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

Maxime Buser, *Monash University*, Australia
Rafael Dowsley, *Monash University*, Australia
Muhammed F. Esgin, *Monash University* and *CSIRO's Data61*, Australia
Shabnam Kasra Kermanshahi, *RMIT University*, Australia
Veronika Kuchta, *The University of Queensland*, Australia
Joseph K. Liu, *Monash University*, Australia
Raphael Phan, *Monash University*, Malaysia
Zhenfei Zhang, *Manta Network*

***Abstract*—In this work, we study verifiable random functions (VRF) that may resist quantum threats. VRFs have a wide range of applications and play a key role in Proof-of-Stake blockchains, such as Algorand. Our main proposal is a VRF construction X-VRF based on XMSS signature scheme. Our construction is the first quantum-safe VRF proposal based on symmetric primitives. Being based on symmetric-key primitives that have long been studied, X-VRF provides strong confidence that it can withstand quantum attacks in the long run.**

Despite its stateful nature, we empower XMSS with blockchain so that the users do not need to maintain individual states. While increasing the usability of XMSS, our technique also enforces honest behaviour 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 various instances of X-VRF, each may suit a different setting. Our X-VRF instances are the most efficient quantum-safe VRF proposals in the literature. Our extensive performance evaluation, analysis, and implementation indicates the effectiveness of our proposed constructions in practice. In particular, we show that X-VRF can maintain a very competitive throughput close to the existing Algorand protocol and can produce substantially more transactions per second than Bitcoin.

I. Introduction

There is an increasing concern that the security of some classical cryptosystems will be compromised due to advances in quantum computing technologies. It is well known that quantum computers are capable of efficiently solving the mathematical problems underlying the most popular cryptosystems used nowadays, i.e., the integer factorization and discrete logarithm problems. A growing body of literature recognises the importance of post-quantum cryptography, i.e., cryptographic constructions that remain secure even in the face of large quantum computers.

Post-quantum blockchains have received considerable attention due to the severe security impact in case the underlying hardness assumptions are broken. Post-quantum cryptographic primitives relevant for blockchain are mainly designed using lattice-based cryptography [28], multivariate cryptography [13], hash-based cryptography [6] and code-based cryptography [26]. However, most of the developed primitives based on mathematical assumptions offer either signatures or public keys that are orders of magnitude bigger than the currently used ones [27]. Solutions based on symmetric key primitives (e.g. hash-based cryptography) meet three important goals of post-quantum cryptography: efficiency, confidence, and usability [6]. Quantum-resistant symmetric cryptography is not significantly different from the current symmetric cryptography (only the key size needs to be doubled). Most of the fundamental symmetric primitives have been defined and standardized for decades hence boosting confidence in using them. The other aforementioned post-quantum candidates are relatively new and perhaps may be broken when more cryptanalytic efforts to attack them are undertaken [6]. Furthermore, symmetric primitives have been deployed widely and are ready for integration into various applications.

Early blockchain systems (e.g. Bitcoin) are based on Proof-of-Work (PoW), which is 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 first node who solves a difficult computational puzzle is the winner. Due to the costly nature of PoW, an alternative consensus mechanism based on Proof-of-Stake (PoS) has gained popularity [4]. The assumption behind PoS is that the majority of the wealth in the system is controlled by the honest participants (whereas in PoW, it is considered that the majority of the computing power belongs to the honest participants). An important cryptographic primitive that many PoS solutions rely on to provide the required security is Verifiable Random Functions (VRF). VRFs can also be used in different blockchain applications as we describe in later sections.

A. Verifiable Random Function (VRF)

Micali et al. [25] introduced the concept of VRF, a variant of Pseudorandom Functions (PRFs) satisfying a *verifiability* property. That is, the knowledge of a secret key enables one to evaluate the pseudorandom function and to prove the correctness of such evaluations without revealing the secret key or compromising the *pseudorandomness* of the function. 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 this 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*, meaning that under a fixed public key and for one specific VRF input x, there cannot exist valid proofs $\widetilde{\pi_{\mathsf{VRF}}}$ of correct computation corresponding to two *distinct* VRF output values $\mathsf{y}_{\mathsf{VRF}} \neq \widetilde{\mathsf{y}_{\mathsf{VRF}}}$.

B. VRFs in Blockchain

VRFs are widely used in PoS blockchains to conduct secret cryptographic sortition such as to elect block proposers and voting committee members [10], [11], [15], [18], [19], [24] as well as various applications in smart contracts such as online lottery. In this paper we focus on the cryptographic sortition of VRF 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. [15] for Algorand leverages 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.[1] 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 [15]. More precisely, this protocol leverages 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 if she is part of the committee or not [17]:

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 [10] 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 $\mathsf{y}_{\mathsf{VRF}}$ in order to find one that falls within the desired range. Moreover, the committee membership procedure as well as its verification are computationally inexpensive, making the consensus protocol highly scalable.

C. Our Contribution

In this work, we introduce the most efficient and practical post-quantum VRF construction, called X-VRF, built from symmetric primitives only. It is based on the stateful XMSS signature. Inheriting from the statefulness of XMSS, X-VRF is a stateful VRF (essentially a counter-VRF where the signer/verifier needs to store a single counter). In contrast to the classical applications of VRF which may require a stateless construction, for most of the blockchain settings it is often sufficient to employ a stateful primitive. 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 X-VRF construction. Therefore X-VRF in the blockchain environment is actually *stateless* from the point of view of a blockchain user (as the user does *not* need to store an individual state information). X-VRF can then be naturally deployed over Algorand, and utilize Algorand's blockchain as the state counter. We believe this 'blockchain-empowered' XMSS/X-VRF construction can prove useful in other applications as well. We provide a technical overview of our approach in the next section.

We implement our construction under different settings and provide a thorough evaluation analysis and show its efficiency and practicality. 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 tradeoff between (one-time) key generation runtime, memory requirement and the lifetime of a key pair.

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

We further discuss the integration of our VRF into the Algorand blockchain. To support 100 nodes in the blockchain system, our VRF can achieve almost 1000 TPS. If we increase the number of nodes to 1000, our VRF can still achieve around 500 TPS. More importantly, all of our different settings provide a practically long to ultra long key lifetime, ranging from 45 hours to 20 years. When the key lifetime passes, the user just needs to generate a new key pair and broadcast this on blockchain. Compared to the only post-quantum VRF [14], which is a lattice-based construction (denoted as LB-VRF), LB-VRF only supports a few VRF outputs to be generated by a key pair (i.e., it is not a long-term VRF). In the Algorand setting, the key lifetime of LB-VRF is just in the order of seconds, meaning that committee members need to update their keys (almost) every round, which is quite costly.

For a good viewpoint of comparison, we also introduce a naive post-quantum VRF proposal that combines a symmetric-key PRF with a non-interactive zero-knowledge proof (NIZK) of correct evaluation. We pick the NIZK based on the proof system in [23] which also uses symmetric primitives and thus is fair to our comparison. As expected, this proposal, we call SL-VRF, is significantly more costly in terms of computation and communication in comparison to X-VRF while being long-term and stateless. Our experiment shows that SL-VRF is 500 to 5000 times slower than our X-VRF in evaluation and verification, while the proof size is also 25 times larger than our X-VRF.

D. Our Approach

Deterministic XMSS [8] is a good candidate for the construction of a post-quantum VRF as 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^+ [20]. 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 since a user can use different tree leaves (that is, different WOTS^+ keys) to construct a new (and of course different) XMSS signature even 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, which we can use 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 2). 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 tradeoffs between computation, storage and lifetime of an X-VRF key pair. For example, 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.

E. Roadmap

The rest of the paper is organized as follows. In Section II, we recall the important signatures and other cryptographic primitives that are useful for our main constructions. In Section III, we present the "natural" VRF construction - SL-VRF - a stateless long-term verifiable random function from a pseudorandom function and a NIZK protocol. Section IV introduces our main construction - X-VRF, where we prove the security requirements. In Section V, we discuss the implementation results and the evaluation of the two VRF constructions, and compare their performances. In Section VI, we describe the integration of our VRFs into the Algorand blockchain.

II. Preliminaries

In this section we recall the cryptographic primitives WOTS^+ (one-time signature), XMSS (stateful signature), and fundamental definitions used in this paper. For more details on the aforementioned signature schemes we refer to [8], [20]. For a hash function H (defined in Definition 1), let $\mathsf{H}^w(m)$ denote the result of iteratively applying w times the function H with m as the input for the first iteration. $x \xleftarrow{\$} \mathcal{X}$ denotes sampling x uniformly from the domain $\mathcal{X}$ from a random seed or at random.

A. Cryptographic Primitives

Definition 1 (Hash-Function)**.** *A hash function is an efficient function defined by:*

$$\mathsf{H} : \{0,1\}^* \to \{0,1\}^n, D \leftarrow \mathsf{H}(m) \tag{1}$$

that maps an arbitrary-length message m to an n-bit hash value denoted as the digest D. A hash function is expected to satisfy the following security properties:

Preimage Resistance: *For any hash digest D, it is computationally infeasible to find a message m s.t. $\mathsf{H}(m) = D$.*

Collision Resistance: *For any two messages $m_1 \neq m_2$, then $\mathsf{H}(m_1) = \mathsf{H}(m_2)$ with negligible probability.*

Second Preimage Resistance: *Given m, it is computationally infeasible to find $m' \neq m$ s.t. $\mathsf{H}(m') = \mathsf{H}(m)$.*

Definition 2 (Pseudorandom function (PRF))**.** *A function* PRF *defined with a key space $\mathcal{K} \in \{0,1\}^n$:*

$$\mathsf{PRF} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n, y \leftarrow \mathsf{PRF}(k,x) \qquad (2)$$

is a pseudorandom function if the advantage of a polynomial-time adversary A *is* $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{PRF}} | \Pr[\mathcal{A}^{\mathsf{PRF}(k,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]| < negl(\lambda)$ *where $k \leftarrow \mathcal{K}$ and f is a random function such that $f : \{0,1\}^n \leftarrow \{0,1\}^n$*

Definition 3 (WOTS$^+$ [20])**.** WOTS$^+$ *is a Winternitz type one-time signature scheme where a public key is used to sign exactly one message m.* WOTS$^+$ *is parametrized by the Winternitz parameter $w \in \{4, 16, 256\}$ (which determines the time-memory trade-off), the message length $|m|$, $\mathtt{len}_1 = \left\lceil \frac{|m|}{\log(w)} \right\rceil$, $\mathtt{len}_2 = \left\lfloor \frac{\log(\mathtt{len}_1(w-1))}{\log(w)} \right\rfloor + 1$ and $\mathtt{len} = \mathtt{len}_1 + \mathtt{len}_2$, where* len *is the number of n-bit string elements in* WOTS$^+$ *keys and signatures. It is defined by the following three algorithms:*

WOTS$^+$.KeyGen(1^λ) : *The key generation algorithm on input the security parameter λ, outputs a pair of secret and public keys:*

1) *For $n = 2\lambda$, sample $k_1, \ldots, k_{\mathtt{len}} \xleftarrow{\$} \{0,1\}^n$.*
2) *For $i = 1, \ldots,$ len and a hash function* H*, compute $y_i = \mathsf{H}^{w-1}(k_i)$.*
3) *Output* WOTS$^+$.sk $= (k_1, \ldots, k_{\mathtt{len}})$ *and* WOTS$^+$.pk $= (y_1, \ldots, y_{\mathtt{len}})$.

WOTS$^+$.Sign(WOTS$^+$.sk, m)**:** *The signing algorithm on input a message m and a private key* WOTS$^+$.sk $= (k_1, \ldots, k_{\mathtt{len}})$ *outputs a signature* WOTS$^+$.σ*:*

1) *Parse m as $(m_1, \ldots, m_{\mathtt{len}_1})$ with $m_i \in [0, w-1]$ and compute the checksum $C = \sum_{i=1}^{\mathtt{len}_1}(w - 1 - m_i)$. Concatenate the message and the checksum to get $(b_1, \ldots, b_{\mathtt{len}})$ with $b_i \in [0, w-1]$.*
2) *For $i = 1, \ldots,$ len, compute* WOTS$^+$.$\sigma_i = \mathsf{H}^{b_i}(k_i)$*. Output the signature* WOTS$^+$.$\sigma = (\mathsf{WOTS}^+.\sigma_1, \ldots, \mathsf{WOTS}^+.\sigma_{\mathtt{len}})$.

WOTS$^+$.Verify(m, WOTS$^+$.σ, WOTS$^+$.pk)**:** *The verification algorithm takes as input the message m the signature* WOTS$^+$.$\sigma = (\mathsf{WOTS}^+.\sigma_1, ..., \mathsf{WOTS}^+.\sigma_{\mathtt{len}})$ *and the public key* WOTS$^+$.pk*, and proceeds as follows:*

1) *Re-compute the checksum C from the message m to obtain $(b_1, \ldots, b_{\mathtt{len}})$.*
2) *Compute (and output)* WOTS$^+$.pk$' \leftarrow (\mathsf{H}^{w-1-b_1}(\mathsf{WOTS}^+.\sigma_1), \ldots, \mathsf{H}^{w-1-b_{\mathtt{len}}}(\mathsf{WOTS}^+.\sigma_{\mathtt{len}}))$*
3) *Accept if* WOTS$^+$.pk $=$ WOTS$^+$.pk$'$*, reject otherwise*

Remark 1. *The public key* WOTS$^+$.pk *is necessary for* WOTS$^+$.Verify *only if* WOTS$^+$ *is used as an independent signature scheme. In its integration to* XMSS *(see below)* WOTS$^+$.Verify *does not output a Boolean value but directly outputs the re-computed* WOTS$^+$.pk$'$ *(in Step 2) from the signature* WOTS$^+$.σ*. Therefore,* WOTS$^+$.pk *is not needed in the input of* WOTS$^+$.Verify *in* XMSS*.*

Remark 2. *It is easy to see that no PPT adversary can create two* distinct WOTS$^+$ *signatures on the same message m and under the same public key* WOTS$^+$.pk *assuming the hash function $\mathsf{H}(\cdot)$ is collision-resistant. This* uniqueness *property is important for our VRF constructions.*

Definition 4 (XMSS [8])**.** XMSS *is a stateful signature scheme constructed on the idea of a Merkle tree (see Figure 1). It is parameterized by the height h of the tree, the length n (in bits) of a message m and the Winternitz parameter w used in the underlying* WOTS$^+$ *one-time signature scheme. We only consider the deterministic* XMSS *construction, which consists of three algorithms:*

XMSS.KeyGen(1^λ) *: The key generation algorithm on input the security parameter λ outputs a pair consisting of secret and public keys:*

1) *Initialize* PRF.sk $\xleftarrow{\$} \{0,1\}^n$*,* XMSS.seed $\xleftarrow{\$} \{0,1\}^n$ *and* XMSS.idx $= 0$*;*
2) *For $i = 1, \ldots, 2^h$ compute* WOTS$^+$.seed$_i \leftarrow$ PRF(PRF.sk, i) *and use* WOTS$^+$.seed$_i$ *as the random seed to generate* WOTS$^+$.sk$_i$ *and* WOTS$^+$.pk$_i$ *as in* WOTS$^+$.KeyGen*, where* PRF *is a pseudorandom function (see Definition 2 for a formal definition of a* PRF*).*
3) *Set* XMSS.root *to be the Merkle root of the tree generated based on the 2^h* WOTS$^+$ *public keys as leaves and* XMSS.seed *to generate the public bitmask b, which is used as a mask of the children of nodes (an example is illustrated see Figure 1).*
4) *Output* XMSS.idx*,* XMSS.sk $=$ (PRF.sk, XMSS.root, XMSS.seed)*,* XMSS.pk $=$ (XMSS.root, XMSS.seed)*.*

XMSS.Sign(XMSS.sk, m, XMSS.idx) : *The signing algorithm takes as input the private key* XMSS.sk $=$ (PRF.sk, XMSS.root, XMSS.seed)*,* XMSS.idx *and a message m, and outputs a signature* XMSS.σ*:*

1) *Compute* WOTS$^+$.seed$_i \leftarrow$ PRF(PRF.sk, i) *with $i =$* XMSS.idx *and use* WOTS$^+$.seed$_i$ *to generate* WOTS$^+$.sk$_i$ *and* WOTS$^+$.pk$_i$ *as in* WOTS$^+$.KeyGen*.*
2) *Generate the* WOTS$^+$ *signature* WOTS$^+$.$\sigma \leftarrow$ WOTS$^+$.Sign(WOTS$^+$.sk$_i$, m)
3) *Add the authentication path* XMSS.Auth *which is the path from the* WOTS$^+$.pk$_i$*, where $i =$* XMSS.idx*, to the root* XMSS.root*. (In the example in Figure 1* XMSS.idx $= (011)_2 = 3_{10}$ *and the authentication path consists of the grey nodes)*
4) *Set* XMSS.$\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth})$ *for $i =$* XMSS.idx *and increment* XMSS.idx*.*
5) *Output* XMSS.σ*.*

XMSS.Verify(XMSS.pk, m, XMSS.σ) : *The verification algorithm takes as input the public key* XMSS.pk $=$ (XMSS.root, XMSS.seed)*, the message m and the signature* XMSS.$\sigma = (\mathsf{WOTS}^+.\sigma, i, \mathsf{XMSS.Auth})$*, and computes:*

1) *Compute* WOTS$^+$.pk$' \leftarrow$ WOTS$^+$.Verify(m, WOTS$^+$.σ) *(see Remark 1).*

2) Compute a root r' starting from $\mathsf{WOTS}^+.\mathsf{pk}'$ and following the authentication path $\mathsf{XMSS.Auth}$ and the direction indicated by i.
3) Accept if $\mathsf{XMSS.root} = r'$, reject otherwise.

Note that each user keeps an $\mathsf{XMSS.idx}$ value as some state information and increments it every time a signature is generated. However, there is no mechanism to *enforce* that the signer really consumes the leaves in this fashion. The verification still succeeds if a user decides to create the signature using some random leaf.

While WOTS^+ keys can only be used to sign once, XMSS keys can be used to sign 2^h times. For WOTS^+, $|\mathsf{WOTS}^+.\mathsf{pk}| = |\mathsf{WOTS}^+.\sigma| = \mathtt{len} \cdot n$ bits. The public key in XMSS consists of only $2n$ bits, but the signatures are bigger than in WOTS^+. We provide a summary of the parameters in Table I.

Table I
SUMMARY OF PARAMETERS

Parameter	Description
λ	level of post-quantum security
n	size of the hash function output and equal to $2 \cdot \lambda$
w	Winternitz parameter
$\mathtt{len}$	number of n-bit values in WOTS^+ keys and signatures. $\mathtt{len} = \mathtt{len}_1 + \mathtt{len}_2$, with $\mathtt{len}_1 = \left\lceil \frac{\lvert m \rvert}{\log(w)} \right\rceil$ and $\mathtt{len}_2 = \left\lfloor \frac{\log(\mathtt{len}_1(w-1))}{\log(w)} \right\rfloor + 1$.
h	height of the hash tree (i.e., there are 2^h leaves)

Remark 3. *Observe that XMSS as defined above does not satisfy uniqueness as a signer can choose different tree leaves (i.e., different WOTS^+ keys) to sign the same message, leading to different signatures on the same message. We overcome this problem when we design our XMSS-based VRF construction.*

B. Non-interactive Zero-Knowledge Proofs (NIZK)

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation and x, w denote the statement and the witness of this relation, respectively. Let Λ be the corresponding language defined as $\Lambda = \{x \mid \exists w \text{ s.t. } (x,w) \in \mathcal{R}\}$.

Definition 5 (NIZK)**.** *A non-interactive zero-knowledge proof Π_{NIZK} for the relation $\mathcal{R}$ consists of the following probabilistic polynomial time algorithms:*

$\mathsf{Setup}(1^\lambda)$: *This algorithm takes as input the security parameter 1^λ and outputs a verification key vk.*

$\mathsf{Prove}(x, w)$: *On input a statement x and a witness w, this algorithm outputs a proof π.*

$\mathsf{Verify}(x, \mathsf{vk}, \pi)$: *On input a verification key vk, a statement x and a proof π, output either 1 (if the proof is accepted) or 0.*

A NIZK proof must satisfy the following security requirements:

Completeness: *For all $(x,w) \in \mathcal{R}$ if we run $(\mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$ then we have*

$$Pr[\pi \leftarrow \mathsf{Prove}(x,w) : \mathsf{Verify}(x, \mathsf{vk}, \pi) = 1] = 1$$

Soundness: *For all (possibly inefficient) adversaries $\mathcal{A}$, if we run $(\mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$, then we have*

$$Pr[(x,\pi) \leftarrow \mathcal{A}^{\mathsf{Verify}(\cdot,\mathsf{vk},\cdot)}(1^\lambda) : x \notin \mathcal{L} \wedge \mathsf{Verify}(x, \mathsf{vk}, \pi) = 1] = negl(\lambda)$$

Zero-Knowledge: *For all PPT adversaries $\mathcal{A}$ there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, s.t. if we run $(\mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\bar{\mathsf{vk}}, \bar{\tau}) \leftarrow \mathcal{S}_1(1^\lambda)$ then we have*

$$|Pr[\mathcal{A}^{\mathsf{Prove}(\cdot,\cdot)}(1^\lambda, \mathsf{vk}) = 1] - Pr[\mathcal{A}^{\mathcal{S}_2(\bar{\mathsf{vk}},\bar{\tau})}(1^\lambda, \bar{\mathsf{vk}}) = 1]| = negl(\lambda)$$

C. Verifiable Random Function (VRF)

Definition 6 (Verifiable Random Function (VRF) [25])**.** *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 $\mathsf{pp}_{\mathsf{VRF}}$.*

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

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

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

A secure VRF satisfies the following properties:

Provability: *If $(\mathsf{y}_{\mathsf{VRF}}, \pi_{\mathsf{VRF}})$ is the output of $\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x)$, where the public and the secret key are honestly generated, then $\mathsf{Verify}(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF}}, x, \pi_{\mathsf{VRF}})$ outputs YES.*

Pseudorandomness: *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time adversary playing the following experiment $\mathtt{Exp}^{pr}_{\mathcal{A}}$:*

1) $\mathsf{pp}_{\mathsf{VRF}} \leftarrow \mathsf{ParamGen}(1^\lambda)$
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) $\mathsf{y}_{\mathsf{VRF},1} \xleftarrow{\$} \{0,1\}^{m(\lambda)}$
6) $\mathsf{b} \xleftarrow{\$} \{0,1\}$
7) $\mathsf{b}' \leftarrow \mathcal{A}_2^{\mathcal{O}\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}},\cdot)}(\mathsf{y}_{\mathsf{VRF},\mathsf{b}}, st)$

where $\mathcal{O}\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, \cdot)$ is an oracle that on input a value x outputs $\mathsf{y}_{\mathsf{VRF}}$ and π_{VRF}.
Any polynomial-time adversary $\mathcal{A}$ that does not issue any queries to $\mathcal{O}\mathsf{VRFEval}$ on the value x, wins the above game with probability at most $1/2 + negl(\lambda)$.

Uniqueness: *No values $(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF},1}, \mathsf{y}_{\mathsf{VRF},2}, x, \pi_{\mathsf{VRF},1},$*

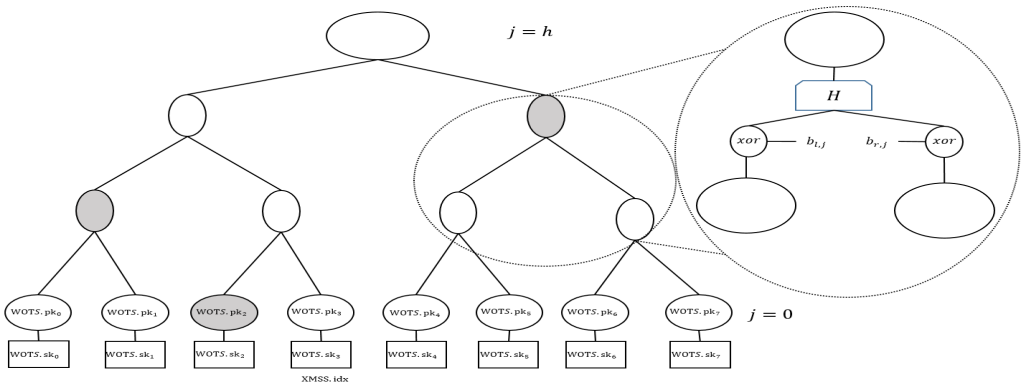

Figure 1. The XMSS tree construction.

$\pi_{\mathsf{VRF},2}$) *can satisfy* $\mathsf{Verify}(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF},1}, x, \pi_{\mathsf{VRF},1}) = \mathsf{Verify}(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{y}_{\mathsf{VRF},2}, x, \pi_{\mathsf{VRF},2}) = 1$, *when* $\mathsf{y}_{\mathsf{VRF},1} \neq \mathsf{y}_{\mathsf{VRF},2}$

Note: In contrast to the *unconditional* uniqueness property defined in [25] 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.

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

We apply the idea of instantiating a VRF from a PRF+NIZK construction which has been introduced in [16]. Recent works like [9], [23] allows one to prove knowledge of a secret key k that generate $y \leftarrow \mathsf{PRF}(k, x)$ where x, y are public information while preserving the secrecy of k, using symmetric primitives only. 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 introduced by Katz et al. (KKW) [23] which is is at the heart of post-quantum security of the digital signature Picnic [9] submitted at the NIST standardization process.

The KKW NIZK protocol introduced in [23] proves the knowledge of a secret key without revealing any information about it with the help of the input and outputs of an arithmetic circuit PRF or a block cipher. KKW is instantiated using the concept of MPC-in-the-head introduced in [21]. MPC-in-the-head simulates a multi-party computation of the PRF 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 [5] which is a cryptographic primitive with low multiplicative complexity, i.e. low number of multiplications in a circuit ("AND" gates).

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 $\mathsf{sk}_{\mathsf{VRF}}$. In other words, taking a message x as an input, a user generates the corresponding pseudo-random outputs $y \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{VRF}}, x)$ and then use the KKW protocol as the VRF proof. KKW protocol is made non-interactive using the Fiat-Shamir transform.

A. SL-VRF *from PRF+NIZK Construction*

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

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{VRF}})$: On input public parameters computes $\mathsf{sk}_{\mathsf{VRF}} \xleftarrow{\$} \{0,1\}^n$.

$\mathsf{VRFEval}(\mathsf{sk}_{\mathsf{VRF}}, x)$: Given $\mathsf{sk}_{\mathsf{VRF}}$ and a message x, the algorithm follows this procedures:

1) $y \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{VRF}}, x)$
2) $\mathsf{pk.check} \leftarrow (x, y)$
3) $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{Prove}(\mathsf{pk.check}, \mathsf{sk}_{\mathsf{VRF}})$. For a relation $\mathcal{R}$ the NIZK proof π_{NIZK} holds iff: $y \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{VRF}}, x)$ (KKW procedure with LowMC block cipher as explained before)

$\mathsf{VRFVerify}(y, x, \pi_{\mathsf{NIZK}})$: 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 NIZK see [23])

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 the fact that 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. The pseudorandomness of the VRF output is inherited from the corresponding pseudorandomness of the PRF.

In order to evaluate the efficiency of our SL-VRF, we need to analyse the the underlying NIZK proof construction. A detailed evaluation is provided in Section V.

IV. X-VRF: Verifiable Random Function from XMSS

In this section we introduce a construction of a secure VRF from XMSS. 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 is able to 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, and then she will be able to amplify his 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 does not have the ability to choose between multiple leaves and can only produce a single signature output on a message.

A. X-VRF *from* XMSS *Construction*

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

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{VRF}})$: 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 XMSS.Auth are inconsistent (i.e., if the leaf index indicated by 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 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 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 $K \bmod N$, 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 2 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.

B. X-VRF *Security Analysis*

The most critical property we need to analyze is the uniqueness. Therefore, before proving the security of X-VRF construction, we first focus on the uniqueness of XMSS under the constraint that the index used to create the signature is the same.

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 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).*

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 $\mathsf{WOTS}^+.\sigma_1 = \mathsf{WOTS}^+.\sigma_2$. This follows immediately from Remark 2. □

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.*

Proof. We prove the three properties of Definition 6.

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. □

V. 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.

A. Implementation Setup

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 taken from [2]. It couples KKW [23] with the Fiat-Shamir transform to get a NIZK. The implementation of X-VRF is derived from the XMSS implementation provided in [3]. Both implementations were deployed on a machine with a Intel(R) Core i7-86500 CPU @ 1.90GHz 12GB of RAM.

VRF Proof Sizes. Table II summarizes the proof size of each instance. As expected, the X-VRF constructions clearly outperform SL-VRF with proof sizes at least 16.1 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 of the size of the NIZK proof [22]. The size that we presented in Figure II differs from the one provided by Katz et al. [23], and the reason behind this is that we used the parameter of the NIST submission presented in [22] which have been optimized. 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], [23] for further details.

B. Memory Requirements

In Table II, we propose applicable memory requirements for our four instances of X-VRF. It is important to know that the memory capacity impact 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 1). 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 1, 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 II. This advantage to have cheap offline computations ($\ll$ 1 ms) needs to be done for only half of the X-VRF evaluations and most importantly it requires a maximum of 4GB of memory, which can be considered to be acceptable and be applicable to lightweight devices.

C. 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, re-

Table II
PERFORMANCE EVALUATION OF X-VRF, SL-VRF, ECVRF AND LB-VRF. FOR LB-VRF, WE REPORT THE RESULTS PROVIDED IN [14]. 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 requirement 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	32 B	3.32 KB
SK size	32 B	132 B	132 B	132 B	132 B	32 B	0.45 KB
Proof size	80 B	2.63 KB	2.76 KB	2.88 KB	3.01 KB	48.7 KB	4.94 KB
KeyGen	0.05 ms	48.9 s	14.2 min	3.73 h	$\approx$ 58 h	0.05 ms	0.33 ms
Eval	0.10 ms	0.72 ms	0.75 ms	0.78 ms	0.80 ms	873.8 ms	3.1 ms
Verify	0.10 ms	0.87 ms	0.91 ms	0.94 ms	0.97 ms	498.3 ms	1.3 ms

spectively, at most 2^{15}, 2^{19}, 2^{23} and 2^{27} VRF evaluations. Table II summarizes the performance of each instance. When it comes to the key generation procedure SL-VRF as expected outperforms all X-VRF instances, as 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 the stateful construction X-VRF. Eval of SL-VRF requires the simulation of MPC computation, which is quite costly.

Regarding the performance of the evaluation algorithm of X-VRF instances the performance of X-VRF is really competitive. The reason behind 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 $(w-1) \cdot \mathtt{len}$ calls of the cryptographic hash function. Our results demonstrates that X-VRF is at least 1092 times faster than the naive SL-VRF for the VRF evaluation.

The VRF verification of X-VRF also outperforms SL-VRF by at least 497 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 [22]), while X-VRF needs only a maximum of $h + w \cdot \mathtt{len} + \log \mathtt{len}$ 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.

VI. 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* pre-determined 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.

A. Performance Estimation

To illustrate our benchmark results better, 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 are not included in the final blocks by design - the committee members only attest that they have seen enough votes, without putting those 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 networks throughput limitation, rather than the actual blocksize. To summarize, we follow [14] and estimate the Algorand TPS throughput as follows:

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

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

Table III
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	923	1015	1000
	Rainbow	1008	1008	1008	1007	922	1013	998
	SPHINCS$^+$	34	34	34	34	31	34	32
50	Ed25519	990	989	988	987	557	1014	939
	Rainbow	988	987	986	985	556	1012	937
	SPHINCS$^+$	33	33	33	33	19	34	22
100	Ed25519	966	963	961	958	100	1014	862
	Rainbow	964	961	959	957	99	1012	861
	SPHINCS$^+$	32	32	32	32	3	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

signature schemes that are used to authenticate transactions. In this computation, we make the following assumptions as in [14] 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[2] [12] -the shortest signature finalist candidate in NIST's Post-Quantum Cryptography standardization process- whose signature size is as small as 66 bytes[3], and (ii) SPHINCS$^+$ [7], 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 III, 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 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.

To the best of our knowledge, the only other *practical* post-quantum VRF is provided in [14] using lattice-based techniques. Table III refers to this construction as LB-VRF and presents its expected TPS in Algorand as given in [14]. Our results show that all four X-VRF instances outperform LB-VRF when it comes to TPS for all node sizes. As the number of nodes increases, the advantage of our constructions increases. Another disadvantage of LB-VRF is that users need to update their keys *every* round. Our X-VRF construction, on the other hand, can support the use of the same key pair even for many years.

B. Dual Key Scheduling

Now that we know our X-VRF provides a much 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 rest of the 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

[2]https://www.pqcrainbow.org/

[3]We note that the signature length of 48 bytes of an earlier Rainbow version is used in [14].

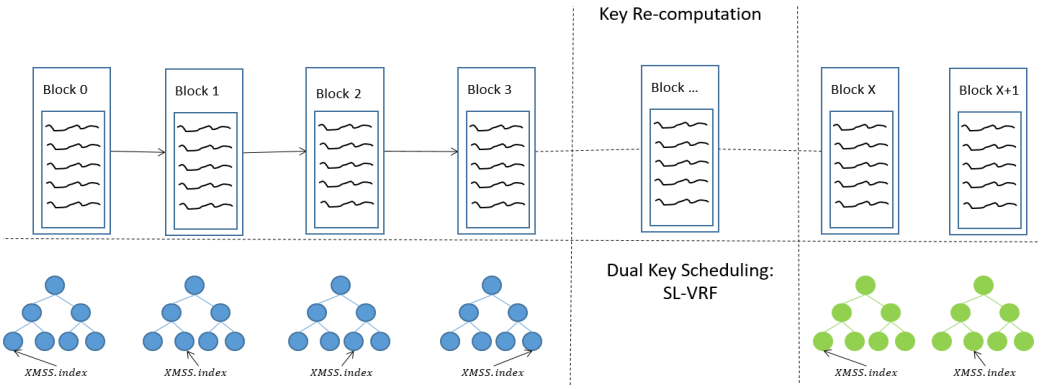

Figure 2. XMSS/X-VRF state and Blockchain

where users may lose their keys. For conservative purpose, 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 Figure 2).

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 that they 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 for them 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 III. Since this dual VRF is an orthogonal direction from this paper, we leave the rigorous analysis to future work.

C. X-VRF *Instances*

As explained previously, we propose four different instances of X-VRF and there are the only applicable constructions to Algorand's 1000-node setting as the results in Table III 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), 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 of 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 III illustrates the expected TPS for each of X-VRF instances, and as explained previously the best performance is achieved with 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 V-B the ideal memory requirement to achieve a balance between offline computations and memory consumption (See Table II). 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 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 tradeoff between TPS and recurrence of key updates. Moreover, our proposition of dual key system by coupling X-VRF with SL-VRF combines the best of two worlds.

VII. Conclusion

In this paper, we introduced "post-quantum" verifiable random functions based on symmetric primitives. Our XMSS-based X-VRF proposals support the highest number of transactions per second in a "post-quantum" PoS-based consensus protocol. Being based on long-studied symmetric primitives, all X-VRF instance provide strong security assurances while also being highly efficient.

References

[1] Algorand-what we do. Available at https://www.algorand.com/what-we-do/faq.

[2] Picnic implementation. Available at https://github.com/microsoft/Picnic.

[3] Xmss implementation. Available at https://github.com/XMSS/xmss-reference.

[4] Proof of stake instead of proof of work, July 2011. Available at https://bitcointalk.org/index.php?topic=27787.0.

[5] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for mpc and fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 430–454. Springer, 2015.

[6] D. J. Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[7] 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, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2129–2146. ACM, 2019.

[8] 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.

[9] 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 *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

[10] B. David, P. Gaži, 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.

[11] A. S. de Pedro, D. Levi, and L. I. Cuende. Witnet: A decentralized oracle network protocol. *arXiv preprint arXiv:1711.09756*, 2017.

[12] J. Ding and D. Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *ACNS*, volume 3531 of *LNCS*, pages 164–175, 2005.

[13] J. Ding and B.-Y. Yang. Multivariate public key cryptography. In *Post-quantum cryptography*, pages 193–241. Springer, 2009.

[14] 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. Cryptology ePrint Archive, Report 2020/1222, 2020. https://eprint.iacr.org/2020/1222 (to appear at FC'21).

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA, 2017. Association for Computing Machinery.

[16] S. Goldwasser and R. Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent. In *Annual International Cryptology Conference*, pages 228–245. Springer, 1992.

[17] S. Gorbunov. Algorand releases first open-source code: Verifiable random function. https://medium.com/algorand/algorand-releases-first-open-source-code-of-verifiable-random-function-93c2960abd61, 2018.

[18] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[19] L. Hellebrandt, I. Homoliak, K. Malinka, and P. Hanáček. Increasing trust in tor node list using blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 29–32. IEEE, 2019.

[20] A. Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In *Progress in Cryptology - AFRICACRYPT*, volume 7918 of *LNCS*, pages 173–188. Springer, 2013.

[21] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

[22] D. Kales and G. Zaverucha. Improving the performance of the picnic signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 154–188, 2020.

[23] J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures.

In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 525–537, 2018.

[24] 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.

[25] 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.

[26] R. Overbeck and N. Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.

[27] M. Raikwar, D. Gligoroski, and K. Kralevska. Sok of used cryptography in blockchain. *IEEE Access*, 7:148550–148575, 2019.

[28] O. Regev. Lattice-based cryptography. In *Annual International Cryptology Conference*, pages 131–141. Springer, 2006.