Key Updatable Hash Based VRF

Suman Ghosh, Ratna Dutta and Sourav Mukhopadhyay

Department of Mathematics, Indian Institute of Technology Kharagpur
Kharagpur, 721302, India
{sg7@kgpian,ratna@maths,sourav@maths}.iitkgp.ac.in

Abstract. Unbiased, unpredictable, and publicly verifiable randomness is essential for a wide range of blockchain-based Web3 applications. Verifiable Random Functions (VRFs) naturally satisfy these requirements. For practical deployment, however, a VRF scheme must support efficient key generation and allow multiple evaluations across different blockchain rounds.

In this work, we present a post-quantum secure, key-updatable VRF construction built from symmetric cryptographic primitives, including hash functions and pseudorandom generators (PRGs). The core of our design is a quantum-secure Extended Merkle Signature Scheme (XMSS) structured over multiple layers. We reorganize the XMSS framework in a systematic way to integrate it seamlessly into our VRF construction. Compared to existing approaches, our scheme offers improved key generation efficiency while enabling multiple evaluations from a single secret–verification key pair.

Keywords: Post-Quantum · Verifiable Random Function · Hash · Merkle Tree · Blockchain · Algorand.

1 Introduction

A Verifiable Random Function (VRF) allows the generation of authenticated randomness that can be efficiently and publicly verified. This was put forth by Micali, Rabin and Vadhan [42] in 1999. VRFs are utilized as a fundamental component in numerous applications including Domain Name System Security Extensions (DNSSEC) protocol [27], Web3-based applications such as decentralized finance and GameFi [44] and in blockchain consensus protocols [17, 25, 16] to establish Proof-of-Stake to supply verifiable random inputs essential to the protocols.

VRF operates through four main algorithms. First, public parameters are generated using algorithm $\mathsf{ParamGen}(1^\lambda)$, which sets up the public parameter pp based on a security parameter λ. A user can run algorithm $\mathsf{KeyGen}(\mathsf{pp})$ to generate a key pair comprising a secret key sk that is kept private to the user and a verification key vk that is published. Using sk, the user executes algorithm $\mathsf{Eval}(\mathsf{pp}, \mathsf{sk}, x)$ on an input x to obtain a pseudorandom output y and a proof π showing that y is correctly computed. Finally, verifier calls algorithm $\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, x, y, \pi)$ to check the validity of the output y and the proof π using

the verification key vk, pp and input x. If verification succeeds, y is accepted as the correct VRF output; otherwise, it is rejected. A secure VRF should satisfy two key properties - *uniqueness* and *pseudorandomness*. Uniqueness means that for any given input and verification key, there exists only one valid VRF output that can be accepted by the verification algorithm. It ensures that no one can produce two different valid outputs for the same input. Pseudorandomness means that the VRF output appears completely random to anyone who does not possess the secret key. In other words, without the secret key, the output is indistinguishable from a truly random value.

BLS-VRF [9, 8] and ECVRF [43] are the most common VRF constructions which rely on cryptographic pairings and elliptic curves respectively and are vulnerable to adversaries equipped with quantum computers. In blockchain systems such as Algorand [25], VRFs are employed to guarantee the honest selection of committee members for all finalized blocks. Each user independently runs a VRF using their secret key and the current round information as input; if the VRF output is below a threshold determined by their stake, the user is selected as a committee member for that round. A newly joining participant, even without access to prior block history, can ascertain the validity of the chain by examining the recorded votes, assuming the continued security of the VRF. In such protocols, since each block is approved by a majority of committee members, forbidding blockchain forks. A blockchain fork occurs when the chain splits into two due to conflicting blocks or protocol changes. It's undesirable because it can cause inconsistencies, double-spending risks, and loss of consensus among network participants. However, if the VRF's security is compromised, an adversary could manipulate the selection of committee members for any round (including past rounds) allowing them to alter history and potentially create forks in the subsequent blockchain. Future quantum threats to the integrity of VRFs are significant concern today. This underscores the necessity of designing VRF schemes that are compact, efficient and resilient against quantum adversaries.

Given the rising significance of quantum-resistant cryptography, research efforts have increasingly targeted the development of blockchain tools that remain secure against quantum attacks. LB-VRF, proposed in 2021 by Esgin et al. [22] is the first quantum-safe VRF scheme based on lattices, which is considered as a significant step toward practical post-quantum VRFs. However, it has a major limitation, as it requires continuous key updates. Each key pair can generate only one VRF output, which introduces additional complications to compensate for the restricted VRF functionality. In 2022, Buser et al. [13] designed X-VRF, a quantum-resistant VRF scheme built upon hash functions that circumvent the problem of LB-VRF at the expense of huge key generation time.

1.1 Our Contribution

Motivated by the work of Buser et al. [13], we concentrate on reducing the key generation time while allowing multiple VRF outputs from a single key pair. We propose a post-quantum VRF construction that achieves faster key generation while enabling the production of multiple VRF outputs from a single key pair.

We introduce XM-VRF, an efficient post-quantum key updatable VRF construction employing hash-based cryptography which is one of the latest families of post-quantum candidates renowned for its compactness. To be more precise, we integrate in our XM-VRF a multi-tree version of the post-quantum secure Extended Merkle Signature Scheme (XMSS) of [11] in our XM-VRF construction. In XMSS, each leaf of the main Merkle tree is generated using an L-tree. Each L-tree accumulates the l components of a WOTS$^+$ [33] public key into a single value and accumulated at it's root using a hash function. At each level of the Merkle tree and L-tree, random bitmasks are applied to the hash inputs, adding an extra layer of randomness and strengthening the security of the tree structure.

Efficient key generation. The primary benefit of our XM-VRF over existing solutions is its improved key generation time and its capability to produce unlimited VRF outputs from single key pair. Table 2 presents a comparison of our XM-VRF with X-VRF [13] and other practical post-quantum VRFs. The main idea to reduce the key generation time in our construction is to use the reduced height XMSS trees in two layers and then fix the upper layer tree and keep changing the lower layer trees. The lower XMSS tree signs the input as a message while the upper XMSS tree signs the root node of the lower XMSS tree. The root node of the upper XMSS tree is embedded in the verification key of XM-VRF. Computing the root node of the top XMSS tree requires less time than computing the root node of the full-height XMSS tree. The verification key of XM-VRF includes this root node, and the key generation efficiency of XM-VRF is achieved by substantially reducing the time needed to compute the root node of the top XMSS tree.

Forward security. To ensure the security of prior rounds even if a user key is later compromised, blockchain protocols such as Algorand [26] rely on VRF schemes and Ouroboros Praos [18] rely on key evolving digital signatures. Generic constructions for achieving forward security can be seamlessly instantiated in the post-quantum setting by replacing these signatures. Our XM-VRF ensures forward security by employing a forward secure stateful pseudorandom generator (PRG) that updates its internal state after each use, preventing compromise of past outputs and thereby making it suitable for blockchain setting. A detailed description of the forward-secure stateful pseudorandom generator (PRG) is given in Appendix C.

Key-updatable VRF: Our XM-VRF autonomously updates its secret key during the algorithm Eval, performing this update in parallel with the VRF output generation to ensure uninterrupted state progression. This property enables continuous and efficient VRF evaluation without explicit key refresh operations, thereby reducing computational overhead and ensuring forward security across multiple evaluations. We use a stateful forward-secure PRG to generate the WOTS$^+$ key pairs sequentially, thereby eliminating the need to store all key pairs in memory.

Uniqueness of XM-VRF: At first glance, the core structure may appear to be simply a multi-tree version of XMSS applied to a VRF, but we introduce several substantial changes to the underlying primitives in order to achieve stronger security guarantees. Bodaghi et al. [6] proposed an attack against the uniqueness property of WOTS$^+$, which in turn leads to uniqueness violations in both XMSS and X-VRF. Hence, a straightforward integration of a multi-tree variant of XMSS into a VRF construction will suffer from uniqueness failure. We alter the function family used in WOTS$^+$ so that WOTS$^+$ regains the uniqueness property. In XM-VRF, we instantiate this modified WOTS$^+$ within the XMSS framework, thereby ensuring the uniqueness of both XMSS and XM-VRF. Additionally, handling the state management for the various layer-specific XMSS trees is difficult, and we addressed this aspect in great detail.

Flexible and Dynamic construction: Our XM-VRF is highly flexible; specifically, it includes a parameter d that can be set based on our specific needs. If we require very fast key generation, we select a large d, but then the XMSS trees must be updated more frequently. Alternatively, by choosing a smaller d, we reduce how often XMSS tree updates are needed. Our XM-VRF uses a dynamic structure. In particular, we refresh the lower XMSS trees and link them to a fixed upper XMSS tree, enabling an unbounded number of evaluations from a single secret–verification key pair.

Protocol for random committee selection in Algorand: Algorand [25] is a proof-of-stake blockchain achieving fast, secure, and decentralized consensus, where network participants collectively agree on the next valid block. Each user independently runs a VRF using their secret key and the current round information containing round number, phase number and random seed as input; if the VRF output is below a threshold determined by their stake, the user is selected as a committee member for that round. We present a protocol for random committee selection within the Algorand blockchain framework, leveraging XM-VRF to ensure unbiased, verifiable and stake-proportional participant selection.

Comparison of XM-VRF with other existing post quantum VRFs: We present the comparison in Table 2 which summarizes the efficiency and security features of the proposed XM-VRF alongside other post-quantum VRF constructions. We make this comparison based on the following key features:
OutputCount refers to the number of VRF outputs that can be generated using a single secret-verification key pair.
KeyLifeTime represents the duration for which all VRF outputs derived from a single secret–verification key pair remain valid. It is measured based on the block generation interval of the Algorand blockchain, indicating how long a key pair can be securely used before renewal. Since Algorand produces a new block approximately every 5 seconds, we assume that each VRF output is valid for a single block. Therefore, we set KeyLifetime to $\alpha = 5$ seconds for each VRF output.
KeyGenTime denotes the time required to generate the secret–verification key pair for the VRF.

ProofSize represents the combined size of all components that constitute the proof accompanying a VRF output, reflecting the communication overhead during verification.
SecurityBasis denotes underlying cryptographic hardness assumptions that form the foundation of the VRF's security.
KeyUpdatable indicates whether the VRF key pair is updated after each evaluation (positive) or remains constant across all output generations (negative).
We now define several key terms used in the comparison to provide a clearer understanding of the parameters and metrics considered in evaluating different VRF constructions.
$\mathsf{XMSSTree}(2^n)$ denotes the time to compute XMSS tree with 2^n leaf nodes
$\mathsf{MerkleTree}(2^n)$ denotes the time to compute Merkle tree with 2^n leaf nodes

	XMSS Tree	Merkle Tree
ComputationCost	$2^n(l+1)-1$ Hashes	$2^{n+1}-1$ Hashes
AuthenticationPathSize	n strings from $\{0,1\}^\lambda$	n strings from $\{0,1\}^\lambda$

Table 1. Comparison of XMSS Tree and Merkle Tree of height n, l is the WOTS$^+$ key size

p denotes the height of the hash chain in iVRF construction, $t_{\mathsf{MatrixMult}}$ denotes the time to compute a matrix multiplication of dimension $(m+c+1)\times m$, t_{Hash} denotes the time to compute a hash function from $\{0,1\}^*$ to $\{0,1\}^{2\lambda}$, t_{LowMC} denotes the time to compute LowMC block cipher.

	X-VRF [13]	LB-VRF [22]	iVRF [21]	SL-VRF [13]	XM-VRF [Our Work]
OutputCount	2^n	1	2^n	practically unlimited	practically unlimited
KeyLifeTime	$\alpha.2^n$ sec	α sec	$\alpha.2^n$ sec	practically unlimited	practically unlimited
KeyGenTime	$\mathsf{XMSSTree}(2^n)$	$t_{\mathsf{MatrixMult}}$	$\mathsf{MerkleTree}(2^n)+n.p\ t_{\mathsf{Hash}}$	t_{LowMC}	$2.\mathsf{XMSSTree}(2^{n/d})$
ProofSize	1 WOTS$^+$ signature + 1 XMSS Authentication Path + 1 index	1 element of ring $\mathcal{R}+1$ element of ring $\mathcal{R}^{m+c+1}$	1 binary string + 1 Merkle Authentication Path	1 KKW NIZK proof	2 WOTS$^+$ signature + 2 XMSS Authentication Path
SecurityBasis	Collision resistant Hash	Module LWE and Module SIS	Collision resistant Hash	Onewayness of LowMC block cipher	Collision resistant Hash and Collision resistant function family
KeyUpdatable	✗	✗	✗	✗	✔

Table 2. Comparison of the proposed XM-VRF with other existing post quantum VRFs in terms of efficiency, security features, and key generation time. n: Height of tree for tree based VRFs such as X-VRF, iVRF, α: Life time of a VRF output in Algorand blockchain (Assuming that each VRF evaluation is valid for a single block), n/d: The height of XMSS tree in each layer of XM-VRF.

In SL-VRF [13] the evaluation relies on a PRF, ensuring pseudorandom outputs, while uniqueness and verifiability are achieved through the KKW NIZK [36] proof system. So SL-VRF can generate different VRF output from different input using the same key pair. Consequently the OutputCount of SL-VRF is practically unlimited so the KeyLifetime of SL-VRF is huge. The KeyGenTime of SL-VRF is quite low, since it only requires a single evaluation of the LowMC block cipher to generate the key pair. But it's proof uses KKW NIZK, which has 20 times more than other VRF's proof size. LB-VRF [22] achieves efficient KeyGenTime since it requires only a single matrix multiplication during key generation. However, its KeyLifetime and OutputCount are very limited, as it can produce only one VRF output per secret–verification key pair. The proof of LB-VRF contains an element of the polynomial ring $\mathcal{R} = \mathbb{Z}_q/(X^d+1)$ and an element of the ring $\mathcal{R}^{m+c+1}$ where m is the rank of Module-SIS (Short Integer Solution), c is the rank of Module-LWE (Learning With Errors). So ProofSize of LB-VRF comparatively smaller. Now if we notice the hash based VRF constructions such as X-VRF [13], iVRF [21] and our work XM-VRF. The OutputCount remains identical for iVRF and X-VRF Therefore, they share the same KeyLifetime. But OutputCount of XM-VRF is practically unlimited as it is dynamic in nature and generates unlimited evaluations per secret-verification key pair. Key generation of iVRF need time to compute a whole Merkle tree of height n and also have to compute some hash functions from $\{0,1\}^*$ to $\{0,1\}^{2\lambda}$. Key generation of X-VRF requires time to compute a whole XMSS tree of height n which is much bigger than that of iVRF as in XMSS we have to compute L-tree for each leaf nodes. But KeyGen of XM-VRF need time to compute only the 2 XMSS tree of height $\frac{n}{d}$ in parallel, which is the smallest from these two hash based VRF construction while also providing more KeyLifetime. Observing the computation cost of the XMSS and Merkle trees from Table 1, XM-VRF have to compute only $2 \cdot 2^{n/d}(l+1) - 2$ hashes which is less than that of iVRF and X-VRF. For instance if we consider $n = 40, l = 10$ and $d = 4$ then KeyGen of XM-VRF have to compute $22 \cdot 2^{10} - 2$ hashes but KeyGen of X-VRF must compute $11 \cdot 2^{40} - 1$ hashes and KeyGen of iVRF must compute atleast $2^{41} - 1$ hashes. The proof of X-VRF contains one WOTS$^+$ signature, one XMSS tree authentication path and an index, proof of iVRF contains one binary string and a Merkle authentication path and proof of XM-VRF contains 2 WOTS$^+$ signature and 2 XMSS authentication path. In iVRF the Merkle tree of height n results in Merkle authentication path comprising n nodes. Similarly in X-VRF the XMSS tree of height n also yields XMSS authentication path containing n nodes. In XM-VRF we uses 2 XMSS tree of height $\frac{n}{d}$ which implies $2 \cdot \frac{n}{d}$ nodes. Thus, we can observe that the proof sizes of XM-VRF is smaller than these two VRF constructions and much smaller than the KKW NIZK [36] proof of SL-VRF. Hence, in terms of communication efficiency, the proposed XM-VRF achieves a standard and practical proof size, comparable to other lightweight post-quantum VRF constructions. If we examine the security of these VRF constructions, we see that X-VRF, iVRF, and XM-VRF all rely on the hardness of a collision-resistant hash function modeled in the random oracle setting. Moreover, the uniqueness

of XM-VRF additionally depends on the hardness of the underlying collision-resistant function family employed in its construction. The security of LB-VRF is based on the hardness assumptions of the Module-LWE and Module-SIS problems, whereas SL-VRF derives its security from the one-wayness property of the LowMC [2] block cipher. Focusing on the KeyUpdatable property, the X-VRF, LB-VRF, iVRF, and SL-VRF constructions generate a secret–verification key pair during KeyGen, which remains unchanged across all VRF evaluations. So these VRF constructions are not key updatable so we denote these with ✗ symbol. But our XM-VRF is a key updatable VRF as it updates the secret key after each evalution and we denote it with ✔ symbol. Recently, several isogeny-based post-quantum VRF constructions [37, 38] have been proposed; however, we do not include them in our comparison since their underlying assumptions and efficiency characteristics differ significantly from our XM-VRF framework. There also exist several classical (pre-quantum) VRF constructions [40, 19, 34, 1] but we exclude them from our comparison as our focus is solely on post-quantum secure VRF schemes.

1.2 Technical Overview

We outline below our technical approach in constructing a post-quantum secure VRF that ensures pseudorandomness, uniqueness and efficient key generation with ability to produce unlimited VRF outputs from a single key pair.

Merkle Tree: A Merkle tree [41] is a binary structure that enables efficient verification and accumulation of large datasets through hashing. Each leaf stores a data hash, internal nodes hash their children, and the root acts as a compact commitment. An authentication path includes sibling nodes from a leaf to the root for efficient verification.

XMSS: Our proposed XM-VRF construction is built upon the Extended Merkle Signature Scheme (XMSS) [11] which serves as its core cryptographic component. XMSS enables multiple VRF evaluations to be derived from a single secret-verification key pair of VRF. Specifically, XMSS tree contains multiple one-time signature WOTS$^+$ [33] public keys, with each leaf node corresponding to one WOTS$^+$ public key. The corresponding WOTS$^+$ secret keys are used for signing, which in turn yield multiple VRF outputs. To compute the leaf nodes from the WOTS$^+$ keys, we employ a tree structure L-tree. L-tree takes the l components of a WOTS$^+$ public key as its leaf nodes and accumulate them into a single hash value. The root of the upper XMSS tree is stored in XM-VRF verification key. The XMSS tree is generated through the algorithm TreeHash which incrementally builds the tree in a bottom-up manner. During this process, a stack is utilized to manage intermediate nodes, enabling efficient computation and storage of partial tree information without requiring the entire tree to be held in memory. The partial tree information is maintained within stacks and auxiliary data structures to facilitate efficient computation of authentication paths corresponding to specific indices. A stateful forward-secure PRG is employed to generate the WOTS$^+$ key pairs sequentially, thereby eliminating the need to

store all key pairs in memory. Specifically, a random seed is chosen and the PRG is iteratively applied to this seed to produce subsequent WOTS$^+$ secret keys and the necessary random strings. A chaining function is then applied to these secret keys to obtain the corresponding WOTS$^+$ public keys. Only the initial seed needs to be stored, significantly reducing memory requirements.

Multi layer XMSS: However, employing an XMSS tree of large height introduces a significant challenge, as computing the VRF verification key necessitates deriving the root node of the entire tree. For an XMSS tree of height h, this requires generating 2^h WOTS$^+$ key pairs and constructing the complete tree, which consequently leads to substantial key generation time and rendering the approach impractical for blockchain applications. To address this, we employ XMSS trees of smaller height arranged across multiple layers. More precisely, we split XMSS trees of height h into 2 layers of XMSS trees of height $\frac{h}{d}$. Therefore, instead of constructing a single XMSS tree of height h, we compute 2 smaller XMSS trees, each of height $\frac{h}{d}$ which can be generated in parallel. Since XMSS tree in each layer are independent to each other, now we compute only $2^{h/d}$ WOTS$^+$ key pairs in 2 layers in parallel to complete the key generation process. These XMSS trees in each layer are connected through WOTS$^+$ signatures such as WOTS$^+$ signatures on the root nodes of lower-layer XMSS trees are computed using the WOTS$^+$ secret key associated with leaf node of the XMSS tree in the upper layer. And we store this WOTS$^+$ signatures in each layer as σ_2 in secret key of XM-VRF. For computing authentication paths of the XMSS trees we store partial information of the current and next trees in states such as $\mathsf{St}_{\mathsf{cur},1}, \mathsf{St}_{\mathsf{cur},2}, \mathsf{St}_{\mathsf{next}}$ inside secret key of XM-VRF. During the evaluation, we update these states to obtain authentication paths efficiently. In the evaluation process , we also computing new WOTS$^+$ key pairs in lower layer to construct the next tree. And once XM-VRF have used all the $2^{h/d}$ WOTS$^+$ keys in the current XMSS tree then we change XMSS trees in lower layer and update the required informations, signatures. During the key generation we compute a single XMSS tree in each layer and we denote it as current tree in each layer. During evaluation, we begin constructing the next set of XMSS trees in lower layer. Once the current XMSS tree in lower level is fully utilized, all relevant information is transitioned to the new tree, while the corresponding procedures for key management and updates continue smoothly. We then set the next trees as current trees in lower layer XMSS tree and again we compute next tree for it and so on. The next tree in lower layer will be connected to the next leaf node of the upper layer XMSS tree. Once all the leaf nodes of the upper layer XMSS tree have been used, we return to the first leaf node and repeat the same process for the newly added new lower layer XMSS trees. Hence, our approach achieves efficient key generation in the XM-VRF scheme. Since each current tree will provide us $2^{h/d}$ VRF outputs so after repeating this current and next tree update process continuously we can get unlimited VRF outputs.

Key-updatable construction: Since our construction allows multiple VRF evaluations from a single secret–verification key pair, distinct WOTS$^+$ secret

keys are required for each evaluation. To derive a specific WOTS$^+$ secret key from the initial seed, we employ a stateful forward-secure PRG. The resulting key information is then incorporated into the XM-VRF secret key and updated accordingly. The secret key of XM-VRF contains the random seeds for the current and next XMSS trees in each layer, as $\mathsf{St}_{\mathsf{cur},1}, \mathsf{St}_{\mathsf{cur},2}, \mathsf{St}_{\mathsf{next}}$. Additionally, WOTS$^+$ signature is included in the secret key of the XM-VRF to establish connections between XMSS trees at two layers. This information is also updated whenever the XMSS trees at each layer are refreshed after evaluation.

1.3 Related Works

Initially proposed by Micali, Rabin, and Vadhan, [42], VRFs have emerged as a fundamental cryptographic primitive with diverse applications. In [42], Micali et al. build VRF upon the Goldreich–Goldwasser–Micali pseudorandom function construction to demonstrate a link between unique signatures and VRFs, combining the inherent unpredictability of unique signatures with their verifiability. The notion of VRF has been further explored by Lysyanskaya et al.[40] in 2002 and Dodis et al.[20] in 2005 each building on distinct number-theoretic assumptions. Lysyanskaya et al. [40] construct a Verifiable Random Function (VRF) from unique signature scheme by performing the transformation introduced by Micali et al. [42]. Additional VRF constructions based on number-theoretic assumptions are presented in [19, 32, 34, 1, 5]. In [10], Brakerski et al. introduced the concept of a weak VRF in which pseudorandomness is guaranteed only for randomly chosen inputs. Additional primitives related to VRF such as constrained VRF, simulatable VRF have been introduced in [15, 24].

In the context of quantum-safe constructions, the feasibility of lattice-based VRFs was demonstrated in [46, 29]. Goyal et al. [29] present a construction that relies on sophisticated tools such as constrained PRFs without providing a practical efficiency evaluation. In a subsequent work, Yang et al. [46] briefly mentioned their VRF construction derived from zero-knowledge proofs, but they did not present any formal security proof or performance evaluation for it. Nevertheless, the authors acknowledge that the proposed construction guarantees only trusted uniqueness, rendering it inadequate for blockchain-based applications. In addition, the approach appears impractical, as even the fundamental proofs in [46] incur communication overheads of several megabytes. Later Esgin et al. proposed a practical lattice-based construction LB-VRF in [22] which have some limitations as it generate only a only one VRF output per secret-verification key pair. In 2023, Esgin et al. [23] presented an improved lattice-based post-quantum VRF construction. Some hash-based constructions are proposed in [13, 21]. The construction by Buser et al. [13] relies on hash-based signature scheme XMSS and have huge key generation time. Esgin et al. in [21] gave an indexed VRF and a authenticated extended version of that. Recently, isogeny-based VRF constructions are proposed in [37–39].

1.4 Organization

The rest of the paper is structured as follows. In Section 2, we introduce the necessary notations, review the fundamental definitions, theorems, present proposed security for the unforgeability of the underlying signature schemes such as $\mathsf{WOTS}^+, \mathsf{XMSS}$ and VRF. In Section 3, we present our XM-VRF construction with detailed algorithms, establish its correctness, storage analysis and prove its security in the random oracle model. In Section 4, we discuss applications of VRFs in blockchain and describe our protocol for random committee selection within the Algorand blockchain framework. Section 5 concludes the paper.

2 Preliminaries

Notation. Let $\lambda \in \mathbb{N}$ denotes the security parameter and $\log a$ stands for $\log_2 a$. For two positive integers a, b, let $r_{a,b} = (r_a, r_{a+1}, \ldots, r_b)$ be a sub-string of $r = r_{1,j} = (r_1, r_2, \ldots, r_j)$ if $b > a$ and $r_{a,b}$ is an empty string if $b < a$. We write $x \xleftarrow{\$} X$ if x is randomly chosen from the set X using the uniform distribution. By $\mathcal{A}^{\mathcal{O}_\mathsf{P}}$ we mean the adversary $\mathcal{A}$ having oracle access to an algorithm P. A function $\mathsf{negl}(\cdot)$ is negligible if for every positive integer c, there exists an integer N such that $\mathsf{negl}(\lambda) < 1/\lambda^c$ for all $\lambda > N$. $\mathsf{H} : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ is a hash function here. $G : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathcal{K}_\lambda$ is a pseudorandom generator which extends a λ-bit string to $\lambda + t$ bit pseudorandom string where elements of $\mathcal{K}_\lambda$ are t bit string. $x||y$ denotes the concatenation of the strings x and y. Here Stack refers to a linear data structure governed by the Last-In, First-Out (LIFO) rule, which means that the most recently inserted element is the first to be retrieved. It supports two fundamental operations: $\mathsf{push}(N)$, which places an element N at the top of the Stack, and $\mathsf{pop}()$ removes the topmost element from the Stack. $N.\mathsf{height}()$ denotes the height of the element N in the Stack.

2.1 One-time Signature $\mathbf{WOTS}^+$

We describe below a type one-time signature scheme $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ of Andreas Hülsing [33] with message space $\mathcal{M} = \{0,1\}^m$.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_{\mathsf{WOTS}^+}$: A trusted authority takes a security parameter λ as input here and performs the following steps:

i. Choose message length m and parameter $w \in \mathbb{N}, w > 1$ and compute $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$, $l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$.
ii. Sample a key space $\mathcal{K}_\lambda$ and pick a keyed function family $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | \ k \in \mathcal{K}_\lambda\}$.
iii. Set public parameter as $\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda)$ and publishes $\mathsf{pp}_{\mathsf{WOTS}^+}$.

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{WOTS}^+}) \to (\mathsf{sk}, \mathsf{pk})$: A user runs this algorithm on input the public parameter $\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda)$ and generates the signing-verification key pair $(\mathsf{sk}, \mathsf{pk})$ in the following way:

i. Compute $l = l_1 + l_2$ and choose $l + w - 1$ random λ-bit strings. The first l many random λ-bit strings are used to construct $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_l)$ with $\mathsf{sk}_i \in \{0,1\}^\lambda$ for $i = 1, 2, \ldots, l$ and the remaining $w-1$ random λ-bit strings are used to construct $r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1})$ with $r_i \in \{0,1\}^\lambda$ for $i = 1, 2, \ldots, w-1$.
ii. Pick $k \xleftarrow{\$} \mathcal{K}_\lambda$, compute $\mathsf{pk}_i = c_k^{w-1}(\mathsf{sk}_i, r)$ for $i = 1, 2, \ldots, l$ and set $\mathsf{pk} = \{\mathsf{aux}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$ where $\mathsf{aux} = (r, k) \in \{0,1\}^{\lambda \times (w-1)} \times \mathcal{K}_\lambda$ and $c_k^i(x, r)$ is a chaining function defined iteratively in **Fig. 1**.
iii. Set the secret key $\mathsf{sk} = \{\mathsf{aux}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$.
iv. Keep sk secret to itself and publishes pk.

$$c_k^i(x, s) = x \text{ if } i = 0 \text{ and}$$
$$c_k^i(x, s) = f_k(c_k^{i-1}(x, s) \oplus s_i) \text{ if } i > 0$$

Fig. 1. Chaining Function $c_k^i(x, s)$ which takes a string $x \in \{0,1\}^\lambda$, an iteration counter i, key $k \in \mathcal{K}_\lambda$ and randomized element $s = s_{1,j} = (s_1, s_2, \ldots, s_j) \in \{0,1\}^{\lambda \times j}$ with $j \geq i$ as input and $f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a keyed function from $\mathcal{F}_\lambda$.

$\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{sk}, M) \to \sigma$: Taking input the public parameter $\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda)$, a signing key $\mathsf{sk} = \{\mathsf{aux} = (r, k), (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$, a message $M \in \{0,1\}^m$, the signer produces a signature σ on M as follows:

i. Compute base w representation of M as $M = M_1 M_2 \ldots . M_{l_1}$ where $M_i \in \{0, 1, \ldots, w-1\}$ for $i = 1, 2, \ldots, l_1$. This can be done as l_1 is set as $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$. Calculate $C = \sum_{i=1}^{l_1} (w - 1 - M_i)$.
ii. Compute base w representation of C as $C_1 C_2 \ldots . C_{l_2}$ where $C_i \in \{0, 1, \ldots, w-1\}$ for $i = 1, 2, \ldots, l_2$.This can be done as l_2 is set as $l_2 = \left\lfloor \frac{\log (l_1 (w-1))}{\log w} \right\rfloor + 1$.
iii. Set $b = (b_1, b_2, \ldots, b_l) = M_1 \ldots M_{l_1} C_1 \ldots C_{l_2}$. As l is set as $l = l_1 + l_2$.
iv. Using pp and sk, generate the signature

$$\sigma = (\sigma_1, \ldots, \sigma_l) = (c_k^{b_1}(\mathsf{sk}_1, r), \ldots, c_k^{b_l}(\mathsf{sk}_l, r))$$

where $r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1})$ with $r_i \in \{0,1\}^\lambda$ for $i = 1, 2, \ldots, w-1$ and $k \in \mathcal{K}_\lambda$ are extracted from aux in sk.
v. Send the message-signature pair (M, σ) over a public channel.

$\mathsf{Vrfy}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{pk}, M, \sigma) \to 1/0$: The verifier checks the validity of the signature $\sigma = (\sigma_1, \ldots, \sigma_l)$ on message $M \in \{0,1\}^m$ using the verification key $\mathsf{pk} = \{\mathsf{aux} = (r, k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$ as follows:

i. Compute $b_1, b_2, \ldots, b_l$ using base w representation $M_1 M_2 \ldots M_{l_1}$ of M and base w representation $C_1 C_2 \ldots . C_{l_2}$ of $C = \sum_{i=1}^{l_1} (w - 1 - M_i)$ exactly in the same manner as in algorithm Sign.

ii. Check $\mathsf{pk}_i \stackrel{?}{=} c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1})$ for $i = 1, 2, \ldots, l$ where $r_{b_i+1,w-1} = (r_{b_i+1}, r_{b_i+2}, \ldots, r_{w-1})$ with $r_{b_i+j} \in \{0,1\}^\lambda$ for $j = 1, 2, \ldots, w-1-b_i$ are extracted from aux in pk.

iii. Output 1 if comparison succeeds, otherwise output 0.

Correctness: Note that, pk_i are computed by invoking chain function c_k iteratively $(w-1)$ times on $(\mathsf{sk}_i, r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1}))$. Also σ_i are generated by running the chaining function c_k iteratively b_i times on $(\mathsf{sk}_i, r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1}))$. During verification, pk_i are recomputed by applying chain function c_k iteratively $(w-1-b_i)$ times on $(\sigma_i, r_{b_i+1,w-1} = (r_{b_i+1}, r_{b_i+2}, \ldots, r_{w-1}))$ where σ_i are generated during signature algorithm by running the chaining function c_k iteratively b_i times on $(\mathsf{sk}_i, r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1}))$.
Suppose $\sigma = (\sigma_1, \ldots, \sigma_l)$ is computed in the correct way as shown in algorithm $\mathsf{WOTS}^+.\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{sk}, M)$ where $\mathsf{pp}_{\mathsf{WOTS}^+} \leftarrow \mathsf{WOTS}^+.\mathsf{Setup}(1^\lambda), (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{WOTS}^+.\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{WOTS}^+})$ and $M \in \mathcal{M}$. We explain the correctness of $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ below by showing the following equation holds for each $i = 1, 2, \ldots, l$.

$$c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1}) = \mathsf{pk}_i$$

For simplicity, let us rename $r_{b_i+1,w-1} = (r_{b_i+1}, r_{b_i+2}, \ldots, r_{w-1})$ as $(s_1, s_2, \ldots, s_{w-1-b_i}) = s_{1,w-1-b_i} = s$ where $s_j = r_{b_i+j}$ for $j = 1, 2, \ldots, w-1-b_i$.
For $w - 1 - b_i > 0$, we have

$$\begin{aligned}
c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1}) &= c_k^{w-1-b_i}(\sigma_i, s) \\
&= f_k(c_k^{w-2-b_i}(\sigma_i, s) \oplus s_{w-1-b_i}) \\
&= \underbrace{f_k(f_k(\ldots f_k}_{w-1-b_i \text{ times}}(c_k^0(\sigma_i, s) \oplus s_1) \cdots \oplus s_{w-1-b_i}) \ldots) \\
&= f_k(f_k(\ldots f_k(\sigma_i \oplus s_1) \cdots \oplus s_{w-1-b_i}) \ldots) \\
&= f_k(f_k(\ldots f_k(\sigma_i \oplus r_{b_i+1}) \cdots \oplus s_{w-1-b_i}) \ldots) \\
&= f_k(f_k(\ldots \underbrace{f_k(c_k^{b_i}(\mathsf{sk}_i, r) \oplus r_{b_i+1})}_{c_k^{b_i+1}(\mathsf{sk}_i, r)} \cdots \oplus s_{w-1-b_i}) \ldots) \\
&= \underbrace{f_k(f_k(\ldots f_k}_{w-2-b_i \text{ times}}(c_k^{b_i+1}(\mathsf{sk}_i, r) \oplus s_2) \cdots \oplus s_{w-1-b_i}) \ldots) \\
&= f_k(f_k(\ldots \underbrace{f_k(c_k^{b_i+1}(\mathsf{sk}_i, r) \oplus r_{b_i+2})}_{c_k^{b_i+2}(\mathsf{sk}_i, r)} \cdots \oplus s_{w-1-b_i}) \ldots) \\
&\quad \vdots \\
&= f_k(c_k^{w-2}(\mathsf{sk}_i, r) \oplus s_{w-1-b_i}) \\
&= f_k(c_k^{w-2}(\mathsf{sk}_i, r) \oplus r_{w-1}) \\
&= c_k^{w-1}(\mathsf{sk}_i, r) = \mathsf{pk}_i
\end{aligned}$$

When $w-1-b_i=0$, we get

$$c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1}) = c_k^0(\sigma_i, r_{b_i+1,w-1}) = \sigma_i = c_k^{b_i}(\mathsf{sk}_i, r) = c_k^{w-1}(\mathsf{sk}_i, r) = \mathsf{pk}_i$$

establishing the correctness of the one time signature scheme WOTS^+.

Theorem 1. *If $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \mathcal{K}_\lambda\}$ is a collision resistant keyed function family, then $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a unique signature scheme.*

Proof. Let us assume that $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ is not a unique signature scheme. So $|\Sigma(m, \mathsf{pk})| \neq 1$ where $\Sigma(m, \mathsf{pk}) = \{\sigma : \mathsf{WOTS}^+.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma) = 1\}$. Therefore assume that there exist two different WOTS^+ signatures σ, σ' of the message m such that

$$\mathsf{WOTS}^+.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma) = 1 = \mathsf{WOTS}^+.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma')$$

Since $(\sigma_1, \sigma_2, \ldots, \sigma_l) = \sigma \neq \sigma' = (\sigma'_1, \sigma'_2, \ldots, \sigma'_l)$, so there must exist some i for which $\sigma_i \neq \sigma'_i$.
Also we have $\mathsf{WOTS}^+.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma) = 1 = \mathsf{WOTS}^+.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma')$ which implies $c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1}) = \mathsf{pk}_i = c_k^{w-1-b_i}(\sigma'_i, r_{b_i+1,w-1})$.
Since $\sigma_i \neq \sigma'_i$, so at some iteration of c_k the output will be equal for σ_i, σ'_i first time, say j. Therefore for $r_{b_i+1,w-1} = s = (s_1, s_2, \ldots, s_{w-1-b_i})$

$$\begin{aligned} & c_k^j(\sigma_i, s) = c_k^j(\sigma'_i, s) \\ \implies & f_k(c_k^{j-1}(\sigma_i, s) \oplus s_j) = f_k(c_k^{j-1}(\sigma'_i, s) \oplus s_j) \end{aligned}$$

Since j is the first iteration of c_k where it outputs equally for σ_i and σ'_i, so

$$\begin{aligned} & c_k^{j-1}(\sigma_i, s) \neq c_k^{j-1}(\sigma'_i, s) \\ \implies & c_k^{j-1}(\sigma_i, s) \oplus s_j \neq c_k^{j-1}(\sigma'_i, s) \oplus s_j \end{aligned}$$

So we obtain collision for the function f_k, which contradicting the fact that $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \mathcal{K}_\lambda\}$ is a collision resistant keyed function family.
So our assumption was wrong. Hence $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a unique signature scheme. □

Theorem 2. [33] *Let $\lambda, w, m \in \mathbb{N}$, $w, n = \mathsf{poly}(\lambda), \mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \mathcal{K}_\lambda\}$ is a second preimage resistant (SPR), undetectable (UD), one-way (OW) keyed function family. Then the advantage of $\mathsf{WOTS}^+ = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ against an* EU-CMA *attack satisfies the following*

$$\mathsf{Adv}^{\text{EU-CMA}}_{\mathcal{A},\mathsf{WOTS}^+}((\lambda, w, m); t, 1) \leq w.\mathsf{Adv}^{\mathsf{UD}}_{\mathcal{B},\mathcal{F}_\lambda}(t^*) + wl.\max\{\mathsf{Adv}^{\mathsf{OW}}_{\mathcal{C},\mathcal{F}_\lambda}(t'), w.\mathsf{Adv}^{\mathsf{SPR}}_{\mathcal{D},\mathcal{F}_\lambda}(t')\}$$

where $\mathcal{B}$ is an adversary against UD, $\mathcal{C}$ is an adversary against OW, $\mathcal{D}$ is an adversary against SPR with $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$, $l_2 = \left\lfloor \frac{\log(l_1(w-1))}{\log w} \right\rfloor + 1$, $l_1 + l_2 = l$, $t' = t + 3lw$ and $t^* = t + 3lw + w - 1$.

As shown in Boneh el al. [7], WOTS^+ satisfies SU-CMA, stronger notion of EU-CMA, where the adversary can forge a message m^* from query list Qlist but with a different valid signature i.e, the queried signature σ on m^* is different from forged signature σ^* on m^* submitted by the adversary where $m^* \in \mathsf{Qlist}$. Further details on these security notions, such as EU-CMA, are provided in Appendix A, while descriptions of different function families, including $\mathsf{OW}, \mathsf{SPR}$ and UD, are given in Appendix B.

2.2 Extended Merkle Signature Scheme XMSS

We recall the extended Merkle signature scheme $\mathsf{XMSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ of Buchmann et al. [11]. It uses one time signature WOTS^+ as the underlying collision resilient signature scheme and invokes a procedure $\mathsf{XMSS\ Tree}$ that assigns WOTS^+ public keys at the leaves.

XMSS Tree: An XMSS tree of height h is a binary tree having $h+1$ levels with the leaves at level 0 and the root at level h. The nodes at level j are denoted by $N_{i,j}$, $0 \leq i < 2^{h-j}, 0 \leq j \leq h$. Taking 2^h many WOTS^+ public keys as input, the $\mathsf{XMSS\ Tree}$ authenticates them by assigning at leaf nodes $N_{i,0}, 1 \leq i \leq 2^h$ and accumulate them at the root node $\mathsf{rt} = N_{0,h}$ as shown in **Algorithm 1** using sub-procedure $\mathsf{L\text{-}tree}$ described in **Algorithm 2**. A pictorial representation of XMSS tree with $h = 3$ is shown in **Fig. 2**.

Algorithm 1: $\mathsf{XMSS\ Tree}$

Public Parameter: $\mathsf{pp} = \mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+}, k, h, G, \mathsf{H})$ where $\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda) \leftarrow \mathsf{WOTS}^+.\mathsf{Setup}(1^\lambda)$ and $k \in \{0,1\}^\lambda, h \in \mathbb{N}$, G:$\{0,1\}^\lambda \rightarrow \{0,1\}^{2\lambda}$ is a PRG, $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$ is a hash function.

Input: $\mathsf{pk}^{(1)}, \mathsf{pk}^{(2)}, \mathsf{pk}^{(3)}, \ldots, \mathsf{pk}^{(2^h)}, B_1, B_2, \ldots, B_{h+\lceil \log l \rceil}$ where $\mathsf{pk}^{(i)} = \{\mathsf{aux}^{(i)}, (\mathsf{pk}_1^{(i)}, \mathsf{pk}_2^{(i)} \ldots, \mathsf{pk}_l^{(i)})\}$ for $i = 1, 2, \ldots, 2^h$ and $B_i \in \{0,1\}^{2\lambda}$ for $i = 1, 2, \ldots, h + \lceil \log l \rceil$

Output: rt

1. **for** $(i = 1; i \leq 2^h; i{+}{+})$ **do**
2. $\quad \mathsf{rt}^{(i)} = \mathsf{L\text{-}tree}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{H}, \mathsf{pk}_1^{(i)}, \mathsf{pk}_2^{(i)} \ldots, \mathsf{pk}_l^{(i)}, B_{h+1}, B_{h+2}, \ldots, B_{h+\lceil \log l \rceil})$;
3. $\quad$ Set $N_{i-1,0} = \mathsf{rt}^{(i)}$;
4. **endfor**
5. **for** $(j = 1; j \leq h; j{+}{+})$ **do**
6. $\quad$ **for** $(i = 0; i < 2^{h-j}; i{+}{+})$ **do**
7. $\quad\quad N_{i,j} = \mathsf{H}((N_{2i,j-1} || N_{2i+1,j-1}) \oplus B_j)$;
8. $\quad$ **endfor**
9. **endfor**
10. **Return** $\mathsf{rt} = N_{0,h}$;

L-tree: An L-tree is a binary hash tree which takes l many λ -bit strings $(\mathsf{pk}_1, \mathsf{pk}_2, ..., \mathsf{pk}_l)$ from public key $\{\mathsf{aux}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$ of WOTS^+ as input and

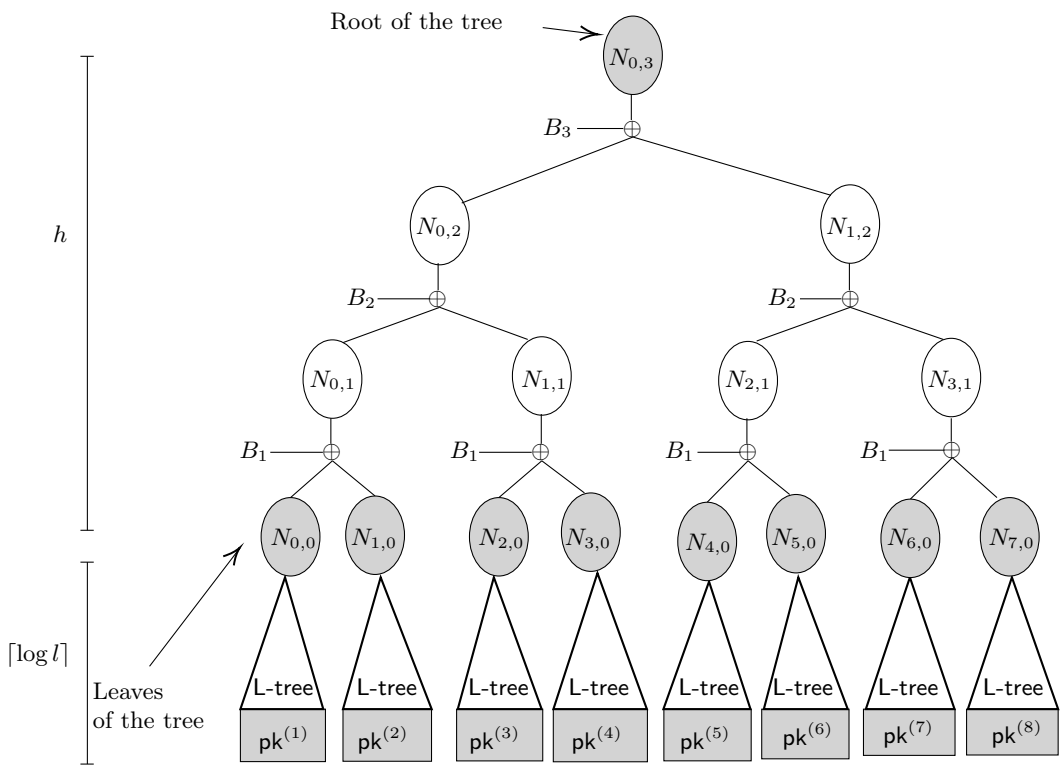

Fig. 2. Example of XMSS tree with $h = 3$ where $N_{i,j} = \mathsf{H}((N_{2i,j-1}||N_{2i+1,j-1}) \oplus B_j)$ and $\mathsf{rt}^{(i)} = \mathsf{L\text{-}tree}$ $(\mathsf{pk}^{(i)}, B_{h+1}, B_{h+2}, \ldots, B_{h+\lceil \log l \rceil})$

condense it to a single λ-bit string as in **Algorithm 2**. **Fig. 3** provides a comprehensive visual representation of L-tree having $l = 5$ leaf nodes.

The Extended Merkle Signature Scheme XMSS comprises of four probabilistic polynomial time (PPT) algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ having the following interface:
$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}_{\mathsf{XMSS}}$: A trusted authority takes as input a security parameter λ and produces public parameter $\mathsf{pp}_{\mathsf{XMSS}}$ in subsequent manner:

i. Run $\mathsf{WOTS}^+.\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda)$ where m is the length of the message, w is the parameter, $\mathcal{K}_\lambda$ is key space, $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda | \ k \in \mathcal{K}_\lambda\}$ is a keyed pseudorandom function family, $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$ and $l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$.
ii. Pick $h \in \mathbb{N}$ and a pseudorandom generator $G : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda \times \mathcal{K}_\lambda$ and a second preimage resistant hash function $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$.
iii. Select $k \in \mathcal{K}_\lambda$ for the key in function family.

Algorithm 2: L-tree

Input: $\mathsf{pp} = \mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda), \mathsf{H}, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_l)$ and $(B_1, B_2, \ldots, B_{\lceil \log l \rceil})$ where $\mathsf{pk}_i \in \{0,1\}^\lambda$ for $i = 1, \ldots, l$, $B_i \in \{0,1\}^{2\lambda}$ for $i = 1, \ldots, \lceil \log l \rceil$, $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$ is a hash function and $\mathsf{pp}_{\mathsf{WOTS}^+} \leftarrow \mathsf{WOTS}^+.\mathsf{Setup}(1^\lambda)$

Output: rt

1. **for** $(i = 0; i < l; i{+}{+})$ **do**
2. $\quad L_{i,0} = \mathsf{pk}_{i+1}$;
3. **endfor**
4. **for** $(j = 1; j \leq \lceil \log l \rceil; j{+}{+})$ **do**
5. $\quad$ **for** $(i = 0; i < \lfloor \frac{l}{2^j} \rfloor; i{+}{+})$ **do**
6. $\quad\quad L_{i,j} = \mathsf{H}((L_{2i,j-1} || L_{2i+1,j-1}) \oplus B_j)$;
7. $\quad$ **endfor**
8. **endfor**
9. **if** (l is not a power of 2) **then**
10. A node with no right sibling is pushed to a higher level of the tree until it
11. becomes the right sibling of another node i.e,
12. **for** $(j = 1; j \leq \lceil \log l \rceil - 1; j{+}{+})$ **do**
13. $\quad$ **if** $(\lceil \frac{l}{2^{j-1}} \rceil \equiv 1 \pmod 2)$ **then**
14. $\quad\quad L_{\lfloor \frac{l}{2^j} \rfloor, j} = L_{\lfloor \frac{l}{2^{j-1}} \rfloor, j-1}$;
15. $\quad$ **endif**
16. **endfor**
17. **endif**
18. Return $\mathsf{rt} = L_{0, \lceil \log l \rceil}$;

iv. Publish the public parameter $\mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+}, k, h, G, \mathsf{H})$.

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{XMSS}}) \rightarrow (\mathsf{sk}, \mathsf{pk})$: A user takes public parameter $\mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+}, k, h, G, \mathsf{H})$ as input and generates XMSS signing-verification key pair $(\mathsf{sk}, \mathsf{pk})$ for XMSS by executing the following steps:

i. Parse $\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \mathcal{K}_\lambda, \mathcal{F}_\lambda)$
ii. Choose $s_0 \in \{0,1\}^\lambda$ and set $l = l_1 + l_2$.
iii. **for** $(i = 1; i \leq 2^h; i{+}{+})$ **do**
Run $G(s_{i-1}) = s_i || t_i$ where $s_i \in \{0,1\}^\lambda$ and $t_i \in \mathcal{K}_\lambda$
Set λ-bit string $\mathsf{rand}_j = f_{t_i}(\mathsf{bin}(j-1))$ for $j = 1, \ldots, l+w-1$ where $\mathsf{bin}(j-1)$ is binary representation of $j-1$ and $f_{t_i} \in \mathcal{F}_\lambda$
Use $\lambda(l+w-1)$-bit string $\mathsf{rand} = (\mathsf{rand}_1, \ldots, \mathsf{rand}_{l+w-1})$ to construct random element $r^{(i)} = (\mathsf{rand}_{l+1}, \ldots, \mathsf{rand}_{l+w-1})$ and $\mathsf{sk}^{(i)} = \{(r^{(i)}, k), (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$ where $\mathsf{sk}_j = \mathsf{rand}_j$ for $j = 1, \ldots, l$
Set $\mathsf{pk}_j = c_k^{w-1}(\mathsf{sk}_i, r^{(i)})$ for $j = 1, \ldots, l$ where $c_k^i(x, r)$ is a chain ing function defined iteratively in **Fig. 1**.
Assign $\mathsf{pk}^{(i)} = \{(r^{(i)}, k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$
endfor
iv. Pick random bitmasks $B_j \in \{0,1\}^{2\lambda}$ for $j = 1, \ldots, h + \lceil \log l \rceil$
v. Run XMSS $\mathsf{Tree}(\mathsf{pp}_{\mathsf{XMSS}}, \mathsf{pk}^{(1)}, \ldots, \mathsf{pk}^{(2^h)}, B_1, B_2, \ldots, B_{h+\lceil \log l \rceil}) \rightarrow N_{0,h}$

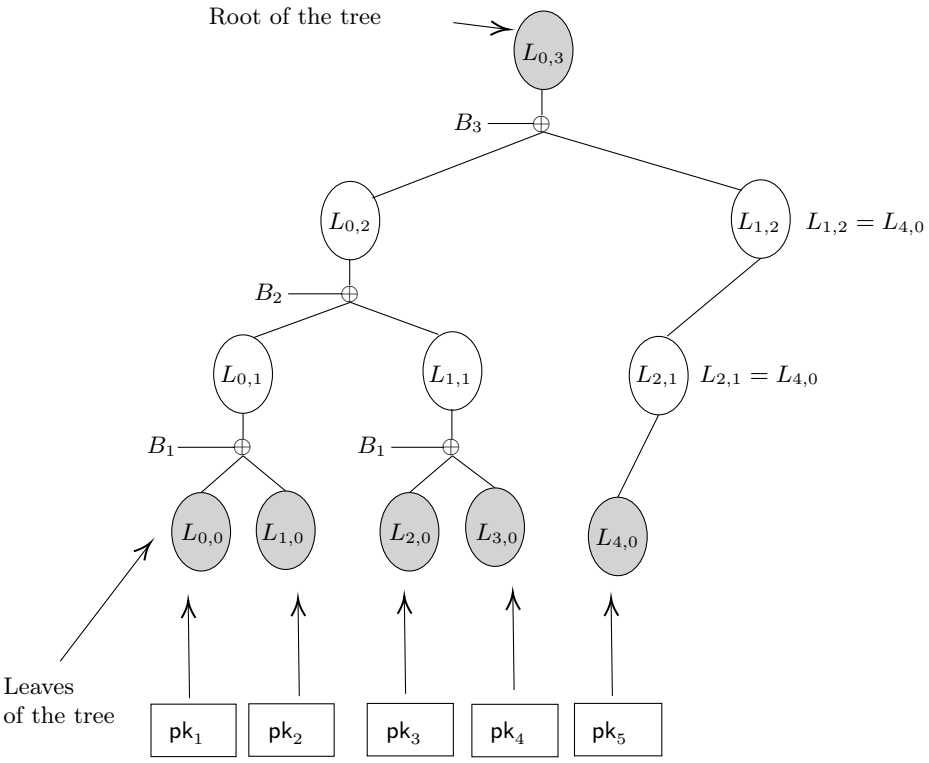

Fig. 3. Example of L-tree with $l = 5$ leaf nodes where $L_{i,j} = \mathsf{H}((L_{2i,j-1}||L_{2i+1,j-1}) \oplus B_j)$

vi. Set the secret key $\mathsf{sk} = s_0$ and the public key $\mathsf{pk} = (N_{0,h}, B_1, B_2, \ldots, B_{h+\lceil \log l \rceil}, \{r^{(i)}\}_{i=1}^{2^h})$
vii. Keep sk secret to itself and publish pk.

$\mathsf{Sign}(\mathsf{pp}_{\mathsf{XMSS}}, \mathsf{sk}, i, M) \rightarrow \mathsf{Sig}$: A signer on input the public parameter $\mathsf{pp}_{\mathsf{XMSS}}$, signing key sk, index i and the message $M \in \{0,1\}^m$ to generate the signature Sig on M as follows:

i. Parse $\mathsf{sk} = s$
for $(j = 1; j \leq i; j{+}{+})$ **do**
$G(\mathsf{s}) = s||t$ where $s \in \{0,1\}^\lambda$ and $t \in \mathcal{K}_\lambda$
endfor
ii. Set λ-bit string $\mathsf{rand}_j = f_t(\mathsf{bin}(j-1))$ for $j = 1, \ldots, l+w-1$ where $\mathsf{bin}(j-1)$ is binary representation of $j-1$ and $f_t \in \mathcal{F}_\lambda$
Use $\mathsf{rand} = (\mathsf{rand}_1, \ldots, \mathsf{rand}_{l+w-1}) \in \{0,1\}^{\lambda \times (l+w-1)}$ to construct random element $r^{(i)} = (\mathsf{rand}_{l+1}, \ldots, \mathsf{rand}_{l+w-1}) \in \{0,1\}^{\lambda \times (w-1)}$ and set $\mathsf{sk}_j = \mathsf{rand}_j$ for $j = 1, \ldots, l$
Set $\mathsf{pk}_j = c_k^{w-1}(\mathsf{sk}_i, r^{(i)})$ for $j = 1, \ldots, l$ where $c_k^i(x, r)$ is a chaining

function defined iteratively in **Fig. 1**
Assign $\mathsf{sk}^{(i)} = \{(r^{(i)}, k), (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$ and $\mathsf{pk}^{(i)} = \{(r^{(i)}, k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$

iii. Generate signature $\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{sk}^{(i)}, M)$ where $\sigma = (\sigma_1, \ldots, \sigma_l) = (c_k^{b_1}(\mathsf{sk}_1, r^{(i)}), \ldots, c_k^{b_l}(\mathsf{sk}_l, r^{(i)}))$

iv. Calculate the authentication path $\mathsf{Auth} = (\mathsf{Auth}_0, ..., \mathsf{Auth}_{h-1})$ (contains the sibling nodes corresponding to each node along the path from the leaf node $N_{0,i}$ to the root) of the i-th leaf of the XMSS tree using AuthCalc algorithm written in **Algorithm 4**.

v. Send the signature $\mathsf{Sig} = (i, \sigma, \mathsf{Auth})$

$\mathsf{Vrfy}(\mathsf{pp}_{\mathsf{XMSS}}, \mathsf{pk}, M, \mathsf{Sig}) \rightarrow 1/0$: The verifier checks the validity of the signature Sig on message M using the verification key pk in the following method:

i. Parse $\mathsf{Sig} = (i, \sigma, \mathsf{Auth})$.

ii. Compute $b_1, b_2, \ldots, b_l$ using base w representation $M_1 \ldots M_{l_1}$ of M and calculating base w representation $C_1 C_2 \ldots . C_{l_2}$ of $C = \sum_{j=1}^{l_1}(w - 1 - M_j)$ exactly in the same manner as in algorithm $\mathsf{WOTS}^+.\mathsf{Sign}$.

iii. Extract $r^{(i)}, k$ from pk, σ from Sig
Set $\mathsf{pk}_j^{(i)} = c_k^{w-1-b_j}(\sigma_j, r_{b_j+1,w-1}^{(i)})$ for $j = 1, \ldots, l$

iv. Run $\mathsf{L\text{-}tree}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{pk}_1^{(i)}, \mathsf{pk}_2^{(i)} \ldots, \mathsf{pk}_l^{(i)}, B_{h+1}, B_{h+2}, \ldots, B_{h+\lceil \log l \rceil}) \rightarrow \mathsf{rt}^{(i)}$. Set the corresponding XMSS tree leaf node as $\mathsf{rt}^{(i)}$. Then use this leaf $\mathsf{rt}^{(i)}$ and the authentication path $\mathsf{Auth} = (\mathsf{Auth}_0, ..., \mathsf{Auth}_{h-1})$ to compute the root node rt of the XMSS tree.

v. Extract $N_{0,h}$ from pk.
Check $\mathsf{rt} \stackrel{?}{=} N_{0,h}$
Output 1, if verification passes, otherwise output 0

Theorem 3. [11] *If* $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^{\lambda}$ *is a second preimage resistant hash function and* $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda | \ k \in \mathcal{K}_\lambda\}$ *is a pseudorandom function family, then* $\mathsf{XMSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is EU-CMA secure.*

2.3 Verifiable Random Function(VRF)

A verifiable random function (VRF) is a function that generates a pseudo-random value that preserves the verifiability property where anyone can verify the evaluated pseudo-random value using a proof provided by the VRF.

Definition 1 (Verifiable Random Function). [42] *A* $\mathsf{VRF} = (\mathsf{ParamGen}, \mathsf{Key\ Gen}, \mathsf{Eval}, \mathsf{Verify})$ *with input length* l *and output length* m *consists of the following polynomial time algorithms:*
$\mathsf{ParamGen}(1^\lambda) \rightarrow \mathsf{pp}_{\mathsf{VRF}}$*: On input a security parameter* λ*, this probabilistic algorithm run by a trusted authority, outputs global public parameter* $\mathsf{pp}_{\mathsf{VRF}}$.

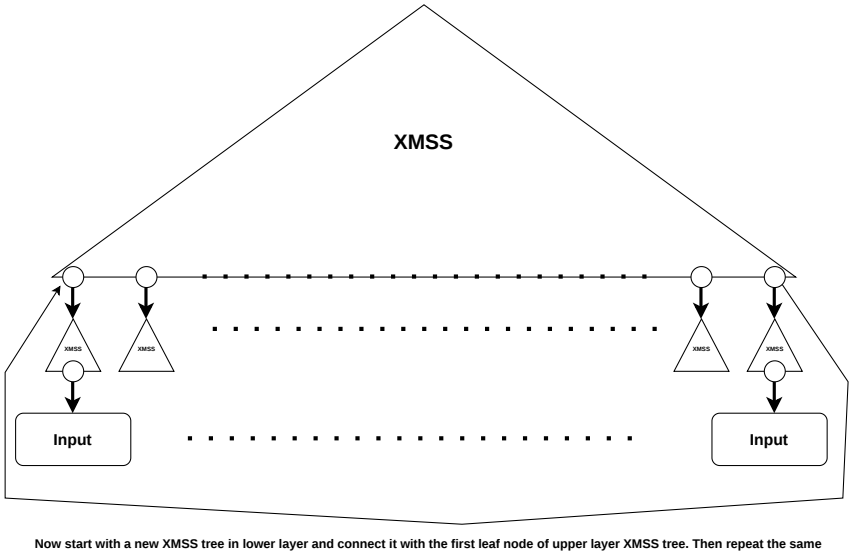

Fig. 4. XMSS tree with 2 layers

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{VRF}}) \rightarrow (\mathsf{sk}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}})$*: On input public parameter* $\mathsf{pp}_{\mathsf{VRF}}$ *this probabilistic algorithm run by the evaluator, generates a secret key* $\mathsf{sk}_{\mathsf{VRF}}$ *and a verification key* $\mathsf{vk}_{\mathsf{VRF}}$.
$\mathsf{Eval}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}, x) \rightarrow (y_{\mathsf{VRF}}, \pi_{\mathsf{VRF}})$*: The evaluator with it's secret key* $\mathsf{sk}_{\mathsf{VRF}}$ *and an input* x *of length* l *this algorithm outputs the evaluation value* y_{VRF} *and the corresponding proof* π_{VRF} *of correct computation of* y_{VRF}.
$\mathsf{Verify}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}, x, y_{\mathsf{VRF}}, \pi_{\mathsf{VRF}}) \rightarrow 1/0$*: On input* $(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}, y_{\mathsf{VRF}}, x, \pi_{\mathsf{VRF}})$ *a verifier executes this probabilistic algorithm and outputs either* 1 *or* 0.

A VRF is required to have the following security properties:
Provability: If $(y_{\mathsf{VRF}}, \pi_{\mathsf{VRF}})$ is the output of $\mathsf{VRF.Eval}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}, x)$ where $\mathsf{sk}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}$ are honestly generated by the algorithm $\mathsf{VRF.Eval}$, then $\mathsf{VRF} = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ is said to satisfy provability if

$$\mathsf{VRF.Verify}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}, x, y_{\mathsf{VRF}}, \pi_{\mathsf{VRF}}) = 1$$

Uniqueness: $\mathsf{VRF} = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ is said to satisfy uniqueness if no values $(\mathsf{vk}_{\mathsf{VRF}}, y^{(1)}_{\mathsf{VRF}}, y^{(2)}_{\mathsf{VRF}}, x, \pi^{(1)}_{\mathsf{VRF}}, \pi^{(2)}_{\mathsf{VRF}})$ can satisfy

$$\mathsf{VRF.Verify}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}, x, y^{(1)}_{\mathsf{VRF}}, \pi^{(1)}_{\mathsf{VRF}}) = \mathsf{VRF.Verify}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}, x, y^{(2)}_{\mathsf{VRF}}, \pi^{(2)}_{\mathsf{VRF}}) = 1$$

when $y^{(1)}_{\mathsf{VRF}} \neq y^{(2)}_{\mathsf{VRF}}$
Pseudorandomness: Let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary participating in the experiment $\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{VRF}}(\lambda)$ in **Experiment 4**.

$\mathsf{VRF} = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ is said to satisfy pseudorandomness if

$$\mathsf{Adv}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{VRF}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{VRF}}(\lambda) = 1 \right] \right\} \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Experiment 4: $\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{VRF}}(\lambda)$

Setup: The challenger generates public parameter $\mathsf{pp}_{\mathsf{VRF}} \leftarrow \mathsf{VRF.ParamGen}(1^\lambda)$ by taking λ as input. Then challenger produces key pairs $(\mathsf{sk}_{\mathsf{VRF}}, \mathsf{vk}_{\mathsf{VRF}}) \leftarrow \mathsf{VRF.KeyGen}(\mathsf{pp}_{\mathsf{VRF}})$. It maintains a list Qlist and initializes $\mathsf{Qlist} = \phi$. It gives the public parameter $\mathsf{pp}_{\mathsf{VRF}}$ and the verification key $\mathsf{vk}_{\mathsf{VRF}}$ to adversary $\mathcal{A}$ and keep $\mathsf{sk}_{\mathsf{VRF}}$ secret to itself.
Query Phase: The adversary $\mathcal{A}$ repeats polynomial times evaluation query on inputs of its choice to obtain VRF outputs from the challenger. For a evaluation query on an input $x \in \{0,1\}^l$ from $\mathcal{A}$, the challenger generates $\mathsf{VRF.Eval}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}, x) \rightarrow (y, \pi)$ and send it to $\mathcal{A}$ and update $\mathsf{Qlist} = \mathsf{Qlist} \cup \{x\}$.
Challenge: After all these queries $\mathcal{A}$ submit an input x^* and a secret state st to the challenger using $\mathcal{A}^{\mathcal{O}_{\mathsf{VRF.Eval}(\cdot)}}(\mathsf{vk}_{\mathsf{VRF}}) \rightarrow (x^*, st)$. The challenger in turn generates $(y_0, \pi_0) \leftarrow \mathsf{VRF.Eval}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}, x^*)$ and challenge $y_1 \leftarrow \{0,1\}^m$. Then the challenger send y_b to the adversary $\mathcal{A}$ where $b \xleftarrow{\$} \{0,1\}$.
Response: In response adversary $\mathcal{A}$ sends b' to the challenger. If $b = b'$ and $x^* \notin \mathsf{Qlist}$), then return 1, otherwise return 0.

where $\mathcal{O}_{\mathsf{VRF.Eval}(\cdot)}$ is an oracle that on an input value x outputs the VRF value y_{VRF} and the corresponding proof π_{VRF} of the correct computation of y_{VRF}.

Algorithm 3: TreeHash

Input: stack Stack, node N, hash function $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$, random bitmasks $\mathrm{B}_i \in \{0,1\}^{2\lambda}$ for $i = 1, \ldots, h + \lceil \log l \rceil$
Output: updated stack Stack
1. **while** (top node on Stack has same height as N) **do**
2. $\quad t \leftarrow N.\mathsf{height}() + 1$;
3. $\quad N \leftarrow \mathsf{H}((\mathsf{Stack.pop}()||N) \oplus B_t)$;
4. **endwhile**
5. $\mathsf{Stack.push}(N)$;
6. **return** Stack;

3 Construction of XM-VRF from XMSS

This section introduces our construction of a hash based post-quantum secure VRF named $\mathsf{XM\text{-}VRF} = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ uses XMSS (described in section 2.2) of height $h' = h/d$ in 2 layers as shown in **Fig. 4**. Informally, our XM-VRF employs a hierarchical XMSS structure, the bottom XMSS tree signs the actual input as message, and top XMSS tree signs the root of the bottom tree. The verification key of XM-VRF contains the root node of the top layer

Algorithm 4: AuthCalc

Input: $\mathsf{index} \in \{1, 2, \ldots, 2^{h'}\}, h, \mathsf{ac} \geq 2, \mathsf{St}_{\mathsf{cur},i}, \mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^{\lambda}$
Output: The updated state $\mathsf{St}_{\mathsf{cur},i}$

1. Extract $\{\mathsf{Auth}_i\}_{i=0}^{h-1}$ from $\mathsf{St}_{\mathsf{cur},i}$
2. $\mathsf{lp} \leftarrow \max_{g \geq 0}\{g | 2^g | (\mathsf{index}+1)\}$;
3. **if** ($\mathsf{lp} < h-1$ and $\lfloor \mathsf{index}/2^{\mathsf{lp}+1} \rfloor$ is even) **then**
4. $\quad \mathsf{Store}_{\mathsf{lp}} \leftarrow \mathsf{Auth}_{\mathsf{lp}}$;
5. **endif**
6. **if** ($\mathsf{lp} = 0$) **then**
7. $\quad \mathsf{Auth}_0 \leftarrow \mathsf{LeafCalc}(\mathsf{index})$;
8. **endif**
9. **if** ($\mathsf{lp} > 0$) **then**
10. $\quad \mathsf{Auth}_{\mathsf{lp}} \leftarrow \mathsf{H}(\mathsf{Auth}_{\mathsf{lp}-1} || \mathsf{Store}_{\mathsf{lp}-1})$;
11. $\quad$ Remove $\mathsf{Store}_{\mathsf{lp}-1}$;
12. $\quad$ **for** ($i = 0; i \leq \mathsf{lp}-1; i++$) **do**
13. $\quad\quad$ **if** ($i \leq h - \mathsf{ac} - 1$) **then**
14. $\quad\quad\quad \mathsf{Auth}_i \leftarrow \mathsf{Stock}_i$;
15. $\quad\quad$ **else**
16. $\quad\quad\quad \mathsf{Auth}_i \leftarrow \mathsf{Cache}_i.pop()$;
17. $\quad\quad$ **endif**
18. $\quad$ **endfor**
19. $\quad$ **for** ($i = 0; i \leq \min\{\mathsf{lp}-1, h-\mathsf{ac}-1\}; i++$) **do**
20. $\quad\quad$ **if** ($\mathsf{index}+1+3.2^i < 2^h$) **then**
21. $\quad\quad\quad \mathsf{TreeHash}_i.init(\mathsf{index}+1+3.2^i)$;
22. $\quad\quad$ **endif**
23. $\quad$ **endfor**
24. **endif**
25. **for** ($i = 1; i \leq \frac{h-\mathsf{ac}}{2}; i++$) **do**
26. $\quad s \leftarrow \min\{j | \mathsf{Treehash}_j.height = \min_{k=0,\cdots,h-\mathsf{ac}-1} = \{\mathsf{Treehash}_k.height\}\}$;
27. $\quad \mathsf{Treehash}_s.update()$;
28. **endfor**

XMSS tree. Sub-procedures namely TreeHash, AuthCalc, WOTS$^+$PKfromSign and Auth2Root presented in **Algorithm 3, Algorithm 4, Algorithm 6** and **Algorithm 5** respectively are employed by our XM-VRF. Our XM-VRF construction allows multiple iterations of VRF evaluations within a single round (i.e, one step of consensus), matching the Algorand [25] setting. The algorithms for the VRF given in sub section 3.1 below. Prior to describing the main algorithms, we define four auxiliary sub-procedures that will be employed throughout the construction.

- The sub-procedure TreeHash is described in **Algorithm 3**. It primarily constructs XMSS trees using a stack-based approach along with node information. When we call this sub-procedure, it takes a node and a Stack (containing essential tree nodes information in some states) as input and update the informations stored in Stack. Stack refers to a linear data structure governed by the Last-In, First-Out (LIFO) rule — meaning the most recently inserted

Algorithm 5: Auth2Root

Input: Public parameter $\mathsf{pp}_{\mathsf{XM\text{-}VRF}} = (\mathsf{ctr}, m, d, h, \mathsf{ac}, w, \mathcal{F}_\lambda, \mathsf{H}, \mathsf{H}_1, G)$, authentication path Auth, leaf node leaf, bitmasks $B_1, \ldots, B_{h'}$
Output: Root node rt
1. Compute $h' = \frac{h}{d}$;
2. Parse $\mathsf{Auth} = (P_1, \ldots, P_{h'})$ where $P_i \in \{0,1\}^\lambda$ for $i = 1, 2, \ldots, h'$;
   Set $L = \mathsf{leaf} \in \{0,1\}^\lambda$;
   for $(i = 1; i \leq h'; i{+}{+})$ **do**
      Compute $L = \mathsf{H}((P_i || L) \oplus B_i) \in \{0,1\}^\lambda$;
   endfor
3. **return** L ;

element is the first to be retrieved. It supports two fundamental operations: $\mathsf{push}(N)$, which places an element N at the top of the Stack, and pop() removes the topmost element from the Stack. $N.\mathsf{height}()$ denotes the height of the element N in the Stack. Finally Stack contains only the root node of the XMSS tree after calling TreeHash sub-procedure iteratively. XM-VRF employs this sub-procedure in algorithms VRF.KeyGen and VRF.Eval.

- The sub-procedure AuthCalc is described in **Algorithm 4**. This sub-procedur e efficiently computes authentication paths of the XMSS tree in an optimal manner. AuthCalc takes leaf index index, height h, parameter ac, hash function H and state St as input and generates authentication path for next leaf index and update that information in the state St. The core idea is that the left child of a node can be derived from values that already appear in the authentication path. It leverages the stored node information from the states maintained and updated by the TreeHash sub-procedure. These state contains Store, Stock, Cache, Auth. We describe these data structures in the following:
  - Store maintains selected nodes that are used to efficiently compute left authentication nodes.
  - Stock stores a single tail node, which is the first node that sub-procedure TreeHash pushes onto the stack Stack. The remaining tail nodes are subsequently pushed onto the stack Stack.
  - Cache maintains a stack of right authentication nodes that assist in the computation of the desired authentication path.
  - Auth is a collection of nodes that store the current authentication path.
  - Here $\mathsf{TreeHash}_i.init()$ initializes the instance with the leaf index index from which to begin.
  - $\mathsf{Treehash}_j.height$ stores the height of the lowest tail node maintained by the sub-procedure TreeHash instance, either on the stack Stack or in the entry Stock. If sub-procedure TreeHash does not maintain any tail nodes, then $\mathsf{Treehash}_j.height$ is set to j. If sub-procedure TreeHash is either finished or uninitialized, then $\mathsf{Treehash}_j.height$ is set to ∞.
  - $\mathsf{Treehash}_s.update()$ executes sub-procedure TreeHash once, which means it first computes the next leaf, and then performs the necessary hash

Algorithm 6: WOTS$^+$PKfromSign

Input: Public parameter $\mathsf{pp}_{\mathsf{XM\text{-}VRF}} = (\mathsf{ctr}, m, d, h, \mathsf{ac}, w, \mathcal{F}_\lambda, \mathsf{H}, \mathsf{H}_1, G)$, message M, signature σ, X,k
Output: WOTS$^+$ public key pk
1. Compute $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$, $l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$ and $l = l_1 + l_2$;
2. Compute $h' = \frac{h}{d}$ and $\mathsf{ctr}' = (\mathsf{ctr} - 1) \mod 2^{h'}$;
3. **if** $(\mathsf{ctr}' = 0)$ **then**
4. $\quad \mathsf{ctr}' = 2^{h'}$;
5. **endif**
6. Set $S_t = X \in \{0,1\}^\lambda$;
7. **for** $(p = 1; p \leq \mathsf{ctr}'; p{+}{+})$ **do**
8. $\quad$ Compute $G(S_t) = (f_{S_t}(0^\lambda) || f_{S_t}(1^\lambda)) = (S_t || R_t) \in \{0,1\}^\lambda$;
9. **endfor**
10. **for** $(j = 1; j \leq w-1; j{+}{+})$ **do**
11. $\quad$ Compute $G(R_t) = (f_{R_t}(0^\lambda) || f_{R_t}(1^\lambda)) = (R_t || r_j) \in \{0,1\}^\lambda$;
12. **endfor**
13. Set $r = r_{1,w-1} = (r_1, r_2, \ldots, r_{w-1})$;
14. Compute $b_1, b_2, \ldots, b_l$ using base w representation $M_1 \ldots M_{l_1}$ of M and
15. calculating base w
16. representation $C_1 C_2 \ldots . C_{l_2}$ of $C = \sum_{i=1}^{l_1} (w - 1 - M_i)$ exactly in the same
17. manner as in algorithm Sign.;
18. **for** $(i = 1; j \leq l; i{+}{+})$ **do**
19. $\quad$ Set $\mathsf{pk}_i = c_k^{w-1-b_i}(\sigma_i, r_{b_i+1,w-1})$ where $r_{b_i+1,w-1} = (r_{b_i+1}, r_{b_i+2},$
21. $\quad \ldots, r_{w-1})$and $c_k(x,s)$ is presented in Fig.1;
20. **endfor**
21. **return** $\mathsf{pk} = \{(r,k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$;

function evaluations to compute the leaf's parent nodes, provided that tail nodes are stored on the stack Stack.

XM-VRF employs this sub-procedure in algorithm VRF.Eval.

- The sub-procedure WOTS$^+$PKfromSign is described in **Algorithm 6**. The purpose of this algorithm is to reconstruct the WOTS$^+$ public key pk from the corresponding WOTS$^+$ signature σ. XM-VRF employs this sub-procedure in algorithm VRF.Verify.
- Finally, the sub-procedure Auth2Root is described in **Algorithm 5**. This sub-procedure computes the root node rt of an XMSS tree using a given leaf node L and its corresponding authentication path Auth. XM-VRF employs this sub-procedure in algorithm VRF.Verify.

3.1 Algorithm

$\mathsf{ParamGen}(1^\lambda) \rightarrow \mathsf{pp}_{\mathsf{XM\text{-}VRF}}$: On input security parameter λ, a trusted authority produces the public parameter $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}$ in the following way:

i. Choose a collision resistant pseudorandom function family $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \{0,1\}^\lambda\}$, a second preimage resistant hash function $\mathsf{H} : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ and a hash function $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$.
ii. Pick $m, h \in \mathbb{N}$ to set input length and the total height of the tree respectively.
iii. Pick $d \in \mathbb{N}$ so that d divides h to compute height of XMSS trees in each layer.
iv. Compute XMSS tree height $h' = \frac{h}{d} \in \mathbb{N}$, choose AuthCalc [12] parameter $\mathsf{ac} \in \mathbb{N}$ such that $\mathsf{ac} < h', h' - \mathsf{ac}$ is even and $3h' - \mathsf{ac} \geq 4$ as well as $h' - \mathsf{ac} \geq 2d - 2$. Pick the parameter $w \in \mathbb{N}, w > 1$.
v. Choose a stateful forward secure PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$
vi. Initialize $\mathsf{ctr} = 1$.
vii. Set $\mathsf{pp}_{\mathsf{XM\text{-}VRF}} = (\mathsf{ctr}, m, d, h, \mathsf{ac}, w, \mathcal{F}_\lambda, \mathsf{H}, \mathsf{H}_1, G)$.

$\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}) \to (\mathsf{sk}_{\mathsf{XM\text{-}VRF}}, \mathsf{vk}_{\mathsf{XM\text{-}VRF}})$: A user takes input public parameter $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}$ and outputs key pair $(\mathsf{sk}_{\mathsf{XM\text{-}VRF}}, \mathsf{vk}_{\mathsf{XM\text{-}VRF}})$ in the following manner:

i. Choose $X_1 \in \{0,1\}^\lambda,\ X_2 \in \{0,1\}^\lambda$ and
Pick $S_{\mathsf{cur},i} \xleftarrow{\$} \{0,1\}^\lambda$ for $i = 1, 2$
Pick $S_{\mathsf{next}} \xleftarrow{\$} \{0,1\}^\lambda$
ii. Compute $l_1 = \left\lceil \frac{m}{\log w} \right\rceil,\ \ l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$ and $l = l_1 + l_2$.
iii. Compute $h' = \frac{h}{d}$, $\mathsf{ct} = h' + \lceil \log l \rceil$ and pick bitmasks $B_i \in \{0,1\}^{2\lambda}$ randomly for $i = 1, \ldots, \mathsf{ct}$
iv. **for** $(i = 1; i \leq 2; i{+}{+})$ **do**
Set $S_t = X_i$
for $(p = 1; p \leq 2^{h'}; p{+}{+})$ **do**
Compute $G(S_t) = (f_{S_t}(0^\lambda) || f_{S_t}(1^\lambda)) = (S_t || R_t) \in \{0,1\}^{2\lambda}$
for $(j = 1; j \leq w - 1; j{+}{+})$ **do**
Compute $G(R_t) = (f_{R_t}(0^\lambda) || f_{R_t}(1^\lambda)) = (R_t || r_j) \in \{0,1\}^{2\lambda}$
endfor
Set $r[i][p] = (r_1, r_2, \ldots, r_{w-1})$
endfor
endfor
v. Compute $2^{h'}$ WOTS^+ secret and public keys of the first tree of each layer and push public keys into Stack_i to compute the trees in the following manner:
Select a key $k \in \{0,1\}^\lambda$ randomly for the key in pseudorandom function family $\mathcal{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda | k \in \{0,1\}^\lambda\}$.
Initialize empty stacks $\mathsf{Stack}_1, \mathsf{Stack}_2, \mathsf{Stack}^{\mathsf{next}}$
for $(i = 1; i \leq 2; i{+}{+})$ **do**
Set $S_t = S_{\mathsf{cur},i}$
for $(p = 1; p \leq 2^{h'}; p{+}{+})$ **do**
Set $\mathsf{aux}[i][p] = (r[i][p], k)$
Compute $G(S_t) = (f_{S_t}(0^\lambda) || f_{S_t}(1^\lambda)) = (S_t || R_t) \in \{0,1\}^{2\lambda}$
for $(j = 1; j \leq l; j{+}{+})$ **do**
$\mathsf{sk}_j[i][p] = f_{R_t}(\mathsf{bin}(j-1))$ where $\mathsf{bin}(j-1)$ is binary representation of $j - 1$

$\mathsf{pk}_j[i][p] = c_k^{w-1}(\mathsf{sk}_j[i][p], r[i][p])$ where c_k^{w-1} is the chaining function provided in **Fig. 1** of section 2.1

endfor

$\mathsf{sk}[i][p] = \{\mathsf{aux}[i][p], (\mathsf{sk}_1[i][p], \ldots, \mathsf{sk}_l[i][p])\}$

$\mathsf{pk}[i][p] = \{\mathsf{aux}[i][p], (\mathsf{pk}_1[i][p], \ldots, \mathsf{pk}_l[i][p])\}$

$\mathsf{TreeHash}(\mathsf{Stack}_i, \mathsf{pk}[i][p])$

endfor

endfor

vi. Storing state informations $\mathsf{St}_{\mathsf{cur},i}$ from stacks Stack_i for AuthCalc in the following way:
Denote XMSS tree nodes as $N_h[a][1](j), N_h[2](j), h = 0, \ldots, h' - 1$ and $j = 0, \ldots, 2^{h-h'} - 1, a = \mathsf{cur}$ (current tree) , next (next tree)
for $(i = 1; i \leq 2; i{+}{+})$ **do**
Initialize state $\mathsf{St}_{\mathsf{cur},i}$
Set $\mathsf{St}_{\mathsf{cur},i} = \{\mathsf{Store}_j[\mathsf{cur}][i]|_{j=0}^{h'-2}, \mathsf{Auth}_j[\mathsf{cur}][i]|_{j=0}^{h'-1}, \mathsf{Cache}_j[\mathsf{cur}][i]|_{j=h'-ac}^{h'-2}, \mathsf{Stock}_j[\mathsf{cur}][i]|_{j=0}^{h'-3}\}$
Initialize $\mathsf{Auth}_j[\mathsf{cur}][i] = N_j[\mathsf{cur}][i](1)$
$\mathsf{Cache}_j[\mathsf{cur}][i] = N_j[\mathsf{cur}][i](2u+3), u = 2^{h'-j-1} - 2, \ldots, 0$
$\mathsf{Stock}_j[\mathsf{cur}][i] = N_j[\mathsf{cur}][i](3)$
endfor
Initialize state $\mathsf{St}_{\mathsf{next}}$

vii. Extract $\mathsf{rt}^{(1)} = \mathsf{XMSS\ Tree}(\mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \{0,1\}^\lambda, \mathcal{F}_\lambda), k, h, G, \mathsf{H}), \mathsf{pk}[1][1], \mathsf{pk}[1][2], \ldots, \mathsf{pk}[1][2^{h'}], B_1, B_2, \ldots, B_{\mathsf{ct}})$ from Stack_1
Compute $\sigma_2 = \mathsf{WOTS}^+.\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \{0,1\}^\lambda, \mathcal{F}_\lambda), \mathsf{sk}[2][1], \mathsf{rt}^{(1)})$

viii. Extract $\mathsf{rt}^{(2)} = \mathsf{XMSS\ Tree}(\mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \{0,1\}^\lambda, \mathcal{F}_\lambda), k, h, G, \mathsf{H}), \mathsf{pk}[2][1], \mathsf{pk}[2][2], \ldots, \mathsf{pk}[2][2^{h'}], B_1, B_2, \ldots, B_{\mathsf{ct}})$ from Stack_2
Delete $\mathsf{sk}[i][p]$ and $\mathsf{pk}[i][p]$ for $i = 1, 2, p = 1, 2, \ldots, 2^{h'}$

ix. Set $\mathsf{sk}_{\mathsf{XM\text{-}VRF}} = (S_{\mathsf{cur},1}, S_{\mathsf{cur},2}, S_{\mathsf{next}}, \sigma_1, \mathsf{St}_{\mathsf{cur},1}, \mathsf{St}_{\mathsf{cur},2}, \mathsf{St}_{\mathsf{next}}, \mathsf{Stack}_1, \mathsf{Stack}_2, \mathsf{Stack}^{\mathsf{next}}, X_1, X_2, k)$ and
$\mathsf{vk}_{\mathsf{XM\text{-}VRF}} = (X_1, X_2, k, \{B_i\}_{i=1}^{\mathsf{ct}}, \mathsf{rt}^{(2)})$
Output $(\mathsf{sk}_{\mathsf{XM\text{-}VRF}}, \mathsf{vk}_{\mathsf{XM\text{-}VRF}})$.

$\mathsf{Eval}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{sk}_{\mathsf{XM\text{-}VRF}}, x) \rightarrow (y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}})$: The evaluator employs Eval algorithm on input $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{sk}_{\mathsf{XM\text{-}VRF}}$, a message $x \in \{0,1\}^m$ to produce the evaluated pseudorandom value $y_{\mathsf{XM\text{-}VRF}}$ and the corresponding proof $\pi_{\mathsf{XM\text{-}VRF}}$ of correct computation of $y_{\mathsf{XM\text{-}VRF}}$ and update the secret key $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$ by approach detailed below:

i. Compute $h' = \frac{h}{d}$ and $\mathsf{ctr}' = \mathsf{ctr} \mod 2^{h'}$ by extracting h, d, ctr from $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}$
If $\mathsf{ctr}' = 0$, then $\mathsf{ctr}' = 2^{h'}$
Compute $\mathsf{ctr}'' = \left\lceil \frac{\mathsf{ctr}}{2^{h'}} \right\rceil$
If $\mathsf{ctr}'' > 2^{h'}$, then $\mathsf{ctr}'' = \mathsf{ctr}'' \mod 2^{h'}$
Set $S_t = S_{\mathsf{cur},1}$
for $(i = 0; i < \mathsf{ctr}'; i{+}{+})$ **do**

Compute $G(S_t) = (f_{S_t}(0^\lambda)||f_{S_t}(1^\lambda)) = (S_t||R_t) \in \{0,1\}^{2\lambda}$
endfor
Compute $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$, $l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$ and $l = l_1 + l_2$
Compute $\mathsf{sk}_j = f_{R_t}(\mathsf{bin}(j-1))$ for $j = 1, 2, \ldots, l$ where $\mathsf{bin}(j-1)$ is binary representation of $j-1$
Set $S_t = X_1$
for $(p = 1; p \leq \mathsf{ctr}'; p{+}{+})$ **do**
Compute $G(S_t) = (f_{S_t}(0^\lambda)||f_{S_t}(1^\lambda)) = (S_t||R_t) \in \{0,1\}^{2\lambda}$
endfor
for $(j = 1; j \leq w-1; j{+}{+})$ **do**
Compute $G(R_t) = (f_{R_t}(0^\lambda)||f_{R_t}(1^\lambda)) = (R_t||r_j) \in \{0,1\}^{2\lambda}$
endfor
Set $\mathsf{aux} = (r, k)$ where $r = (r_1, r_2, \ldots, r_{w-1})$ and k is extracted from $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$
Set $\mathsf{sk} = \{\mathsf{aux}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$

ii. Compute $\sigma_1 \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \{0,1\}^\lambda, \mathcal{F}_\lambda), \mathsf{sk}, x)$ where $\sigma_1 \in \{0,1\}^{l \times \lambda}$
Extract $\mathsf{Auth}(\mathsf{ctr}') = \{\mathsf{Auth}_1(\mathsf{ctr}'), \mathsf{Auth}_2(\mathsf{ctr}')\}$ from $\mathsf{St}_{\mathsf{cur},1}, \mathsf{St}_{\mathsf{cur},2}$
Extract σ_2 from $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$

iii. Set $\Sigma = (\sigma_1, \mathsf{Auth}_1(\mathsf{ctr}'), \sigma_2, \mathsf{Auth}_2(\mathsf{ctr}'))$.

iv. Compute WOTS^+ public keys of the next tree of lower layer and push them into $\mathsf{Stack}^{\mathsf{next}}$ in the following manner
Set $S_t = S_{\mathsf{next}}$ and $S'_t = X_1$
for $(p = 1; p \leq \mathsf{ctr}'; p{+}{+})$ **do**
Compute $G(S_t) = (f_{S_t}(0^\lambda)||f_{S_t}(1^\lambda)) = (S_t||R_t) \in \{0,1\}^{2\lambda}$
Compute $G(S'_t) = (f_{S'_t}(0^\lambda)||f_{S'_t}(1^\lambda)) = (S'_t||R'_t) \in \{0,1\}^{2\lambda}$
endfor
for $(j = 1; j \leq w-1; j{+}{+})$ **do**
Compute $G(R'_t) = (f_{R'_t}(0^\lambda)||f_{R'_t}(1^\lambda)) = (R'_t||r'_j) \in \{0,1\}^{2\lambda}$
endfor
Set $r[[1][\mathsf{ctr}'] = (r'_1, \ldots, r'_{w-1})$,
$\mathsf{sk}_j[1][\mathsf{ctr}'] = f_{R_t}(\mathsf{bin}(j-1))$ and $\mathsf{pk}_j[1][\mathsf{ctr}'] = c_k^{w-1}(\mathsf{sk}_j[1][\mathsf{ctr}'], r[1][\mathsf{ctr}'])$
for all $j = 1, \ldots, l$ where c_k^{w-1} is the chaining function defined in **Fig. 1**
$\mathsf{aux}[1][\mathsf{ctr}'] = (r[1][\mathsf{ctr}'], k)$
$\mathsf{pk}[1][\mathsf{ctr}'] = \{\mathsf{aux}[1][\mathsf{ctr}'], (\mathsf{pk}_1[1][\mathsf{ctr}'], \ldots, \mathsf{pk}_l[1][\mathsf{ctr}'])\}$
Run $\mathsf{TreeHash}(\mathsf{Stack}^{\mathsf{next}}, \mathsf{pk}[1][\mathsf{ctr}'])$

v. Store state informations $\mathsf{St}_{\mathsf{next}}$ from stack $\mathsf{Stack}^{\mathsf{next}}$ for $\mathsf{AuthCalc}$ using the following steps:
Denote XMSS tree nodes as $N_h[a][1](j), N_h[2](j), h = 0, \ldots, h'-1$ and $j = 0, \ldots, 2^{h-h'} - 1, a = \mathsf{cur}$ (current tree) , next (next tree)
for $(i = 1; i \leq 2; i{+}{+})$ **do**
Set $\mathsf{St}_{\mathsf{next}} = \{\mathsf{Store}_j[\mathsf{next}][i]|_{j=0}^{h'-2}, \mathsf{Auth}_j[\mathsf{next}][i]|_{j=0}^{h'-1}, \mathsf{Cache}_j[\mathsf{next}][i]|_{j=h'-ac}^{h'-2}, \mathsf{Stock}_j[\mathsf{next}][i]|_{j=0}^{h'-3}\}$
Initialize $\mathsf{Auth}_j[\mathsf{next}][i] = N_j[\mathsf{next}][i](1)$
$\mathsf{Cache}_j[\mathsf{next}][i] = N_j[\mathsf{next}][i](2u+3), u = 2^{h'-j-1} - 2, \ldots, 0$

$\mathsf{Stock}_j[\mathsf{next}][i] = N_j[\mathsf{next}][i]$(3)
endfor

vi. **if** $(\mathsf{ctr}' = 2^{h'})$ **then**
Extract $\{0,1\}^\lambda \ni \mathsf{rt}^{(1)} \leftarrow \mathsf{XMSS\ Tree}(\mathsf{pp}_{\mathsf{XMSS}} = (\mathsf{pp}_{\mathsf{WOTS}^+} = (m, w, l_1, l_2, \{0,1\}^\lambda, \mathcal{F}_\lambda), k, h, G, \mathsf{H}), \mathsf{pk}[i][1], \mathsf{pk}[i][2], \ldots, \mathsf{pk}[i][2^{h'}], B_1, B_2, \ldots, B_{\mathsf{ct}})$from $\mathsf{Stack}^{\mathsf{next}}$
Set $S_t = S_{\mathsf{cur},2}$
for $(i = 0; i < \mathsf{ctr}'' + 1; i{+}{+})$ **do**
Compute $G(S_t) = (f_{S_t}(0^\lambda)||f_{S_t}(1^\lambda)) = (S_t||R_t) \in \{0,1\}^{2\lambda}$
endfor
Compute $\mathsf{sk}_j = f_{R_t}(\mathsf{bin}(j-1))$ for $j = 1, 2, \ldots, l$ where $\mathsf{bin}(j-1)$ is binary representation of $j-1$
Set $S_t = X_2$
for $(p = 1; p \leq \mathsf{ctr}'' + 1; p{+}{+})$ **do**
Compute $G(S_t) = (f_{S_t}(0^\lambda)||f_{S_t}(1^\lambda)) = (S_t||R_t) \in \{0,1\}^{2\lambda}$
endfor
for $(j = 1; j \leq w - 1; j{+}{+})$ **do**
Compute $G(R_t) = (f_{R_t}(0^\lambda)||f_{R_t}(1^\lambda)) = (R_t||r_j) \in \{0,1\}^{2\lambda}$
endfor
Set $\mathsf{aux} = (r, k)$ where $r = (r_1, r_2, \ldots, r_{w-1})$ and k is extracted from $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$
Set $\mathsf{sk}' = \{\mathsf{aux}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)\}$
$\sigma_2' = \mathsf{WOTS}^+.\mathsf{Sign}(\mathsf{pp}_{\mathsf{WOTS}^+}, \mathsf{sk}', \mathsf{rt}^{(1)})$
endif

vii. **for** $(i = 1; i \leq 2; i{+}{+})$ **do**
Run $\mathsf{AuthCalc}(\mathsf{ctr}', h, ac, \mathsf{St}_{\mathsf{cur},i}, \mathsf{H}) \rightarrow (\mathsf{Auth}(\mathsf{ctr}' + 1), \mathsf{St})$
Set $\mathsf{St}_{\mathsf{cur},i} = \mathsf{St}$
endfor
Set $\mathsf{ctr} = \mathsf{ctr} + 1$

viii. **if** $(\mathsf{ctr}' = 1)$ **then**
Pick $S'_{\mathsf{next}} \xleftarrow{\$} \{0,1\}^\lambda$
endif
if $(\mathsf{ctr}' = 2^{h'})$ **then**
Set $S_{\mathsf{cur},1} = S_{\mathsf{next}}$ and $S_{\mathsf{next}} = S'_{\mathsf{next}}$
$\mathsf{St}_{\mathsf{cur},1} = \mathsf{St}_{\mathsf{next}}$
Clear $\mathsf{Stack}^{\mathsf{next}}$, $\mathsf{St}_{\mathsf{next}}$
Set $\sigma_2 = \sigma_2'$
endif

ix. Set $\pi_{\mathsf{XM\text{-}VRF}} = \Sigma$ and compute $y_{\mathsf{XM\text{-}VRF}} = \mathsf{H}_1(\Sigma, x) \in \{0,1\}^\lambda$
Output $(y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}})$

$\mathsf{Verify}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{vk}_{\mathsf{XM\text{-}VRF}}, x, y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}}) \rightarrow 1/0$: The verifier checks the validity of the pseudorandom value $y_{\mathsf{XM\text{-}VRF}}$ on x using the verification key $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$ and the proof $\pi_{\mathsf{XM\text{-}VRF}}$ with the subsequent syntax:

i. Parse $\pi_{\mathsf{XM\text{-}VRF}} = (\sigma_1, \mathsf{Auth}_1, \sigma_2, \mathsf{Auth}_2)$
Compute $l_1 = \left\lceil \frac{m}{\log w} \right\rceil$, $l_2 = \left\lfloor \frac{\log (l_1(w-1))}{\log w} \right\rfloor + 1$ and $l = l_1 + l_2$
Extract X_1 from $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$
ii. Run $\mathsf{WOTS}^{+}\mathsf{PKfromSign}(\mathsf{pp}_{\mathsf{WOTS}^{+}}, x, \sigma_1, X_1, k) \rightarrow \mathsf{pk}^{(1)}$
Parse $\mathsf{pk}^{(1)} = \{(r,k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$
Now compute $\mathsf{L\text{-}Tree}(\mathsf{pp}_{\mathsf{WOTS}^{+}}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_l), (B_1, \ldots, B_{\lceil \log l \rceil})) \rightarrow L_1$
Run $\mathsf{Auth2Root}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{Auth}_1, L_1, B_{\lceil \log l \rceil + 1}, \ldots, B_{\mathsf{ct}}) \rightarrow x_1$
iii. Extract X_2 from $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$
Run $\mathsf{WOTS}^{+}\mathsf{PKfromSign}(\mathsf{pp}_{\mathsf{WOTS}^{+}}, x_1, \sigma_2, X_2, k) \rightarrow \mathsf{pk}^{(2)}$
Parse $\mathsf{pk}^{(2)} = \{(r,k), (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)\}$
Now compute $\mathsf{L\text{-}Tree}(\mathsf{pp}_{\mathsf{WOTS}^{+}}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_l), (B_1, \ldots, B_{\lceil \log l \rceil})) \rightarrow L_2$
Run $\mathsf{Auth2Root}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{Auth}_2, L_2, B_{\lceil \log l \rceil + 1}, \ldots, B_{\mathsf{ct}}) \rightarrow x_2$
iv. Extract $\mathsf{rt}^{(2)}$ from $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$
if $(x_2 = \mathsf{rt}^{(2)})$ **then**
proceed to next step
else
output 0
endif
v. **if** $(\mathsf{H}_1(\pi_{\mathsf{XM\text{-}VRF}}, x) = y_{\mathsf{XM\text{-}VRF}})$ **then**
output 1
else
output 0
endif

3.2 Storage Analysis

We present the storage requirements, in terms of the number of binary strings, for the secret key $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$, verification key $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$, and proof $\pi_{\mathsf{XM\text{-}VRF}}$ in Table 3.

	$\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$	$\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$	$\pi_{\mathsf{XM\text{-}VRF}}$
Storage	$9 \cdot 2^{\frac{h}{d}} + 6\frac{h}{d} + l - 9$ strings from $\{0,1\}^{\lambda}$	4 strings from $\{0,1\}^{\lambda}$ + $\frac{h}{d} + \lceil \log l \rceil$ strings from $\{0,1\}^{2\lambda}$	$2(l + \frac{h}{d})$ string from $\{0,1\}^{\lambda}$

Table 3. Storage of the secret key $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$, verification key $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$, and proof $\pi_{\mathsf{XM\text{-}VRF}}$

3.3 Correctness

Here we will discuss the correctness of our construction. More precisely we have to show the provability property of the above algorithm in succeeding manner.

i. Assume $(y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}})$ is the output of the $\mathsf{VRF.Eval}(\cdot)$ where $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$, $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$ are generated honestly.
ii. Parse $\pi_{\mathsf{XM\text{-}VRF}} = (\sigma_1, \mathsf{Auth}_1, \sigma_2, \mathsf{Auth}_2)$. So we will have all correct WOTS^{+} signatures σ_1, σ_2.

iii. Therefore the corresponding WOTS^+ public key $\mathsf{pk}^{(1)}$ can be computed from σ_1 using $\mathsf{WOTS}^+\mathsf{PKfromSign}(\cdot)$ on inputs σ_1, x. Now use $\mathsf{pk}^{(1)}$ and Auth_1 to compute the root of the lowest XMSS tree by running $\mathsf{Auth2Root}(\cdot)$, let's say the generated root is x_1.
iv. Therefore the corresponding WOTS^+ public key $\mathsf{pk}^{(2)}$ can be computed from σ_2 using $\mathsf{WOTS}^+\mathsf{PKfromSign}(\cdot)$ on inputs σ_2, x_1. Now use $\mathsf{pk}^{(2)}$ and Auth_2 to compute the root of the lowest XMSS tree by running $\mathsf{Auth2Root}(\cdot)$, let's say the generated root is x_2 which should be the root node of the top XMSS tree and this root is also stored in the $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$ as $\mathsf{rt}^{(2)}$.
v. Since $(y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}})$ is the output of the $\mathsf{VRF.Eval}(\cdot)$ where $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$, $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}$ are generated honestly, so the comparison check $x_2 \stackrel{?}{=} \mathsf{rt}^{(2)}$ and checking of $\mathsf{H}_1(\pi_{\mathsf{VRF}}, x) \stackrel{?}{=} y_{\mathsf{VRF}}$ both will pass. So $\mathsf{VRF.Verify}(\cdot)$ will output 1. Hence our algorithm satisfies provability property.

3.4 Security

In this section we analyze the uniqueness and pseudorandomness of our XM-VRF. First, we deep dive into the most critical property uniqueness. Next, we will focus on the other property pseudorandomness.

Theorem 4. *If hash function* H *and function family* $\mathcal{F}_\lambda$ *in the public parameter* $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}$ *are collision resistant , then* XM-VRF *satisfies the uniqueness property in the random oracle model.*

Proof. We prove uniqueness of XM-VRF by reduction technique. Let $\mathcal{A}$ be a PPT adversary against the uniqueness of our XM-VRF scheme. We will show that if $\mathcal{A}$ can break the uniqueness of XM-VRF, then there is an adversary $\mathcal{B}$ which can break the uniqueness of the WOTS^+.
Suppose $\mathcal{A}$ can break the uniqueness of XM-VRF i.e, $\mathcal{A}$ have two different evaluations and corresponding proofs for the same input $(y_{\mathsf{VRF},1}, y_{\mathsf{VRF},2}, \pi_{\mathsf{VRF},1}, \pi_{\mathsf{VRF},2}, x)$ satisfying

$$\mathsf{VRF.Verify}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{pk}_{\mathsf{XM\text{-}VRF}}, x, y_{\mathsf{VRF},1}, \pi_{\mathsf{VRF},1}) = \mathsf{VRF.Verify}(\mathsf{pp}_{\mathsf{XM\text{-}VRF}}, \mathsf{pk}_{\mathsf{XM\text{-}VRF}}, x, y_{\mathsf{VRF},2}, \pi_{\mathsf{VRF},2}) = 1$$

when $y_{\mathsf{VRF},1} \neq y_{\mathsf{VRF},2}$
Now recall that $y_{\mathsf{VRF},i} = \mathsf{H}_1(\Sigma_i, x)$ and $\pi_{\mathsf{VRF},i} = \Sigma_i$ for $i = 1, 2$. Since $y_{\mathsf{VRF},1} \neq y_{\mathsf{VRF},2}$, so $\Sigma_1 \neq \Sigma_2$.
We also know that $\Sigma_j = (\sigma_1^j, \mathsf{Auth}_1^j, \sigma_2^j, \mathsf{Auth}_2^j)$ for $j = 1, 2$. Since same ctr or leaf index $\mathsf{pp}_{\mathsf{XM\text{-}VRF}}$ is used for VRF.Eval, same tree root is used and the used hash function H is collision resistant, so $\mathsf{Auth}_i^1 = \mathsf{Auth}_i^2$ for $i = 1, 2$. Therefore $\Sigma_1 \neq \Sigma_2$ implies $\sigma_i^1 \neq \sigma_i^2$ for atleast one i. Now the adversary $\mathcal{B}$ returns σ_i^1, σ_i^2 to break the uniqueness of WOTS^+. But by Theorem 1, WOTS^+ is unique if function family $\mathcal{F}_\lambda$ is collision resistant. So contradiction. Hence XM-VRF is unique. □

Theorem 5. XM-VRF *satisfies the pseudorandomness property in the random oracle model.*

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary against the pseudorandomness of our XM-VRF scheme. Recall that $y_{\mathsf{VRF}} = \mathsf{H}_1(\Sigma, x)$ where H_1 is modeled as a random oracle. Also recall that the signature $\Sigma = (\sigma_1, \mathsf{Auth}_1, \sigma_2, \mathsf{Auth}_2)$ contains WOTS^+ signatures σ_1, σ_2 which is obtained by iteratively hashing a set of completely independent and uniformly random λ-bit strings unknown to $\mathcal{A}$. Consequently, each σ_i generates a bit string that is statistically indistinguishable from a random sequence, and this string is an element of the set Σ. Hence, $\mathcal{A}$ can distinguish y_{VRF} from a uniformly random value only if $\mathcal{A}$ has queried H_1 on the input (Σ, x). However, this occurs with negligible probability, as $\mathcal{A}$ is unable to query the signing oracle on x. From this point, the pseudorandomness property can be established. We demonstrate this by contradiction, employing the following approach. Suppose the adversary $\mathcal{A}$ wins the experiment $\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{XM\text{-}VRF}}(\lambda)$ with probability $> \frac{1}{2} + \mathsf{negl}(\lambda)$. So $\mathcal{A}$ can distinguish between λ-bit random string and $\mathsf{H}_1(\Sigma, x)$. This implies contradiction to the fact that H_1 is a random oracle. Therefore the adversary can win the experiment $\mathsf{Exp\text{-}PRand}_{\mathcal{A}}$ with probability $\leq \frac{1}{2} + \mathsf{negl}(\lambda)$. Hence pseudorandomness holds for our XM-VRF. □

Theorem 6. XM-VRF *satisfies the forward security property.*

Proof. XM-VRF employs a stateful forward secure PRG G in their construction, which updates it's internal state after each use. In XM-VRF, the evaluation key is refreshed using the stateful pseudorandom generator G. As a result, an adversary cannot infer earlier outputs of the PRG G from pseudorandom values generated later. Therefore, even if a later key is compromised, all previous evaluations remain secure. Hence, XM-VRF achieves forward security. □

4 Application of XM-VRF in Blockchain

VRFs serve as a fundamental component in PoS blockchains, facilitating secure and private cryptographic sortition for selecting block proposers and voting committees [26, 18, 31] and are further employed in smart contract applications, including decentralized online lotteries. VRFs underpin decentralized lotteries and random-beacon schemes, delivering trustworthy, unpredictable randomness that is vital for many blockchain mechanisms notably sharding, where secure and unguessable shard assignments are essential. Dfinity Internet Computer leverages a VRF-based random beacon to generate decentralized randomness, which is subsequently utilized for task assignment, transaction validation, and maintaining network security. For more details look into [30]. In Ethereum 2.0, VRFs are used to securely and randomly assign validators to different shards. Our study emphasizes the role of VRF in enabling cryptographic sortition for blockchain consensus.

Cryptographic sortition, introduced by Algorand, allows users to privately determine their own eligibility to participate in the consensus protocol. In this process, participants remain anonymous to all others including potential adversaries. In Algorand, the committee-based consensus mechanism introduced

by Gilad et al. [26] leverages a VRF to perform cryptographic sortition. Moreover, the VRF mechanism in Algorand facilitates a secure and unbiased selection of committee members in a non-interactive manner, where each participant's chance of selection is proportional to their stake. This randomized committee selection in Algorand also mitigates targeted attacks by preventing adversaries from identifying specific committee members in advance. Furthermore, the incorporation of a VRF in Algorand's consensus protocol enhances scalability and performance, enabling the system to efficiently support millions of users. At its core, Algorand employs a Byzantine Agreement protocol [14], run by a randomly chosen committee of users for each consensus round. More precisely, this protocol employs XM-VRF in the following manner:

i. Every user possesses a secret–public key pair $(\mathsf{sk}_{\mathsf{XM\text{-}VRF}}, \mathsf{pk}_{\mathsf{XM\text{-}VRF}})$.
ii. Let A denote a block to be appended to the blockchain.
iii. To decide if they are chosen to join the committee for validating block A, each user follows these procedures:
   (a) The user computes $(y_{\mathsf{XM\text{-}VRF}}, \pi_{\mathsf{XM\text{-}VRF}}) \leftarrow \mathsf{XM\text{-}VRF.Eval}(\mathsf{sk}_{\mathsf{XM\text{-}VRF}}, Q)$ using their secret key $\mathsf{sk}_{\mathsf{XM\text{-}VRF}}$ and a publicly known randomness value Q and output a pseudorandom value $y_{\mathsf{XM\text{-}VRF}}$ along with a proof $\pi_{\mathsf{XM\text{-}VRF}}$ that the computation was performed correctly.
   (b) The user then verifies whether $y_{\mathsf{XM\text{-}VRF}}$ falls within the target range $[0, P]$, where P is a parameter determined by the user's current stake. If this condition is satisfied, the user is selected as a committee member for block A.
iv. Any user can confirm committee membership by executing the VRF verification algorithm with $\mathsf{vk}_{\mathsf{XM\text{-}VRF}}, Q, y_{\mathsf{XM\text{-}VRF}}$ and $\pi_{\mathsf{XM\text{-}VRF}}$ as input and checking that $y_{\mathsf{XM\text{-}VRF}} \in [0, P]$.

5 Conclusion

In this work, we propose a straightforward and efficient method to realize VRF functionality tailored for blockchain-based leader election. Our construction addresses the challenge of long key generation times, enabling a single key pair to produce multiple outputs across different election rounds. This construction can allow a single key pair to provide 2^{80} outputs, an almost unlimited number of outputs virtually. Our approach avoids the race conditions seen in Bitcoin and can be built using standard (post-quantum) cryptographic primitives.

Acknowledgment

We would like to thank the Prime Minister's Research Fellowship (PMRF), Government of India, for providing financial support for this research. The continued encouragement and resources offered under this fellowship have been instrumental in advancing the progress of this work.

References

1. Abdalla, M., Catalano, D., Fiore, D.: Verifiable random functions from identity-based key encapsulation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 554–571. Springer (2009)
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 430–454. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
3. Alkandari, A.A., Al-Shaikhli, I.F., Alahmad, M.A.: Cryptographic hash function: A high level view. In: 2013 International Conference on Informatics and Creative Multimedia. pp. 128–134 (2013). https://doi.org/10.1109/ICICM.2013.29
4. Bellare, M., Yee, B.: Forward-security in private-key cryptography. In: Proceedings of the 2003 RSA Conference on The Cryptographers' Track. p. 1–18. CT-RSA'03, Springer-Verlag, Berlin, Heidelberg (2003)
5. Bitansky, N.: Verifiable random functions from non-interactive witness-indistinguishable proofs. Journal of Cryptology **33**(2), 459–493 (2020)
6. Bodaghi, O., Safavi-Naini, R.: Short paper: Breaking x-vrf, a post-quantum verifiable random function. In: Clark, J., Shi, E. (eds.) Financial Cryptography and Data Security. pp. 90–100. Springer Nature Switzerland, Cham (2025)
7. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Annual international cryptology conference. pp. 213–229. Springer (2001)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22. pp. 416–432. Springer (2003)
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
10. Brakerski, Z., Goldwasser, S., Rothblum, G.N., Vaikuntanathan, V.: Weak verifiable random functions. In: Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings 6. pp. 558–576. Springer (2009)
11. Buchmann, J., Dahmen, E., Hülsing, A.: Xmss-a practical forward secure signature scheme based on minimal security assumptions. In: Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4. pp. 117–129. Springer (2011)
12. Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. In: Post-Quantum Cryptography: Second International Workshop, PQCrypto 2008 Cincinnati, OH, USA, October 17-19, 2008 Proceedings 2. pp. 63–78. Springer (2008)
13. Buser, M., Dowsley, R., Esgin, M.F., Kasra Kermanshahi, S., Kuchta, V., Liu, J.K., Phan, R.C.W., Zhang, Z.: Post-quantum verifiable random function from symmetric primitives in pos blockchain. In: European Symposium on Research in Computer Security. pp. 25–45. Springer (2022)
14. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. p. 173–186. OSDI '99, USENIX Association, USA (1999)
15. Chase, M., Lysyanskaya, A.: Simulatable vrfs with applications to multi-theorem nizk. In: Annual International Cryptology Conference. pp. 303–322. Springer (2007)

16. Chen, J., Gorbunov, S., Micali, S., Vlachos, G.: Algorand agreement: Super fast and partition resilient byzantine agreement. Cryptology ePrint Archive (2018)
17. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 66–98. Springer International Publishing, Cham (2018)
18. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37. pp. 66–98. Springer (2018)
19. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6. pp. 1–17. Springer (2002)
20. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: International Workshop on Public Key Cryptography. pp. 416–431. Springer (2005)
21. Esgin, M.F., Ersoy, O., Kuchta, V., Loss, J., Sakzad, A., Steinfeld, R., Yang, X., Zhao, R.K.: A new look at blockchain leader election: Simple, efficient, sustainable and post-quantum. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. pp. 623–637 (2023)
22. Esgin, M.F., Kuchta, V., Sakzad, A., Steinfeld, R., Zhang, Z., Sun, S., Chu, S.: Practical post-quantum few-time verifiable random function with applications to algorand. In: International Conference on Financial Cryptography and Data Security. pp. 560–578. Springer (2021)
23. Esgin, M.F., Steinfeld, R., Liu, D., Ruj, S.: Efficient hybrid exact/relaxed lattice proofs and applications to rounding and vrfs. In: Annual International Cryptology Conference. pp. 484–517. Springer (2023)
24. Fuchsbauer, G.: Constrained verifiable random functions. In: Security and Cryptography for Networks: 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings 9. pp. 95–114. Springer (2014)
25. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. p. 51–68. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132757, https://doi.org/10.1145/3132747.3132757
26. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th symposium on operating systems principles. pp. 51–68 (2017)
27. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: Nsec5: Provably preventing dnssec zone enumeration (2015)
28. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on computing **17**(2), 281–308 (1988)
29. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II 15. pp. 537–566. Springer (2017)
30. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)

31. Hellebrandt, L., Homoliak, I., Malinka, K., Hanáček, P.: Increasing trust in tor node list using blockchain. In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 29–32. IEEE (2019)
32. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29. pp. 656–672. Springer (2010)
33. Hülsing, A.: W-ots+–shorter signatures for hash-based signature schemes. In: Progress in Cryptology–AFRICACRYPT 2013: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings 6. pp. 173–188. Springer (2013)
34. Jager, T.: Verifiable random functions from weaker assumptions. In: Theory of Cryptography: 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II 12. pp. 121–143. Springer (2015)
35. Kaliski, B.: Pseudorandom Function, pp. 995–995. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5_427
36. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 525–537. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243805, https://doi.org/10.1145/3243734.3243805
37. Lai, Y.F.: Capybara and tsubaki: verifiable random functions from group actions and isogenies. Cryptology ePrint Archive (2023)
38. Leroux, A.: Verifiable random function from the deuring correspondence and higher dimensional isogenies (2023)
39. Levin, S., Pedersen, R.: Faster proofs and vrfs from isogenies. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025. pp. 307–340. Springer Nature Singapore, Singapore (2026)
40. Lysyanskaya, A.: Unique signatures and verifiable random functions from the dh-ddh separation. In: Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22. pp. 597–612. Springer (2002)
41. Merkle, R.C.: A digital signature based on a conventional encryption function. In: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. p. 369–378. CRYPTO '87, Springer-Verlag, Berlin, Heidelberg (1987)
42. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th annual symposium on foundations of computer science (cat. No. 99CB37039). pp. 120–130. IEEE (1999)
43. Papadopoulos, D., Wessels, D., Huque, S., Naor, M., Včelák, J., Reyzin, L., Goldberg, S.: Making nsec5 practical for dnssec. Cryptology ePrint Archive (2017)
44. Proelss, J., Sévigny, S., Schweizer, D.: Gamefi: The perfect symbiosis of blockchain, tokens, defi, and nfts? International Review of Financial Analysis **90**, 102916 (2023). https://doi.org/https://doi.org/10.1016/j.irfa.2023.102916, https://www.sciencedirect.com/science/article/pii/S1057521923004325
45. Robshaw, M.J.B.: One-Way Function, pp. 887–888. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5_467
46. Yang, R., Au, M.H., Zhang, Z., Xu, Q., Yu, Z., Whyte, W.: Efficient lattice-based zero-knowledge arguments with standard soundness: construction and applications.

In: Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39. pp. 147–175. Springer (2019)

Appendix

A Signature Scheme

Definition 2 (Digital Signature Scheme). [28] *A digital signature scheme corresponding to a given message space $\mathcal{M}$ comprises a suite of probabilistic polynomial-time (PPT) algorithms, represented as* $\text{DSS}= (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$*, with the following requirements:*

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$*: A trusted party taking security parameter λ as input, outputs a public parameter* pp *and makes it publicly available.*

$\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{sk}, \mathsf{pk})$*: On input* pp*, a user runs this algorithm to generate a secret key and public key pair* $(\mathsf{sk}, \mathsf{pk})$ *for signing and verification respectively. The secret key* sk *is kept secret to the user while the public key* pk *is made public.*

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M) \rightarrow \sigma$*: Taking input* pp, sk *and a message $M \in \mathcal{M}$, the signer executes this algorithm to generate a signature σ on the message M. The message-signature pair (M, σ) is sent over a public channel.*

$\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, M, \sigma) \rightarrow 1/0$*: Given inputs* pp, pk*, a message $M \in \mathcal{M}$ and a signature σ, a verifier checks the validity of the signature σ on M. If the signature is valid, it outputs* 1*, otherwise* 0*.*

Correctness. *For all* $\mathsf{pp} \leftarrow \text{DSS}.\mathsf{Setup}(1^\lambda)$*, all* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \text{DSS}.\mathsf{KeyGen}(\mathsf{pp})$ *and all signature* $\sigma \leftarrow \text{DSS}.\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M)$*, it must satisfy that*

$$\text{DSS}.\mathsf{Vrfy}(\mathsf{pp}, \mathsf{pk}, M, \sigma) \rightarrow 1$$

EU-CMA Security: The standard security notion for the digital signature scheme $\text{DSS}= (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ is *existential unforgeability under adaptive chosen message attacks* (EU-CMA).

Definition 3 (EU-CMA). [28] *Let $\lambda, t, q \in \mathbb{N}$, $t, q = \mathsf{poly}(\lambda)$. Digital Signature Scheme* $\text{DSS}= (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is EU-CMA secure if*

$$\mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A},\text{DSS}}(\lambda; t, q) = \max_{\mathcal{A}} \left\{ \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A},\text{DSS}}(\lambda) = 1 \right] \right\} \leq \mathsf{negl}(\lambda)$$

for all probabilistic adversaries $\mathcal{A}$, running in time $\leq t$ and making at most q queries to $\mathcal{O}_{\mathsf{Sign}(\mathsf{sk},\cdot)}$ in the experiment $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A},\text{DSS}}(\lambda)$ between a challenger and an adversary defined in Experiment 1 below for some negligible function $\mathsf{negl}(\cdot)$.

Experiment 1: $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A},\text{DSS}}(\lambda)$

Setup: The challenger takes input λ to generate public parameter $\mathsf{pp} \leftarrow \text{DSS.Setup}(1^\lambda)$. Then challenger produces key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \text{DSS.KeyGen}(\mathsf{pp})$. It maintains a list Qlist and initializes $\mathsf{Qlist} = \phi$. It gives the public parameter pp and pk to adversary $\mathcal{A}$ and keep sk secret to itself.
Query Phase: The adversary $\mathcal{A}$ repeats polynomial times sign query on messages of its choice to obtain signatures from the challenger. For a signature query on a message $M \in \mathcal{M}$ from $\mathcal{A}$, the challenger generates signature $\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}(\mathsf{sk},\cdot)}}(M)$ and send it to $\mathcal{A}$ and update $\mathsf{Qlist} = \mathsf{Qlist} \cup \{M\}$.
Forgery: The challenger in turn checks whether adversary $\mathcal{A}$ submits a forgery (M^*, σ^*) to $(\text{DSS.Vrfy}(\mathsf{pp}, \mathsf{pk}, M^*, \sigma^*) \rightarrow 1$ and $M^* \notin \mathsf{Qlist})$. If verification succeeds, then return 1, otherwise return 0.

Definition 4 (Unique Signature Scheme). [36] *Let* DSS$=$ $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a signature scheme and* $\Sigma(m, \mathsf{pk}) = \{\sigma : \text{DSS.Vrfy}(\mathsf{pp}, \mathsf{pk}, m, \sigma) = 1\}$. *Then* DSS *is said to be a unique signature if* $|\Sigma(m, \mathsf{pk})| = 1$ *for all* m *and* pk.

B Function Family

Definition 5 (Second Preimage Resistant). [3] *A function* $f : A \rightarrow B$ *is said to be second preimage resistant if given* $x \in A$ *there exist no polynomial time adversary which can find another* $x' \in A$ *such that* $f(x) = f(x') \in B$. *If each* $f \in \mathcal{F}$ *is second preimage resistant for a function family* $\mathcal{F}$, *then* $\mathcal{F}$ *is second preimage resistant (*SPR*).*

Definition 6 (One-way). [45] *A function* f *is a one-way function if the following holds*

i. The description of $f : A \rightarrow B$ *is publicly known and does not require any secret information for its operation.*
ii. Given $x \in A$, *it is easy to compute* $f(x) \in B$.
iii. Given $y \in B$, *it is hard to find an* $x \in A$ *such that* $f(x) = y \in B$. *More precisely, any polynomial time algorithm succeeds in inverting* f *with negligible probability.*

If each $f \in \mathcal{F}$ *is one-way for a function family* $\mathcal{F}$, *then* $\mathcal{F}$ *is one-way (*OW*).*

Definition 7 (Undetectable). [35] *A function* $f : A \rightarrow B$ *is undetectable if there is no polynomial time distinguisher which can distinguish between* $f(x) \in B$ *and* $r \xleftarrow{\$} B$. *If each* $f \in \mathcal{F}$ *is undetectable for a function family* $\mathcal{F}$, *then* $\mathcal{F}$ *is undetectable (*UD*).*

C Stateful Forward Secure PRG

Bellare et al. [4] proposed stateful pseudo-random generators (PRG) where the maximum number of outputs that the PRG is permitted to generate is considered as a parameter of this stateful PRG. Given a state, it outputs a new state and some output bits.

Definition 8 (Stateful PRG). [4] *A stateful PRG G with security parameter λ is mapping from $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$ which takes an initial state $s \in \{0,1\}^\lambda$ as input and generates a new state $s' \in \{0,1\}^\lambda$ and $r \in \{0,1\}^\lambda$ as output. And Q is the maximum number of outputs the PRG is allowed to produce.*

A stateful PRG G must satisfy the following properties:

Pseudorandomness: Let $\mathcal{A}$ be a polynomial-time adversary playing the experiment $\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{PRG}}(\lambda)$ in **Experiment 2**.

Experiment 2: $\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{PRG}}(\lambda)$

Setup: The challenger first choose $s \in \{0,1\}^\lambda$ and $b \in \{0,1\}$ uniformly random.
Query Phase: The adversary $\mathcal{A}$ repeats polynomial times query to oracle $\mathcal{O}^G(s)$ to obtain random λ-bit from the challenger. For all queries oracle $\mathcal{O}^G(s)$ uses either $r_0 \in \{0,1\}^\lambda$ where $G(s) = (s', r_0)$ or randomly chosen $r_1 \in \{0,1\}^\lambda$ to respond $r_b \in \{0,1\}^\lambda$ to the adversary $\mathcal{A}$. So, all queries to $\mathcal{O}^G(s)$ should be indistinguishable from random.
Response: The adversary $\mathcal{A}$ submits a response $b' \in \{0,1\}$ for that challenge. If $b = b'$, then return 1, otherwise return 0.

A stateful PRG G is said to satisfy pseudorandomness if

$$\mathsf{Adv}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{PRG}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{PRand}}_{\mathcal{A},\mathsf{PRG}}(\lambda) = 1\right] \right\} \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Forward Security: Let $\mathcal{A}$ be a polynomial-time adversary playing the following experiment $\mathsf{Exp}^{\mathsf{Forward}}_{\mathcal{A},\mathsf{PRG}}(\lambda)$ in **Experiment 3**:

Experiment 3: $\mathsf{Exp}^{\mathsf{Forward}}_{\mathcal{A},\mathsf{PRG}}(\lambda)$

Setup: The challenger first choose $s \in \{0,1\}^\lambda$ and $b \in \{0,1\}$ uniformly random.
Query Phase: The adversary $\mathcal{A}$ repeats polynomial times query to oracle $\mathcal{O}^G(s)$ to obtain random λ-bit from the challenger. For all queries oracle $\mathcal{O}^G(s)$ uses either $r_0 \in \{0,1\}^\lambda$ where $G(s) = (s', r_0)$ or randomly chosen $r_1 \in \{0,1\}^\lambda$ to respond $r_b \in \{0,1\}^\lambda$ to the adversary $\mathcal{A}$. So, all queries to $\mathcal{O}^G(s)$ should be indistinguishable from random. Adversary $\mathcal{A}$ has access to an extra oracle $\mathcal{O}^{\mathsf{State}}()$ to obtain the current state s.
Response: The adversary $\mathcal{A}$ submits a response $b' \in \{0,1\}$ for that challenge. If $b = b'$, then return 1, otherwise return 0.

A stateful PRG G is said to satisfy (t,q,ϵ)-forward security if for ay adversary $\mathcal{A}$ running in time atmost t, making atmost q queries to $\mathcal{O}^G(\cdot)$ and one call to $\mathcal{O}^{\mathsf{State}}$

$$\mathsf{Adv}^{\mathsf{Forward}}_{\mathcal{A},\mathsf{PRG}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Forward}}_{\mathcal{A},\mathsf{PRG}}(\lambda) = 1\right] \right\} \leq \frac{1}{2} + \epsilon$$