. To mine i^{th} block, we have to consider the stake defined by the state of $(i-s)^{th}$ block.

Instead of using the longest chain rule directly for ordering the chains, we will make the comparison based on their truncated lengths, i.e., $\tilde{l_1}$ and $\tilde{l_2}$ instead of l_1 and l_2.

This rule has a drawback, s is still unspecified, and ideally, we want s to be as close to 0 as possible. Suppose we have s very small, we have a k-deep confirmation rule to confirm blocks, and we are going to wait until we are k blocks deep, but those blocks that come after s are not useful anymore because they do not increase security. So if we choose s very small, the k deep rule will not be effective, and it cannot give you reliability. Therefore, s should be chosen larger than k,

$$s \geq k$$

Intuitively, the condition $s \geq k$ means that we are only using the stake distribution that has been confirmed; it cannot be messed around by the attacker. For s larger than k, we can provide security with reversal property exponential in k. If s is less than k, we can only provide error probability, which is exponential in s, so we are limited by s. If we want a particular error probability, we have to choose s sufficiently large enough. As a result,

$$\text{Prob}(reversal) \sim \exp(-\min(s, k))$$

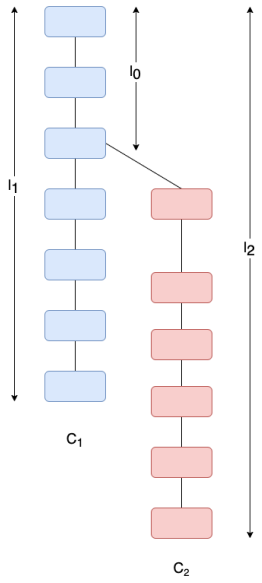

Figure 6: The adversary is mining on his private chain C_2, which is longer than the public chain C_1. If we prefer the chains according to Nakamoto's k-deep rule, C_2 should win. In PoS, unlike PoW, stake grinding could happen as a result of which the length of the chain is not a pure reflection of the stake in the system.

We can think that the longest chain rule is setting $s = \infty$, so s-truncation rule with $s = \infty$ we get the longest chain rule, and your error probability is exponential in k. If you choose s to be less than k, then error probability is limited by s because s, the duration of old stake states that even if you grow a very long chain it is not useful for us anymore. It does not provide you with further reliability and support for the vote. Something similar to the s-truncation rule was invented in Ouroboros Genesis [1]. There are other rules, but they are pretty similar: basically, the chains that are too long are not reliable anymore. So you can measure the length of chains in different ways, you can measure in time like you can use a timestamp. For example, you can say that you can only use the blocks that arrived within a specific time.

References

[1] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 913–930, New York, NY, USA, 2018. Association for Computing Machinery.

[2] P. Daian, R. Pass, and E. Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security*, pages 23–41. Springer, 2019.

[3] B. David, P. Ga***v***zi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[4] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.