Post-Quantum Verifiable Random Function from Symmetric Primitives in PoS Blockchain

Maxime Buser[1], Rafael Dowsley[1], Muhammed F. Esgin[1,2], Shabnam Kasra Kermanshahi[3], Veronika Kuchta[4], Joseph K. Liu[1], Raphaël C.-W. Phan[5,1], Zhenfei Zhang[6]

[1] Monash University, Australia
[2] CSIRO's Data61, Australia
[3] RMIT University, Australia
[4] University of Queensland, Australia
[5] Monash University, Malaysia
[6] Ethereum Foundation, Australia

Abstract. Verifiable Random Functions (VRFs) play a key role in Proof-of-Stake blockchains such as Algorand to achieve highly scalable consensus, but currently deployed VRFs lack post-quantum security, which is crucial for future-readiness of blockchain systems. This work presents the first quantum-safe VRF scheme based on symmetric primitives. Our main proposal is a practical many-time quantum-safe VRF construction, X-VRF, based on the XMSS signature scheme. An innovation of our work is to use the state of the blockchain to counter the undesired stateful nature of XMSS by constructing a blockchain-empowered VRF. While increasing the usability of XMSS, our technique also enforces honest behavior when creating an X-VRF output so as to satisfy the fundamental uniqueness property of VRFs. We show how X-VRF can be used in the Algorand setting to extend it to a quantum-safe blockchain and provide four instances of X-VRF with different key life-time. Our extensive performance evaluation, analysis and implementation indicate the effectiveness of our proposed constructions in practice. Particularly, we demonstrate that X-VRF is the most efficient quantum-safe VRF with a maximum proof size of 3 KB and a possible TPS of 449 for a network of thousand nodes.

1 Introduction

Blockchain technologies have attracted tremendous attention from the research and industrial community. This immense interest ensues from their great promise and the expansion of cryptocurrencies such as Bitcoin or Algorand. Early blockchain systems (e.g., Bitcoin) are based on Proof-of-Work (PoW). This is essentially a consensus mechanism that enables a "lottery" among the miners where the winner gets the reward and decides how to extend the blockchain (i.e., adds the next block). The winner is elected as the first node who solves a difficult computational puzzle. Due to the costly nature of PoW thus less environmentally sustainable, an alternative consensus mechanism based on Proof-of-Stake

(PoS) has gained popularity [2]. The assumption behind PoS is that the majority of the wealth in the system is controlled by the honest participants. In contrast, PoW considers that the majority of the computing power belongs to the honest participants.

An important cryptographic primitive that many PoS solutions rely on for security is Verifiable Random Functions (VRFs). VRFs can also be used in different blockchain applications as described in later sections. Despite their usefulness, a significant concern regarding the currently deployed VRF solutions is that they are susceptible to attacks by powerful quantum computers. Major recent advances [10] in quantum computing technologies have increased the importance of research in the domain of post-quantum cryptography, i.e. schemes that resist attacks from scalable quantum computers. Indeed, such quantum computers can break the currently used classical cryptosystems such as RSA and discrete logarithm based systems, including ECVRF deployed in Algorand.

In light of the increasing importance of quantum-safe cryptographic solutions, researchers have also been focusing on making tools used in blockchain to be quantum-safe. For example, Esgin et al. [12] designed a quantum-safe VRF scheme based on lattice-based cryptography, named LB-VRF. Despite being a significant step forward in making post-quantum VRFs practical, this scheme has a major drawback. In particular, it requires constant key updates since a key pair can just be used to generate only a few VRF outputs (e.g. their most efficient scheme just has a single output). This requirement of constant key updates leads to further complications to accommodate for the weaker VRF functionality.

Our goal in this work is to overcome such challenges in this only-known practical post-quantum VRF scheme [12], by introducing a VRF construction based solely on symmetric primitives. In addition to being more efficient than prior post-quantum schemes and supporting much longer key lifetime, our approach is based the safest post-quantum cryptography alternative. Indeed, symmetric primitives have been defined and standardized for decades, hence boosting confidence in using them. Other post-quantum candidates such as multivariate or lattice-based cryptography are relatively new and may more likely be broken when more cryptanalytic efforts are undertaken [4]. We proceed to discuss VRFs in more detail and how they are employed in the blockchain.

1.1 Verifiable Random Function (VRF)

Micali et al. [24] introduced the concept of VRF, a variant of Pseudorandom Functions (PRFs) satisfying a *verifiability* property. This means that the knowledge of a secret key enables to evaluate the pseudorandom function and prove the correctness of such evaluations without revealing the secret key or compromising the function's *pseudorandomness*. In more detail, a VRF is associated with a secret key $\mathsf{sk}_{\mathsf{VRF}}$ and the corresponding public key $\mathsf{pk}_{\mathsf{VRF}}$, which can be used to verify the VRF output. Using the secret key $\mathsf{sk}_{\mathsf{VRF}}$, a user can compute the function $(\mathsf{y}_{\mathsf{VRF}}, \pi_{\mathsf{VRF}}) \leftarrow \mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x)$ at some input x and generate a corresponding proof π_{VRF} of the correct computation of $\mathsf{y}_{\mathsf{VRF}}$. The proof π_{VRF} can be verified using $\mathsf{pk}_{\mathsf{VRF}}$ (and public parameters). Additionally to the pseudorandomness and verifiability properties, a secure VRF should also satisfy the notion of *uniqueness*. This means that under a fixed public key and a fixed VRF input x, there cannot exist valid proofs $\widetilde{\pi_{\mathsf{VRF}}}$ of correct computation corresponding to two *distinct* VRF output values $\mathsf{y}_{\mathsf{VRF}} \neq \widetilde{\mathsf{y}_{\mathsf{VRF}}}$.

1.2 VRFs in the Blockchain

VRFs are widely used in PoS blockchains to conduct secret cryptographic sortition. These include electing block proposers and voting committee members [13,23,9,16,8,17] as well as various applications in smart contracts such as online lottery. Our focus VRF application in this paper is the cryptographic sortition in the blockchain.

Cryptographic sortition is an innovation of Algorand which enables a set of users to select themselves to participate in Algorand's consensus protocol in a private manner. That is, they are not identified to anyone else; including potential adversaries [1]. The committee-based consensus protocol proposed by Gilad et al. [13] for Algorand leverages on a VRF to implement cryptographic sortition. Moreover, the VRF arrangement in Algorand enables fair private non-interactive random selection of committee members, weighted by their account balances.[7] This random selection of committee members in Algorand also prevents attackers from targeting a specific committee member. Additionally, the use of a VRF in Algorand's consensus protocol provides scalability and performance required to support millions of users. The core of Algorand's blockchain is a Byzantine Agreement protocol that is executed among a small randomly chosen committee of users for each round [13]. More precisely, this protocol makes use of a VRF in the following way. Each user holds a secret/public key pair $(\mathsf{sk}_{\mathsf{VRF}}, \mathsf{pk}_{\mathsf{VRF}})$. Let B be a block to be added. Each user should take the following steps to determine whether she is part of the committee [15]:

1. Compute $(\mathsf{y}_{\mathsf{VRF}}, \pi_{\mathsf{VRF}}) \leftarrow \mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, Q)$ for the user secret key $\mathsf{sk}_{\mathsf{VRF}}$ and a publicly known random seed Q and output a pseudorandom value $\mathsf{y}_{\mathsf{VRF}}$ and a proof of correct computation π_{VRF}.
2. Check if $\mathsf{y}_{\mathsf{VRF}}$ is in the target range $[0, P]$, where P is a parameter that depends on the current stake of the user. If this condition holds, the user will be a committee member for B.

The committee membership can be verified by all users in Algorand's network by executing the verification algorithm of the VRF with $\mathsf{pk}_{\mathsf{VRF}}$, Q, $\mathsf{y}_{\mathsf{VRF}}$ and π_{VRF} as input, and additionally checking if $\mathsf{y}_{\mathsf{VRF}} \in [0, P]$. Algorand instantiates their protocol with a long-term (practically unlimited) stateless VRF based on elliptic curves (ECVRF). Similarly, Ouroboros Praos [8] conducts a private test that is executed locally using a VRF to determine whether a participant belongs to the slot leader set for any slots within a specific time period.

The uniqueness, pseudorandomness and provability properties of the VRF play crucial roles in preventing brute-force attacks that try various output values

[7] A user would not benefit from having/creating multiple accounts.

$\mathsf{y_{VRF}}$ in order to find one that falls within the desired range. Particularly, a user with access to the secret key cannot create multiple valid $\mathsf{y_{VRF}}$ values (thanks to uniqueness) and, also it is infeasible for an adversary to predict $\mathsf{y_{VRF}}$ in advance (thanks to pseudorandomness). Moreover, the committee membership procedure as well as its verification are computationally inexpensive, making the consensus protocol highly scalable.

1.3 Our Contributions

As mentioned above, our goal in this work is to introduce an efficient post-quantum VRF solution that overcomes the drawbacks of prior state-of-the-art schemes. In particular, our contributions can be summarized as follows:

- **Post-quantum VRF X-VRF:** We introduce a practical many-time 'blockchain-empowered' post-quantum VRF, called X-VRF (see Section 4). It is built only from symmetric primitives, particularly XMSS signature. We avoid the need for a stateless signature by utilizing the state of the blockchain (i.e., block number) as a counter. This approach makes the VRF stateless from a user's point of view. In fact, the blockchain's block number, which is already maintained by and available to all users, is the only state information required for our application of X-VRF. We provide a technical overview of our approach in the next section.
- **"Naive" post-quantum VRF:** For a good viewpoint of comparison, we also introduce a naive post-quantum VRF scheme that combines a PRF with a non-interactive zero-knowledge proof (NIZK) of correct evaluation (see Section 3). We pick the NIZK based on the proof system in [22] which also uses symmetric primitives and thus is fair to our comparison. As expected, this proposal yields a stateless VRF construction, which we call SL-VRF. It is significantly more costly in terms of computation and communication in comparison to X-VRF while being long-term and stateless. We discuss (in Section 6) how SL-VRF can be deployed in conjunction with X-VRF to exploit the advantages of each proposal. Particularly, SL-VRF can be deployed as a fall-back option thanks to its (practically) unlimited key lifetime.
- **Implementation and performance evaluation:** We implement X-VRF under different settings and provide a thorough evaluation analysis that shows its efficiency and practicality (see Section 5). All of our settings provide a very efficient performance: the computation time for both the evaluation and verify functions is less than 1 ms. A user public key and a secret key are just 64 bytes and 132 bytes, respectively, while the proof size is only around 3 KB for all settings. The main difference between X-VRF instances is the trade-off between (one-time) key generation runtime, memory requirement and the lifetime of a key pair. Our experiments show that SL-VRF is 500 to 5000 times slower than our X-VRF in evaluation and verification, while the proof size is also 13 times larger than our X-VRF (see Table 1).
- **Integration to Algorand:** We integrate X-VRF into the Algorand blockchain and compare our result with the only practical post-quantum

VRF, namely LB-VRF [12], to demonstrate the practicality of our X-VRF (See section 6). To support 100 nodes in the blockchain system, our VRF can achieve almost 1000 transactions per second (TPS). If we increase the number of nodes to 1000, our VRF can still achieve around 500 TPS. Our integration demonstrates that all X-VRF settings offer a better TPS than LB-VRF and additionally providing a practically long to ultra long key life-time, ranging from 45 hours to 20 years, while the LB-VRF key life-time is only of 5 seconds (see Table 2).

1.4 Our Approach

Deterministic XMSS [6], presented in Appendix A.3, is a good candidate for the construction of a post-quantum VRF. The main reason is that it is the most efficient post-quantum signature scheme constructed from symmetric primitives in terms of signature size. XMSS employs a hash tree, where each leaf uses a one-time signature scheme called WOTS^+ [19]. WOTS^+ is by construction unique (i.e., for any fixed message and public key, one can only create a single valid signature). However, by itself, deterministic XMSS does not satisfy the fundamental uniqueness property of a VRF. This is since a user can use different tree leaves (that is, different WOTS^+ keys) to construct a new XMSS signature for the same message. Thus, it is obvious that a user can generate two different valid XMSS signatures for a single message.

To satisfy the uniqueness of XMSS, we need to force the user to use a pre-determined WOTS^+ key pair in signing. For this, we make use of the blockchain state, i.e., empower XMSS with blockchain. In particular, the block number of a particular round in the blockchain consensus can serve as a global counter. This , can be used to force users to use a specific WOTS^+ key pair at each round. More precisely, at block number K, when verifying the XMSS-based VRF output, the verifier also checks that the leaf index indicated by the authentication path is consistent with K (see Figure 1). As a result, this ensures that the user cannot choose between different WOTS^+ keys and also allows users to avoid maintaining a local state. We formally prove that our X-VRF constructed from this approach satisfies all security requirements of a VRF. On the other hand, due to the computational/storage cost of creating and storing a big hash tree, X-VRF cannot be used to create, say, 2^{64} outputs as each output consumes one leaf. We investigate various X-VRF instances with trade-offs between computation, storage and lifetime of an X-VRF key pair. For example, the X-VRF-27 (with 2^{27} leaves) construction offers a key lifetime of more than 20 years in the Algorand setting and hence can be seen as long-term VRF, though it requires a relatively long (around 2 days on a single core) one-time key generation process. Of course, this process can be optimized using standard techniques such as parallel processing or delegating computation.

In the rest of the work, we formally define VRFs, and commence by presenting our "naive" VRF proposal, SL-VRF. Then, our main constructions, X-VRF, is presented, followed by our implementation and evaluation results. We finally discuss the integration of our proposals to Algorand.

2 Verifiable Random Function (VRF)

This section formally defines the concept of VRF and its security requirements.

Definition 1 (Verifiable Random Function (VRF) [24]). *A VRF with input length $\ell(\lambda)$ and output length $m(\lambda)$ consists of the following polynomial-time algorithms:*

$\mathsf{ParamGen}(1^\lambda)$**:** *On input the security parameter 1^λ, this probabilistic algorithm outputs some global, public parameter $\mathsf{pp_{VRF}}$.*

$\mathsf{KeyGen}(\mathsf{pp_{VRF}})$**:** *On input public parameter $\mathsf{pp_{VRF}}$ this probabilistic algorithm outputs two binary strings, a secret key $\mathsf{sk_{VRF}}$ and a public key $\mathsf{pk_{VRF}}$.*

$\mathsf{VRFEval}(\mathsf{sk_{VRF}}, x)$**:** *On input a secret key $\mathsf{sk_{VRF}}$ and an input x this algorithm outputs the VRF value VRF and the corresponding proof π_{VRF} proving that $\mathsf{y_{VRF}}$ was correctly computed.*

$\mathsf{Verify}(\mathsf{pk_{VRF}}, \mathsf{y_{VRF}}, x, \pi_{\mathsf{VRF}})$**:** *On input $(\mathsf{pk_{VRF}}, \mathsf{y_{VRF}}, x, \pi_{\mathsf{VRF}})$ this probabilistic algorithm outputs either YES or NO.*

A secure VRF satisfies the following properties:

Provability: *If $(\mathsf{y_{VRF}}, \pi_{\mathsf{VRF}})$ is the output of $\mathsf{VRFEval}(\mathsf{sk_{VRF}}, x)$, where $\mathsf{pk_{VRF}}, \mathsf{sk_{VRF}}$ are honestly generated, then: YES$\leftarrow \mathsf{Verify}(\mathsf{pk_{VRF}}, \mathsf{y_{VRF}}, x, \pi_{\mathsf{VRF}})$.*

Pseudorandomness: *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time adversary playing the experiment $\mathsf{Exp}^{pr}_{\mathcal{A}}$ presented in Fig. 2. A VRF achieves pseudorandomnes iff $\Pr[\mathsf{Exp}^{pr}_{\mathcal{A}} = 1] < 1/2 + negl(\lambda)$.*

Uniqueness: *No values $(\mathsf{pk_{VRF}}, \mathsf{y_{VRF,1}}, \mathsf{y_{VRF,2}}, x, \pi_{\mathsf{VRF,1}}, \pi_{\mathsf{VRF,2}})$ can satisfy $\mathsf{Verify}(\mathsf{pk_{VRF}}, \mathsf{y_{VRF,1}}, x, \pi_{\mathsf{VRF,1}}) = \mathsf{Verify}(\mathsf{pk_{VRF}}, \mathsf{y_{VRF,2}}, x, \pi_{\mathsf{VRF,2}}) = 1$, when $\mathsf{y_{VRF,1}} \neq \mathsf{y_{VRF,2}}$*

Note: In contrast to the *unconditional* uniqueness property defined in [24], our VRF constructions satisfy *computational* uniqueness, where the running time of the adversary attacking the uniqueness property is polynomially bounded. This stems from the fact that we rely on the collision-resistance of a hash function.

3 SL-VRF: Stateless Verifiable Random Function from PRF and NIZK

We adopt the idea of instantiating a VRF from a PRF+NIZK construction which was introduced in [14]. Recent works like [7,22] allow to prove knowledge of a secret key k that generates $y \leftarrow \mathsf{PRF}(k, x)$ while preserving the secrecy of k, using only symmetric primitives; where x, y are public information. This means that PRF is an arithmetic circuit like a block cipher or a hash function. The current state of the art is the NIZK scheme of Katz et al. (KKW) [22] which is at the heart of the post-quantum security of the digital signature Picnic [7] submitted to the NIST standardization process.

The KKW NIZK protocol [22] proves the knowledge of a secret key without revealing any information about it, based on the input and outputs of a binary circuit like a PRF or a block cipher. KKW is instantiated using the concept of MPC-in-the-head introduced in [20]. MPC-in-the-head simulates a multi-party computation of the circuit between P parties. Then to prove the knowledge of the secret key, the protocol reveals the views of all parties except one. The proof is verified by recomputing the output based on the view of $P-1$ parties.

The size of the proof depends directly on the number of parties P of the MPC protocol. In the circuit, there are four different operations: Addition with a public constant, addition of values computed by all parties ("OR" gates), multiplication with a constant and multiplication of values computed by all parties ("AND" gates). Only the last one requires communication between the parties to be performed. This means that the proof contains the views of $P-1$ parties for each "AND" gate of the circuit, and therefore its number needs to be optimized. For this reason, KKW works with the block cipher LowMC [3] which is a cryptographic primitive with low multiplicative complexity, i.e. low number of multiplications in a circuit ("AND" gates).
Ensuring Uniqueness. The idea behind our stateless VRF construction is to use the block cipher LowMC as a PRF to generate the random outputs and then prove with KKW the knowledge of the secret key $\mathsf{sk_{VRF}}$. In other words, taking a message x as an input, a user generates the corresponding pseudorandom outputs $y \leftarrow \mathsf{PRF}(\mathsf{sk_{VRF}}, x)$ and then uses the KKW protocol as the VRF proof. KKW protocol is made non-interactive using the Fiat-Shamir transform. However, KKW allows to prove the knowledge of the secret key that generates y from the public x without requiring the use of the public key linked to the secret key $\mathsf{sk_{VRF}}$. This causes a problem for the uniqueness of the VRF. Indeed, a user could generate a different value y' for the same input x using another secret key $\mathsf{sk_{VRF}}'$. Therefore, we modify the public key as follows: $\mathsf{pk_{VRF}}$ is now composed of two elements $\mathsf{pk_{VRF_1}}$ and $\mathsf{pk_{VRF_2}}$ such that $\mathsf{pk_{VRF_2}} \leftarrow \mathsf{PRF}(\mathsf{sk_{VRF}}, \mathsf{pk_{VRF_1}})$. The evaluation procedure will prove that the secret key which generated y from x is the same as that which generated $\mathsf{pk_{VRF_2}}$ from $\mathsf{pk_{VRF_1}}$. Thus it is infeasible for a malicious user to change its secret key in order to generate another output of the VRF.

3.1 SL-VRF from PRF+NIZK Construction

$\mathsf{ParamGen}(1^\lambda)$: Pick a collision-resistant hash $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ (for the Fiat-Shamir transform) and a Pseudorandom function $\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n$. Output public parameters $\mathsf{pp_{VRF}} = (\mathsf{H}, \mathsf{PRF})$.

$\mathsf{KeyGen}(\mathsf{pp_{VRF}})$: On input public parameters computes $\mathsf{sk_{VRF}} \xleftarrow{\$} \{0,1\}^n$ and $\mathsf{pk_{VRF_1}} \xleftarrow{\$} \{0,1\}^n$. Then it computes $\mathsf{pk_{VRF_2}} \leftarrow \mathsf{PRF}(\mathsf{sk_{VRF}}, \mathsf{pk_{VRF_1}})$ and sets $\mathsf{pk_{VRF}} = (\mathsf{pk_{VRF_1}}, \mathsf{pk_{VRF_2}})$

$\mathsf{VRFEval}(\mathsf{sk_{VRF}}, x)$: Given $\mathsf{sk_{VRF}}$ and a message x, the algorithm computes:
1. $y \leftarrow \mathsf{PRF}(\mathsf{sk_{VRF}}, x)$
2. $\mathsf{pk.check} \leftarrow (x, y, \mathsf{pk_{VRF}})$
3. $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{Prove}(\mathsf{pk.check}, \mathsf{sk_{VRF}})$. For a relation $\mathcal{R}$ the NIZK proof π_{NIZK} holds iff:

(a) $y \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{VRF}}, x)$ and
(b) $\mathsf{pk}_{\mathsf{VRF}_2} \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{VRF}}, \mathsf{pk}_{\mathsf{VRF}_1})$.
This can be done due to the KKW [22,21] procedure on a binary circuit composed of two PRFs (LowMC block cipher as explained before) linked with an additional "AND" gate, assuring that both statements are fulfilled.

$\mathsf{VRFVerify}(y, x, \pi_{\mathsf{NIZK}})$: On input $(\pi_{\mathsf{NIZK}}, x, y)$ it runs NIZK verification algorithm $\mathsf{Verify}(x, y, \pi_{\mathsf{NIZK}})$ of the underlying KKW NIZK proof and outputs 0/1.

The security depends on the properties of the underlying PRF for a given key $\mathsf{sk}_{\mathsf{VRF}}$ (collision resistance and one-wayness) and the security of KKW [22].

SL-VRF ***Security Discussion.*** The provability of SL-VRF follows via direct investigation. As long as the underlying KKW NIZK proof is correct, SL-VRF is provable. The security, i.e. uniqueness and pseudorandomness of our SL-VRF depends on the properties of the underlying PRF. The uniqueness follows from two facts; first, the PRF output is a deterministic function of the secret key $\mathsf{sk}_{\mathsf{VRF}}$ and the input x, meaning that evaluating the PRF twice on the same value yields the same output. Secondly, the validity proof of the statement $(3)(b)$ in the VRF evaluation procedure (VRFEval) presented in Section 3.1 forces each user to use her fixed secret key for each evaluation. This ensures uniqueness due to the deterministic property of the PRF as explained before. The pseudorandomness of the VRF output is inherited from the corresponding pseudorandomness property of the underlying PRF. In order to evaluate the efficiency of our SL-VRF, we need to analyse the underlying NIZK proof construction. A detailed evaluation is provided in Section 5.

4 X-VRF: Verifiable Random Function from XMSS

This section introduces a construction of a secure VRF from XMSS (see Definition 2 in Appendix A.3). As discussed in the introduction, naively extending XMSS to a VRF does not result in a secure construction as the uniqueness property can easily be violated. In particular, when a user constructs a hash tree in XMSS, she can use any of the leaves (i.e., WOTS^+ keys) to create the XMSS signature. As a result, XMSS by itself does not satisfy uniqueness.

This is indeed very problematic in a blockchain application that, for example, uses the VRF output to perform leader election (as in Algorand). More specifically, the user can simply create a huge hash tree, say, with N leaves for XMSS. Then she will be able to amplify her success probability of being elected by a factor of N, as she can try to create the XMSS-based VRF output from each leave and can output the one that is successful.

To circumvent this problem, we index every VRF evaluation and modify the uniqueness requirement to the case where for a fixed message and public key, the VRF evaluations with the same index always lead to the same value. Then, in the later sections, we *enforce all* users to use a pre-determined index ctr when creating an XMSS-based VRF output. This way, the users cannot choose between multiple leaves and can only produce a single signature output on a message.

4.1 X-VRF from XMSS Construction

$\mathsf{ParamGen}(1^\lambda)$: On input security parameter λ, output public parameters $\mathsf{pp}_{\mathsf{VRF}} = \mathsf{H}$, for $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^n$. Where H is a hash function.

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{VRF}})$: On input public parameters $\mathsf{pp}_{\mathsf{VRF}}$,

1. Run $(\mathsf{XMSS.idx}, \mathsf{XMSS.sk}, \mathsf{XMSS.pk}) \leftarrow \mathsf{XMSS.KeyGen}(1^\lambda)$,
2. Output $(\mathsf{idx}_{\mathsf{VRF}}, \mathsf{pk}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}) = (\mathsf{XMSS.idx}, \mathsf{XMSS.pk}, \mathsf{XMSS.sk})$.

$\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x, \mathsf{idx}_{\mathsf{VRF}})$: On input $\mathsf{sk}_{\mathsf{VRF}} = \mathsf{XMSS.sk}$, a message x and an index $\mathsf{ctr} = \mathsf{idx}_{\mathsf{VRF}}$,

1. Run $\mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth}) \leftarrow \mathsf{XMSS.Sign}(\mathsf{XMSS.sk}, x, \mathsf{ctr})$,
2. Set $\pi_{\mathsf{VRF}} = \mathsf{XMSS}.\sigma$,
3. Compute $\mathsf{y}_{\mathsf{VRF}} \leftarrow \mathsf{H}(\mathsf{XMSS}.\sigma, x)$,
4. Output $(\pi_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF}})$.

$\mathsf{VRFVerify}(\mathsf{pk}_{\mathsf{VRF}}, x, \mathsf{y}_{\mathsf{VRF}}, \pi_{\mathsf{VRF}})$: On input $(\pi_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF}})$, the public key $\mathsf{pk}_{\mathsf{VRF}}$ and a VRF input x,

1. Parse $\pi_{\mathsf{VRF}} = \mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth})$,
2. If i and $\mathsf{XMSS.Auth}$ are inconsistent (i.e., if the leaf index indicated by $\mathsf{XMSS.Auth}$ is not equal to i), output NO.
3. Otherwise, if the verification of $\mathsf{XMSS}.\sigma$ succeeds and $\mathsf{y}_{\mathsf{VRF}} = \mathsf{H}(\mathsf{XMSS}.\sigma, x)$, output YES.
4. Otherwise output NO.

In VRFEval, the counter decides which leaf of the XMSS tree is used and this is checked in VRFVerify. In our blockchain application, we enforce users to use a specific publicly known counter value. This is so that the user cannot choose multiple leaves to create a VRF output at a particular point. This is crucial to guarantee uniqueness. It is indeed easy to establish a global counter value in the blockchain environment since it can simply be set to the block number K mod N, where N is a fixed public integer denoting the maximum number of rounds a key pair can be used. In particular, we set $N = 2^h$ for an XMSS tree of height h (See Figure 1 for an example with $N = 4$).

We also remark that with access to such a global counter, the users no longer need to store individual state information. That is, the VRF itself in a way becomes stateless as the users can simply retrieve the block number from the blockchain and do not need to worry about maintaining a state themselves.

4.2 X-VRF Security Analysis

The most critical property we need to analyze is the uniqueness. To this end, we first focus on the uniqueness of XMSS under the constraint that the index used to create the signature is the same. This leads to the following lemma whose proof is given in Appendix A.1. Then, we state the security of X-VRF and provide its proof in Appendix A.2.

Lemma 1 (XMSS **Uniqueness).** *Let* $\mathsf{XMSS}.\sigma_1 = (\mathsf{WOTS}^+.\sigma_1, i, \mathsf{XMSS.Auth}^1)$ *and* $\mathsf{XMSS}.\sigma_2 = (\mathsf{WOTS}^+.\sigma_2, i, \mathsf{XMSS.Auth}^2)$ *be two valid* XMSS *signatures created by a PPT adversary on the same message m and under the same public*

Table 1: Performance evaluation of X-VRF, SL-VRF, ECVRF and LB-VRF. For LB-VRF, we report the results provided in [12]. For the memory requirement of X-VRF, an evaluator stores 2^h 256-bit values for $h \in \{15, 19, 23, 27\}$.

Instances	ECVRF	X-VRF-15	X-VRF-19	X-VRF-23	X-VRF-27	SL-VRF	LB-VRF
Memory for Eval	negl.	1 MB	16 MB	256 MB	4 GB	negl.	negl.
PK size	32 B	64 B	64 B	64 B	64 B	48 B	3.32 KB
SK size	32 B	132 B	132 B	132 B	132 B	24 B	0.45 KB
Proof size	80 B	2.63 KB	2.76 KB	2.88 KB	3.01 KB	40 KB	4.94 KB
KeyGen	0.05 ms	48.9 s	14.2 min	3.73 h	$\approx$ 58 h	0.38 ms	0.33 ms
VRFEval	0.10 ms	0.72 ms	0.75 ms	0.78 ms	0.80 ms	765 ms	3.1 ms
VRFVerify	0.10 ms	0.87 ms	0.91 ms	0.94 ms	0.97 ms	475 ms	1.3 ms

key XMSS.pk *and the same index* i *(i.e.,* $\mathsf{XMSS.Verify}(\mathsf{XMSS.pk}, m, \mathsf{XMSS}.\sigma_1) = \mathsf{XMSS.Verify}(\mathsf{XMSS.pk}, m, \mathsf{XMSS}.\sigma_2) = 1$*). If the hash function used in the* XMSS *definition is collision-resistant, then* $\mathsf{XMSS}.\sigma_1 = \mathsf{XMSS}.\sigma_2$ *(i.e.,* XMSS *is unique provided that the indices in the two signatures are the same).*

Theorem 1 (X-VRF Security). X-VRF *is correct and satisfies the properties of computational uniqueness and pseudorandomness in the random oracle model. In particular, the uniqueness holds in the sense that the same* ctr *(or leaf index) must be used in* VRFEval *as in Lemma 1.*

5 Implementation and Evaluation

This section presents the implementation results of our X-VRF construction as well as the naive SL-VRF which is used as a baseline for comparison. Traditionally, VRF constructions from unique signatures require the signature to be stateless. However, we argue that in most of the blockchain applications a stateful VRF is sufficient as the blockchain can easily maintain the state.

Both SL-VRF and X-VRF have been implemented in C for a level of post-quantum security of $\lambda = 128$. We choose to work with SHA-256 as the hash function. The implementation of SL-VRF is based on [25]. It couples KKW [22] with the Fiat-Shamir transform to get a NIZK. The implementation of X-VRF is derived from the XMSS implementation provided in [18]. Both implementations were deployed on a machine with an Intel(R) Core i7-86500 CPU @ 1.90GHz 12GB of RAM.

5.1 VRF Proof Sizes

Table 1 summarizes the proof size of each instance. As expected, the X-VRF constructions clearly outperform SL-VRF with proof sizes at least 13.3 times smaller. The VRF proof size in a X-VRF instance denoted by $|\mathsf{X\text{-}VRF}.\pi_{\mathsf{VRF}}|$ is computed by the following formula $|\mathsf{X\text{-}VRF}.\pi_{\mathsf{VRF}}| = h \cdot n + n \cdot \mathtt{len}$. The VRF proof of our SL-VRF construction depends only on the size of the NIZK proof

[21]. The size that we presented in Figure 1 differs from the one provided by Katz et al. [22]. The reason behind this is that we used the parameter of the NIST submission presented in [21] which have been optimized to give a compromise between algorithm efficiency and proof size. The size of the VRF proof for SL-VRF construction is denoted by $|\mathsf{SL\text{-}VRF}.\pi_{\mathsf{VRF}}|$, and we refer the reader to [22,21] for further details.

5.2 Memory Requirements

In Table 1, we propose applicable memory requirements for our four instances of X-VRF. It is important to know that the memory capacity impacts principally the X-VRF evaluation procedure. The X-VRF key generation procedure does not require expensive memory as the full XMSS tree does not need to be fully stored.

A capacious memory can reduce the offline computations for the XMSS evaluation procedure that are the authentication path selection (the grey nodes in Figure 3). Devices with high memory capacity will be able to store the whole XMSS tree and therefore avoid any offline computation. However, the required memory to store the full tree (together with the WOTS^+ keys) would be impractical particularly for the X-VRF-23 and X-VRF-27 instances which would require respectively 35 GB and 350 GB to store the complete binary tree. Therefore, we propose to store the $h-1$ levels of the XMSS tree (i.e., the whole tree except the bottom leaves and WOTS^+ keys). This means that offline computations are necessary every two evaluations and requires the computation of the two WOTS^+ key pairs based on a secret seed and a PRF. For example in Figure 3, if $\mathsf{XMSS.index} = 0$, the offline phase computes both first WOTS^+ key pairs. Then, if $\mathsf{XMSS.index} = 1$ there is no offline computation required as both first WOTS^+ key pairs have been generated. Then, when $\mathsf{XMSS.index} = 2$ the offline computation will generate WOTS^+ key pairs number 2 and 3. The memory required with this technique for all instances is highlighted in Table 1. This advantage to have cheap offline computations ($\ll 1$ ms) needs to be done for only half of the X-VRF evaluations. Most importantly, it requires a maximum of 4GB of memory, which can be considered to be acceptable and applicable to lightweight devices.

5.3 VRF Computation Efficiency

We further present four instances of X-VRF with different heights of the XMSS tree. We evaluated VRF with heights 15 (denoted as X-VRF-15), 19 (denoted as X-VRF-19), 23 (denoted as X-VRF-23) and 27 (denoted as X-VRF-27). This means each of these instances can generate, respectively, at most 2^{15}, 2^{19}, 2^{23} and 2^{27} VRF evaluations. Table 1 summarizes the performance of each instance. When it comes to the key generation procedure, SL-VRF outperforms all X-VRF instances as expected. This is because it only requires the selection of a random element of n bits, while all four instances of X-VRF need to generate XMSS tree with a greater height which leads to more computation.

Although the KeyGen of SL-VRF is much faster than that of X-VRF, the running time of the evaluation algorithm Eval of SL-VRF cannot compete with

Table 2: Estimated TPS for our VRFs with different signatures on various number of nodes. 'N/A' means the given number of nodes cannot be supported.

#Nodes	Signature	X-VRF-15	X-VRF-19	X-VRF-23	X-VRF-27	SL-VRF	ECVRF	LB-VRF
10	Ed25519	1010	1010	1010	1009	940	1015	1000
	Rainbow	1008	1008	1008	1007	938	1013	998
	SPHINCS$^+$	34	34	34	34	32	34	32
50	Ed25519	990	989	988	987	639	1014	939
	Rainbow	988	987	986	985	638	1012	937
	SPHINCS$^+$	33	33	33	33	21	34	22
100	Ed25519	966	963	961	958	263	1014	862
	Rainbow	964	961	959	957	263	1012	861
	SPHINCS$^+$	32	32	32	32	9	34	10
1000	Ed25519	521	496	474	449	N/A	1000	N/A
	Rainbow	520	495	473	448		998	
	SPHINCS$^+$	17	17	16	15		34	
Key Lifetime		45 hours	1 month	1.3 years	>20 years	Practically unlimited		5 seconds

the stateful construction X-VRF. Eval of SL-VRF requires the simulation of MPC computation, which is quite costly.

The performance of the evaluation algorithm of X-VRF instances is really competitive. The underlying reason is that only the WOTS$^+$ signature needs to be computed at the spot, as the authentication path could be pre-computed. For X-VRF, the evaluation cost is at most $(w-1) \cdot \texttt{len}$ calls of the cryptographic hash function. Our results demonstrates that X-VRF is at least 956 times faster than SL-VRF for the VRF evaluation.

The VRF verification of X-VRF also outperforms SL-VRF by at least 474 ms. The total cost verification for SL-VRF is the verification of a number of execution of the MPC-in-the-head with $P-1$ parties (details presented in [21]), while X-VRF needs only a maximum of $h + w \cdot \texttt{len} + \log \texttt{len}$ calls to the hash function H. Note that verification runtime in the blockchain application is an important metric as this needs to be repeated by all (honest) committee members.

6 Integration to Algorand

In this section, we discuss the details of our X-VRF integration into the Algorand consensus protocol. As discussed earlier, the uniqueness of X-VRF (and the underlying XMSS signature) crucially relies on enforcing the use of a *single* predetermined counter ctr in VRFEval (or index XMSS.idx in XMSS.Sign). We can achieve this easily in the blockchain setting. In particular, there is already the block number that serves as a globally agreed, inalterable and publicly accessible counter. Let $N = 2^h$ be the number of leaves in XMSS and K be the block number. Then, we let the verifiers check that the ctr (or XMSS.idx) used at block number K is equal to $K \bmod N$. Therefore, every user is forced to use the leaves in a certain order and we can achieve uniqueness.

6.1 Performance Estimation

To better illustrate our benchmark results, it is important to understand the bottleneck of the current Algorand protocol. As of September 2020, Algorand's mainnet employs over 1000 nodes, and allows for roughly 5.4 MB of data propagated per block, as a result of their efficient consensus protocol. It consists of 5000 signed transactions, at 1064 bytes each, and 80 KB for VRF data. To break up the VRF part, 1000 nodes implies 1000 ECVRF proofs, which is around 80 KB of data. It is straightforward to see that the majority of the data is reserved for transactions. Under the assumption that a transaction is 1KB on average, and the signature is Ed25519, Algorand allows for 5K transactions per block, or, roughly 1K transaction per second (TPS) as Algorand generates a block in about 5 seconds.

Note that, in Algorand, although the final blocks only log transactions (while VRF payload is not included in the final blocks by design - the committee members only attest that they have seen enough votes, without putting that information to the block for performance reasons), the actual data propagated through the network during each block is indeed the combination of VRF payload and the transaction payload. Therefore it makes sense to use this total payload size as the network's throughput limitation, rather than the actual blocksize. To summarize, we follow [12] and estimate the Algorand TPS throughput as follows:

$$\text{TPS} = \frac{\text{payload size} - \text{total VRF cost} \times \#\text{nodes}}{(\text{transaction size} + \text{signature size}) \times \text{blocktime}}.$$

Note that as 'refreshing' a key pair happens much less frequently for X-VRF (in comparison to LB-VRF), the per-round cost of a key refreshment is negligible in our setting. Using the above formula, we estimate the TPS throughput of Algorand using our VRF in combination with different signature schemes that are used to authenticate transactions. In this computation, we make the following assumptions as in [12] for a fair comparison. We assume a payload size of 5.4 MB. We follow Algorand and assume 1 KB data for transaction size. As Algorand generates a block in about 5 seconds, we take blocktime as 5 seconds. The last moving part in the equation is the signature size. For this component, we consider the original Ed25519 signature used by Algorand, whose signature size is 64 bytes. In addition, we also consider two extreme cases in the post-quantum setting: (i) Rainbow[8] [11] -the shortest signature finalist candidate in NIST's Post-Quantum Cryptography standardization process- whose signature size is as small as 66 bytes[9], and (ii) SPHINCS$^+$ [5], whose signature is 30696 bytes, which relies on symmetric primitives only. For X-VRF, we further set the tree heights as 15, 19, 23, 27. This means a user can use the same key in X-VRF for roughly 45 hours, 30.3 days, 1.33 years and more than 20 years, respectively. For SL-VRF, the nodes would not ever need to re-generate keys in practice. We will talk about X-VRF key schedules in the next section.

Turning to the performance comparison, as one shall see in Table 2, our X-VRF can be integrated into Algorand for all four settings. For the real-world scenario (1000 nodes), with X-VRF we see a roughly 55% reduction in TPS for

[8] `https://www.pqcrainbow.org/`

[9] The signature length of 48 bytes of an earlier Rainbow version is used in [12].

both Ed25519 and Rainbow. Note that it is a common understanding that post-quantum cryptography performs much worse, compared to classical ones. Hence, we believe that even a 55% reduction should be considered as a great achievement of our solution, rather than a drawback. The throughput for SPHINCS$^+$ is much worse, recording 16 TPS on average. We note that even this case is still faster than Bitcoin (at 5 TPS). On the other hand, the stateless VRF SL-VRF does not perform well for large networks. Our simulation shows that the consensus is only possible for a network of at most around 100 nodes. When the number of nodes is higher, the blockchain capacity is not sufficient to transmit the SL-VRF payload, thus, making the protocol unusable.

6.2 Dual Key Scheduling

Now that we know X-VRF provides a more practical solution than SL-VRF, it is imperative to argue the usability of our stateful X-VRF. In our vision, a protocol should deploy both X-VRF and SL-VRF. X-VRF provides great performance, and should always be used when they are available. However, as per setup, an X-VRF key needs to be refreshed once in a while, requiring the user to be online at a certain time. This update requires an additional 64 bytes for VRF public keys, and a signature on the public key for authenticity, per cycle. We consider this cost to be negligible, compared to the remaining cost as for X-VRF-23, for example, a cycle happens only every 1.33 years. In practice though, we cannot rule out the cases where users may lose their keys. For conservative purposes, nonetheless, it is desirable to have a backup plan: the user falls back to SL-VRF if he has consumed all X-VRF keys and has not uploaded a new X-VRF key (see Fig. 1).

There are nonetheless two additional subtleties here. First, if every user needs to update their keys periodically, the network may be flooded by X-VRF keys that are never used. Our solution is as follows. For relay nodes who may be very frequently selected as committee members, we suggest to use X-VRF. They are actually a very small portion of the user base, and account for the majority of VRF payloads. For casual users who perhaps will vote rarely in their lifetime, it is sufficient to use SL-VRF, which minimizes the number of key updates.

The other issue is with the VRF randomness. At a given round when the user does not have an X-VRF key, the user may actually choose to either upload a new X-VRF key, or use his default SL-VRF key. This breaks the uniqueness of the VRF. Our solution is to enforce the user to announce its new X-VRF key a few (say, k) rounds prior to it being active, where k is a system parameter and is currently set to 10 by Algorand blockchain. This approach is indeed already adopted by Algorand with its ECVRF, to limit attackers with a large share of tokens from speculating the block randomness (derived from ECVRF) in the future. It is straightforward to see that, with this restriction, for any given round, if the user has announced an X-VRF key 10 blocks earlier, then it must use that key; otherwise it must use its SL-VRF key. Thus, uniqueness remains intact.

Eventually, we achieve a post-quantum blockchain that supports both X-VRF and SL-VRF. Under the assumption that most of the users will be online regularly, we further assert that the final TPS will be (very) close to the data for X-VRF in Table 2. Since this dual VRF is an orthogonal direction from this paper, we leave the rigorous analysis to future work.

6.3 X-VRF Instances

As explained previously, we propose four different instances of X-VRF and these are the only applicable constructions to Algorand's 1000-node setting as the results in Table 2 demonstrate. Because of the stateful nature of the XMSS signature, there is a maximum number of possible VRF evaluations per key pair. Our goal in the choice of XMSS tree heights is to have VRF instances that can be used for at least one day in Algorand (X-VRF-15) without updating the keys, another for at least one month (X-VRF-19), the third one for at least one year (X-VRF-23) and the last one (X-VRF-27) which could be used for more than 20 years. Each of these VRF instances has different advantages which can be summarized as follows. When using the X-VRF-15, the keys need to be updated every 45 hours. If we assume that each node stores the full XMSS tree it will require 1 MB of storage which is 256 times less than the memory required to store a XMSS tree when using the X-VRF-23 and even 4096 times less than in the case of X-VRF-27. The main disadvantage of using X-VRF-15 is the regular key update that needs to be performed every 45 hours yielding several regular updates during the year. X-VRF-19 allows the network to update the keys only once per month but the computational cost of these monthly updates is 32 times higher than the cost needed when X-VRF-15 is used. X-VRF-23 offers the possibility to make this update only every 1.33 years but the cost of this update for each node takes 3.73 hours on our machine and is 256 times greater than the cost required in X-VRF-15. Finally our last instance, X-VRF-27 avoids the need for a key update for more than 20 years, the key generation would be only necessary when new nodes join the network or when a node has lost its key. The main advantage of this instance is that the network does not need to go through a regular key update similar to the SL-VRF instances. The disadvantage is the cost to join the network or the cost of losing the key which takes around two days on our machine and is 4096 times greater than the cost of an update in X-VRF-15.

Table 2 illustrates the expected TPS for each of X-VRF instances, and as explained previously the best performance is achieved with the Rainbow signature scheme. For this part, we assume that nodes will not lose their key. For a network composed of 10 nodes, all X-VRF instances achieve the same expected TPS which means that in a network of 10 to 50 nodes instances with fewer key updates could be privileged. For a network of 100 nodes, X-VRF-15 achieves the best TPS, however its TPS difference with X-VRF-27 consists of only 8 transaction per second. When there are 1000 nodes the difference of TPS is logically larger, X-VRF-15 could process 72 transactions per second more than X-VRF-27 when using Ed25519 or Rainbow signature scheme. However, the synchronization of a generalized key update for a larger network could be more challenging and could slow down the process.

Memory Optimization. We presented in Section 5.2 the ideal memory requirement to achieve a balance between offline computations and memory consumption (See Table 1). However, it is important to know that these memory requirements are flexible and can be adapted to user specified preconditions. As previously explained, the fast way to evaluate the VRF is to pre-store the path in the tree and then compute the WOTS^+ signature for the current round. However, the current node does not necessarily need to store $h-1$ levels of the *full* tree. Indeed, the node can pre-compute and store only certain paths that are needed for the rounds in the near future. As the rounds progress, the paths that are no longer needed can be discarded and new paths can be pre-computed and stored. This way, we can keep the memory requirements at even lower levels. Overall, there are straightforward trade-offs to be considered depending on the user's system specifications.

Choice of Instance. We showed that all four X-VRF constructions are promising post-quantum VRFs applicable in an existing network like Algorand. Each of them have different advantages going from memory consumption to key update times. To avoid the challenges of synchronizing key updates throughout the network, X-VRF-27 appears to be the best. If the focus is on achieving the best TPS and reducing the impact of key loss, then X-VRF-15 would be the best. X-VRF-23 provides a trade-off between TPS and the recurrence of key updates. Moreover, our proposition of a dual key system by coupling X-VRF with SL-VRF combines the best of two worlds.

6.4 Comparison with Current State-of-the-art and Final Remarks

To the best of our knowledge, there exists only one other *practical* post-quantum VRF provided in [12] using lattice-based techniques and we refer to it as LB-VRF. Table 2 presents the expected TPS in Algorand, as given in [12], for LB-VRF. Our results show that all four X-VRF instances outperform LB-VRF when it comes to TPS for all node sizes due to its shortest proof size of X-VRF instances compared to LB-VRF (see Table 1). As the number of nodes increases, the advantage of our constructions increases. Another disadvantage of LB-VRF is that the users need to update their keys at *every* round (block generation), hence every 5 seconds in the case of Algorand. Our X-VRF construction, on the other hand, can support the use of the same key pair for at least 45 hours; e.g. X-VRF-15, which has the shortest key lifetime. X-VRF-27 offers the possibility to work with the same key for more than 20 years. Even if it is difficult to compare the algorithms' performances because they were not executed on the same machine as LB-VRF numbers were taken from the original paper [12], X-VRF seems to be more efficient when it comes to VRF Evaluation performances as LB-VRF takes 3.1ms while the slowest X-VRF takes only 0.8 ms. The difference between both verification procedures is too small to draw any conclusion.

This paper introduced the first-known post-quantum VRFs based on symmetric primitives. Our XMSS-based X-VRF proposals, which are made possible thanks to the innovative idea of linking the state of the blockchain with the state of XMSS, support a competitive number of transactions per second in a post-quantum PoS-based consensus protocol. It outperforms the one-time lattice-based VRF when it comes to proof size, while allowing the evaluation of multiple input. The X-VRF is based on long-studied symmetric primitives, all X-VRF instances provide strong security assurances, while also being highly efficient and substantially outperforming current state of the art performances.

References

1. Algorand-what we do. Available at `https://www.algorand.com/what-we-do/faq`.
2. Proof of stake instead of proof of work, July 2011. Available at `https://bitcointalk.org/index.php?topic=27787.0`.
3. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for mpc and fhe. In *EUROCRYPT 2015*, pages 430–454. Springer, 2015.
4. D. J. Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.
5. D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe. The sphincs$^+$ signature framework. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 2129–2146. ACM, 2019.
6. J. Buchmann, E. Dahmen, and A. Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.
7. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017*, pages 1825–1842, 2017.
8. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT 2018*, pages 66–98. Springer, 2018.
9. A. S. de Pedro, D. Levi, and L. I. Cuende. Witnet: A decentralized oracle network protocol. *arXiv preprint arXiv:1711.09756*, 2017.
10. O. Dial. Eagle's quantum performance progress. IBM Research Blog, March 24, 2022, 2022. `https://research.ibm.com/blog/eagle-quantum-processor-performance`.
11. J. Ding and D. Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *ACNS 2005*, volume 3531 of *LNCS*, pages 164–175, 2005.
12. M. F. Esgin, V. Kuchta, A. Sakzad, R. Steinfeld, Z. Zhang, S. Sun, and S. Chu. Practical post-quantum few-time verifiable random function with applications to algorand. In *FC*, pages 560–578. Springer, 2021.
13. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP 2017*, page 51–68. Association for Computing Machinery, 2017.
14. S. Goldwasser and R. Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent. In *CRYPTO 1992*, pages 228–245. Springer, 1992.
15. S. Gorbunov. Algorand releases first open-source code: Verifiable random function, 2018. Available at `https://medium.com/algorand/algorand-releases-first-open-source-code-of-verifiable-random-function-93c2960abd61`.
16. T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

17. L. Hellebrandt, I. Homoliak, K. Malinka, and P. Hanáček. Increasing trust in tor node list using blockchain. In *IEEE ICBC 2019*, pages 29–32. IEEE, 2019.
18. A. Hülsing. Xmss implementation. Available at `https://github.com/XMSS/xmss-reference`.
19. A. Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In *AFRICACRYPT*, volume 7918, pages 173–188. Springer, 2013.
20. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.
21. D. Kales and G. Zaverucha. Improving the performance of the picnic signature scheme. *IACR TCHES*, pages 154–188, 2020.
22. J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*, pages 525–537, 2018.
23. W. Li, S. Andreina, J.-M. Bohli, and G. Karame. Securing proof-of-stake blockchain protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 297–315. Springer, 2017.
24. 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.
25. G. Zaverucha. Picnic implementation. Available at `https://github.com/microsoft/Picnic`.

A Appendix

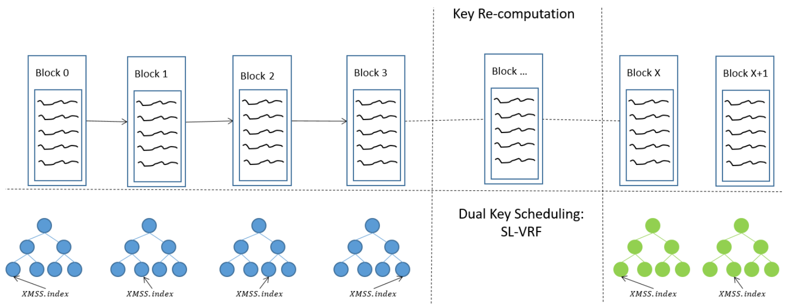

Fig. 1: XMSS/X-VRF state and Blockchain

A.1 Proof of Lemma 1

Proof. Let XMSS.pk be a public key and m be a message. Fix an index $i \in [0, 2^h - 1]$. Also, let $\mathsf{XMSS}.\sigma_1 = (\mathsf{WOTS}^+.\sigma_1, i, \mathsf{XMSS.Auth}^1)$ and $\mathsf{XMSS}.\sigma_2 = (\mathsf{WOTS}^+.\sigma_2, i, \mathsf{XMSS.Auth}^2)$ be two valid signatures created by a PPT adversary on m using XMSS.pk and i. It is clear that $\mathsf{XMSS.Auth}^1 = \mathsf{XMSS.Auth}^2$ as the leaf index and the tree root is the same if the hash function is collision-resistant. We now just need to show that $\mathsf{WOTS}^+.\sigma_1 = \mathsf{WOTS}^+.\sigma_2$, which is true for deterministic XMSS as explained in [6].

$\mathsf{Exp}^{pr}_{\mathcal{A}}$:	$\mathcal{O}\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x)$
1 $\mathsf{pp}_{\mathsf{VRF}} \leftarrow \mathsf{ParamGen}(1^{\lambda})$	return $(\mathsf{y}_{\mathsf{VRF}}, \mathsf{pk}_{\mathsf{VRF}})$
2. $(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}_{\mathsf{VRF}})$	
3. $(x, st) \leftarrow \mathcal{A}_1^{\mathcal{O}\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, \cdot)}(\mathsf{pk}_{\mathsf{VRF}})$	
4. $(\mathsf{y}_{\mathsf{VRF},0}, \cdot) \leftarrow \mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x)$	
5. $\mathsf{y}_{\mathsf{VRF},1} \overset{\$}{\leftarrow} \{0,1\}^{m(\lambda)}$	
6. $\mathsf{b} \overset{\$}{\leftarrow} \{0,1\}$	
7. $\mathsf{b}' \leftarrow \mathcal{A}_2^{\mathcal{O}\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, \cdot)}(\mathsf{y}_{\mathsf{VRF},\mathsf{b}}, st)$	

Fig. 2: Pseudorandomness Experiment

A.2 Proof of Theorem 1

Proof. We prove the three properties of Definition 1.

Correctness. The correctness of X-VRF follows via direct investigation. As long as the underlying XMSS scheme is correct, X-VRF is correct.

Uniqueness. To prove uniqueness of our X-VRF scheme by a reduction to the uniqueness property of the underlying XMSS scheme, we assume $\mathcal{A}_{\text{unq}}$ being an adversary against uniqueness property of our X-VRF scheme. We can construct an adversary $\mathcal{B}_{\text{unq}}$ against the uniqueness property of the underlying XMSS. Let $\mathsf{y}_{\mathsf{VRF}1}, \mathsf{y}_{\mathsf{VRF}2}$ be two different outputs and $\pi_{\mathsf{VRF}1}, \pi_{\mathsf{VRF}2}$ the two respective proofs generated by $\mathcal{A}_{\text{unq}}$ on the same input x. We know that $\mathsf{y}_{\mathsf{VRF}i} = \mathsf{H}(\mathsf{XMSS}.\sigma_i, x)$ and $\pi_i = \mathsf{XMSS}.\sigma_i$ for $i \in \{1, 2\}$. If $\mathsf{y}_{\mathsf{VRF}1} \neq \mathsf{y}_{\mathsf{VRF}2}$, then we must have $\mathsf{XMSS}.\sigma_1 \neq \mathsf{XMSS}.\sigma_2$. Set $m = x$ being the input message of the XMSS.Sign algorithm. Since x is the same in both signatures $\mathsf{XMSS}.\sigma_1$ and $\mathsf{XMSS}.\sigma_2$, it follows that the XMSS signature scheme is not unique, which contradicts the uniqueness property stated in Lemma 1.

Pseudorandomness. Let $\mathcal{A}_{\text{pr}}$ be a PPT adversary against the pseudorandomness of our X-VRF scheme. Recall that $\mathsf{y}_{\mathsf{VRF}} = \mathsf{H}(\mathsf{XMSS}.\sigma, x)$ where H is modelled as a random oracle and $\mathsf{XMSS}.\sigma$ is a signature on x. Also recall that $\mathsf{XMSS}.\sigma$ contains $\mathsf{WOTS}^{+}.\sigma$ which is the (iterated) hash of some completely random and independent n-bit strings unknown to $\mathcal{A}_{\text{pr}}$. So, any $\mathsf{WOTS}^{+}.\sigma$ results in just some random bit string that is contained in $\mathsf{XMSS}.\sigma$. Hence, the only way $\mathcal{A}_{\text{pr}}$ can distinguish $\mathsf{y}_{\mathsf{VRF}}$ from a uniformly random value happens if $\mathcal{A}_{\text{pr}}$ has queried H on the input $(\mathsf{XMSS}.\sigma, x)$, which happens with negligible probability since $\mathcal{A}_{\text{pr}}$ cannot query the signing oracle on x. From here, the pseudorandomness property follows.

A.3 XMSS Signature Scheme

We introduce the concept of XMSS signature from which our VRF is constructed. XMSS is based on the idea of Merkle trees (see Fig. 3) which are binary trees where each nodes is the hash of both its children. Each leaf correspond to the key pair of a One-time digital signature named WOTS^{+}. By definition, a WOTS^{+}

key pair can be used to sign only one message and therefore, each leaf can be only used once. Each signer keep a state $\mathsf{XMSS.idx}$ which is incremented after each signature. A XMSS signature $\mathsf{XMSS}.\sigma$ is composed of a WOTS^+ signature $\mathsf{WOTS}^+.\sigma$, an index i, which indicates the position of the WOTS^+ key pair in the tree and the authentication path $\mathsf{XMSS.Auth}$, which allows to recompute the Merkle root from the WOTS^+ signature to the root. The root is the the XMSS public key $\mathsf{XMSS.pk}$. A simple example is given in Fig. 3 which shows the fourth signatures performed with the XMSS scheme.

Definition 2. *XMSS is defined by a tuple of three algorithms*

$(\mathsf{XMSS.idx}, \mathsf{XMSS.sk}, \mathsf{XMSS.sk}) \leftarrow \mathsf{XMSS.KeyGen}(1^\lambda)$ *: The key generation algorithm on input the security parameter λ outputs a pair consisting of secret and public keys and an index set to* 0 *which is the sate and indicate which leaf to use for a signature. One part of the public key is the root of the tree* $\mathsf{XMSS.root}$ *and the other part is a seed used to compute the bitmask (see Fig. 3).*

$(\mathsf{XMSS}.\sigma) \leftarrow \mathsf{XMSS.Sign}(\mathsf{XMSS.sk}, m, \mathsf{XMSS.idx})$: *The signing algorithm takes as input the secret key* $\mathsf{XMSS.sk}$, $\mathsf{XMSS.idx}$ *and a message m, and outputs a signature* $\mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth})$ *which composed of a* WOTS^+ *signature, the index i that indicates the position of the* WOTS^+ *signature in the tree and the authentication path* $\mathsf{XMSS.Auth}$ *(the grey nodes in Fig. 3)*

$Accept/Reject \leftarrow \mathsf{XMSS.Verify}(\mathsf{XMSS.pk}, m, \mathsf{XMSS}.\sigma)$: *The verification algorithm takes as input the public key* $\mathsf{XMSS.pk} = (\mathsf{XMSS.root}, \mathsf{XMSS.seed})$, *the message m and the signature* $\mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth})$. *It verifies the validity of the* WOTS^+ *signature and then recompute the merkle root r' from the* WOTS^+ *public using the auhtentication path* $\mathsf{XMSS.Auth}$ *and following the direction indicated by i. This outputs Accept iff $r' =$* $\mathsf{XMSS.root}$, *Reject otherwise.*

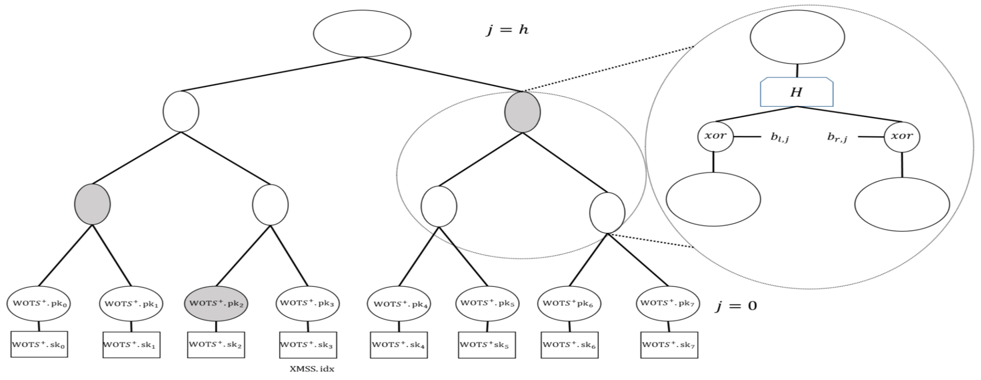

Fig. 3: The XMSS tree construction.