EE 374: Blockchain Foundations **Stanford, Spring 2022**

Lecture 17: VRF, Streamlet

May 29th, 2022

Lecturer: Prof. David Tse Scribe: Sergio Charles and Neetish Sharma

1 Proof-of-Stake Recap

Recall that in Proof-of-Stake (PoS), each staker $i \in \{1, \ldots, n\}$ will attempt to satisfy the PoS equation:

$$H(s_0||pk_i||r) < T_p, \tag{1}$$

where $s_0 = H(G)$ is the hash of the genesis block, pk_i is the public key of the i-th staker, r is the round, and T_p is the threshold. The staker that satisfies Equation (1) effectively wins the "lottery." Each block $B = (s||x||pk_i||r||\sigma)$ consists of the hash s of the previous block, the Merkle root x of the list of transactions, the public key pk_i of the staker who won the lottery, the round r and the signature σ of the staker signing the block.

2 Verifiable Random Function

2.1 Motivation

There is a glaring problem with the current definition of PoS, namely the public key is visible and known to everyone. Thus, anyone can predict from the beginning of the execution who will win the lottery. Therefore, bad actors can determine specific individuals to bribe to help build an alternate chain. Thus, they can effectively mount an attack by bribing a small fraction of potential stakers who win the lottery. In our previous security analysis, we ignored this by assuming that the adversary is static.

In contrast, an adaptive adversary can observe the execution of the protocol and adaptively select stakers to attack. In what follows, we modify the PoS equation and block data structure so we can not easily predict the public keys. That is, we want a way of running the lottery using the secret key, instead of using the public key–which is publicly known information. However, stakers cannot use their secret keys in clear. Fundamentally, the idea is that every staker evaluates based on the secret key, obtains a value, and determines whether it is below or above a threshold. If it is below the threshold, it can send proof out that it is the winner without sending the private key.

2.2 Definition

To realize this goal, we introduce the following cryptographic primative–Verifiable Random Function (VRF) [3]:

Algorithm 1 Verifiable Random Function

$(sk, pk) \leftarrow \text{Gen}(1^\kappa)$
$(\nu, \pi) \leftarrow \text{Eval}(sk, u)$
$\text{Ver}(pk, u, v, \pi) \in 0, 1$

We want to introduce unpredictability so we use the secret key in the evaluation. This is random in the sense that the evaluation output ν is not predictable, i.e. we cannot tell if it is evaluated by a particular staker or just a randomly drawn number. Here, ν is the analog of the hash function output in the previous PoS equation. We compute this when we want to determine the winner of the lottery. The π is used by the verifier to tell us whether this ν output is valid. The verifier has no access to the staker's secret key sk, so it needs ν to verify the evaluation was done properly, i.e. the the particular staker has actually won the lottery.

2.3 New PoS Equation

To verify the legitimacy of a block, we define a new PoS equation. The evaluation function allows the lottery staker to verify whether or not he will win the round. The input of the evaluation function u is concatenation of s_0 and r so the new PoS equation is:

$$\text{Eval}(sk, u).\nu < T_p, \tag{2}$$

where $u = s_0||r$. Furthermore, we want to replace the predictable block, defined above, with a new block structure, defined as $B = (s, x, pk, \nu, \pi, r, \sigma)$ where s is the hash of the previous block, x is the Merkle root of the transactions list, pk is the public key of the staker, ν is the VRF evaluation output, π is the VRF proof, r is the round, and σ is the signature.

Upon receiving a block, the verifiers check that $\text{Ver}(pk, u, v, \sigma) = 1$.

2.4 Correctness and Security

We need to guarantee correctness and security. For correctness, we must do so for every input u. Correctness means that anyone who is doing an honest evaluation should be permissible. Using the VRF, we check if the verification function returns 1. Thus, we verify using the public key corresponding to the secret key–if we claim the output is ν with proof π–evaluated on the input:

Definition 2.1 (Correctness)**.** A VRF is called *correct* if for all inputs u and keys $(sk, pk) \leftarrow \text{Gen}(1^\kappa)$, we have

$$\text{Ver}(pk, u, \text{Eval}(sk, u)) = 1\,.$$

Correctness alone does not suffice. Namely, we also need to guarantee unpredictability so the attacker cannot predict the VRF and corrupt the future leader.

Algorithm 2 Unpredictability of VRF Game

function $\mathsf{Predict}_{\mathcal{A}}(\kappa)$
  $(sk, pk) \leftarrow \mathrm{Gen}(1^{\kappa})$
  function $\mathcal{O}(u)$
    $M \leftarrow M \cup \{u\}$
    return $\mathrm{Eval}(sk, u)$
  end function
  $u, \mathrm{aux} \leftarrow \mathcal{A}_1^{\mathcal{O}}(pk)$ ▷ aux is the additional information that $\mathcal{A}_1$ passes to $\mathcal{A}_2$
  $b \xleftarrow{\$} \{0,1\}$ ▷ Challenger generates a random bit
  if $b = 0$ **then**
    $\nu, \pi \leftarrow \mathrm{Eval}(sk, u)$
  else
    $\nu \xleftarrow{\$} \{0,1\}^{\kappa}$
  end if
  $\hat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}}(pk, \nu, u, \mathrm{aux})$
  return $b = \hat{b} \wedge u \notin M$
end function

The adversary picks an input u to try to break the VRF given the public key pk. The challenger first generates a binary digit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then it will do an evaluation based on the VRF $\mathrm{Eval}(sk, u)$. Otherwise, if $b = 1$ then it will generate a κ-bit random number. The challenger tasks the adversary with the challenge of distinguishing if the staker did the evaluation or if it's a random number. In one world, the challenger really did the evaluation; in the other world, it is a completely random number. We want to do this because, if the adversary could tell, then she could predict and, therefore, distinguish the VRF output from a random number. We want the adversary to not be able to discriminate between these two.

The adversary is forced to come up with an estimate $\hat{b}$ with $\mathcal{A}_2^{\mathcal{O}}$ given pk, ν, u. The challenger then returns whether $b = \hat{b}$. Note that the adversary actually has more information; namely, she sees the output of previous evaluations of the VRF. Thus, there is a "leakage of information" and we want to make sure that leaking of information does not change the chance of winning the game. Hence, we give an oracle $\mathcal{O}$ to the adversary. The oracle gives the adversary the evaluation on any input she desires, except for the challenge input. Another important point to note is that the adversary $\mathcal{A}^{\mathcal{O}}$ is written by two programs, $\mathcal{A}_1^{\mathcal{O}}$ and $\mathcal{A}_2^{\mathcal{O}}$. Therefore, we also allow $\mathcal{A}_1^{\mathcal{O}}$ to pass auxiliary information aux to $\mathcal{A}_2^{\mathcal{O}}$. We define the *unpredictability* of the VRF as follows:

Definition 2.2 (VRF Unpredictability)**.** A VRF is called *unpredictable* if for all pairs of oraclized PPT algorithms $\mathcal{A} = (\mathcal{A}_1^?, \mathcal{A}_2^?)$ we have

$$\Pr[\mathsf{Predict}_{\mathcal{A}}(\kappa)] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

The second part of the security definition is the *uniqueness* of the VRF, which is defined in the following game:

Algorithm 3 Uniqueness of VRF Game

function $\mathsf{Unique}_{\mathcal{A}}(\kappa)$
$(u, v_1, v_2, \pi_1, \pi_2, pk) \leftarrow \mathcal{A}(1^\kappa))$
return $\mathrm{Ver}(pk, u, v_1, \pi_1) \wedge \mathrm{Ver}(pk, u, v_2, \pi_2) \wedge v_1 \neq v_2$
end function

The above ensures the adversary is not able to produce two different outputs that both verify for the same input of a particular VRF public key. Similarly, we can now define the *uniqueness* property of the VRF:

Definition 2.3 (VRF Uniqueness)**.** A VRF is called *unique* if for all PPT algorithms $\mathcal{A}$ we have

$$\Pr[\mathsf{Unique}_{\mathcal{A}}(\kappa)] \leq \mathrm{negl}(\kappa)$$

Security of a VRF is defined as being both *unpredictable* and *unique*:

Definition 2.4 (VRF Security)**.** A VRF is called *secure* if it is both *unpredictable* and *unique*.

3 Streamlet

3.1 BFT vs longest chain protocols

So far we have been focusing on PoW and PoS longest chain protocols. Such protocols, as we showed, use the k-confirmation rule and suffer from long confirmation latency - $O(k)$. But is it possible to have blockchains with fast block confirmation?

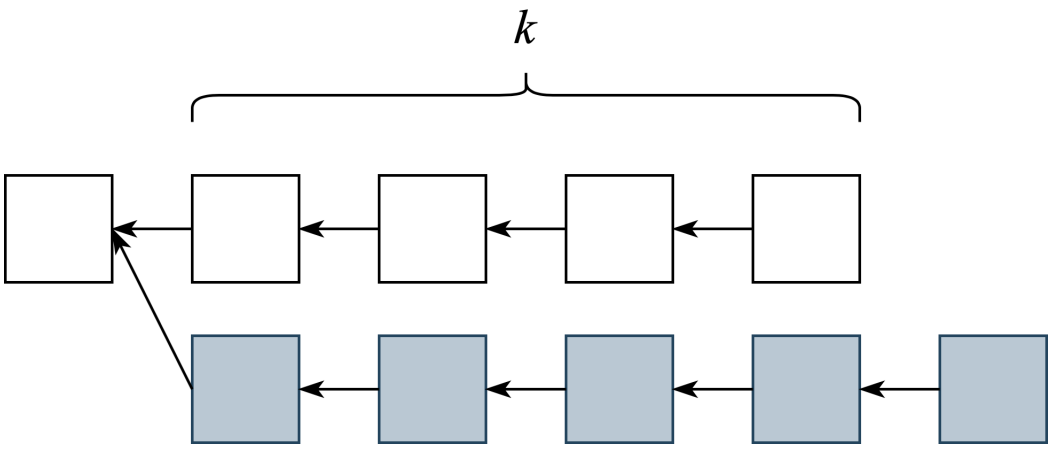

Figure 1: k-confirmation rule: if we choose k too small, the adversary can cause safety violations.

BFT (Byzantine Fault Tolerant) protocols achieve fast confirmation. In such protocols, safety and liveness is guaranteed as long as no more than t out of n nodes are malicious. In BFT protocols, *every* participant votes, while in the longest chain protocols voters are *sampled*. Usually, BFT protocols are pretty complex so in this lecture we will focus on a simple example of such protocols - Streamlet.

3.2 Streamlet Protocol

1. The system consists of n nodes, their public keys $pk_1, pk_2, \ldots, pk_n$ are fixed and known to every protocol participant.
2. The network is synchronized with a delay bounded by Δ. Time is divided into epochs of length 2Δ.
3. Every epoch has a leader L, which is randomly selected via hash function $H(e) \to \{1 \ldots n\}$.
4. Leader L proposes a block B extending the longest *notarized* chain. Initially, only the genesis block is notarized.
5. Each node votes on the first valid block it receives by sending a signed message. A block is considered valid if it is proposed in the current epoch by the epoch leader and extends the longest notarized chain.
6. Notarized blocks are defined as those blocks that contain at least $\frac{2}{3}n$ votes.

We defined what it means for a Streamlet block to be notarized but what is a *confirmed* block? Let us try to design the confirmation rule. To do so, we will first look into a simple strawman confirmation rule and see if such protocol is secure.

3.3 Streamlet strawman confirmation rule

Let us consider every notarized block as confirmed. Can adversary attack such a protocol? She can cause a safety violation if she manages to produce two notarized blocks in the same epoch.

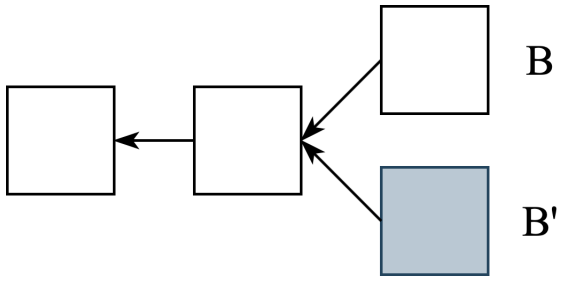

Figure 2: Safety violation in streamlet with strawman confirmation rule

Let us assume that both blocks B and B' illustrated in Figure 2 are notarized. Additionally, assume that the adversary controls less than $\frac{1}{3}n$ nodes. As it was stated earlier in the protocol description, every notarized block has at least $\frac{2}{3}n$ votes. Therefore, by the quorum intersection argument illustrated in 3, at least $\frac{1}{3}n$ nodes voted on both blocks B and B'. Since adversary controls $< \frac{1}{3}n$ nodes, there must exist at least one honest node that voted for two blocks in the same epoch which is not possible. Thus, the adversary cannot cause a violation this way. Is there another attack vector? We will explore it in the next lecture.

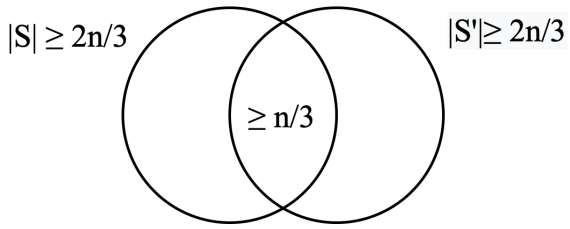

Figure 3: Venn diagram for the quorum intersection argument: let S and S' denote the sets of nodes that voted for the blocks B and B', respectively. By the definition on notarized block, $|S| \geq \frac{2}{3}n$ and $|S'| \geq \frac{2}{3}n$, so $|S \cap S'| \geq \frac{n}{3}$.

References

[1] B. Y. Chan and E. Shi. Streamlet: Textbook streamlined blockchains. In *AFT*, pages 1–11. ACM, 2020.

[2] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[3] 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.