A New Look at Blockchain Leader Election: Simple, Efficient, Sustainable and Post-Quantum

Muhammed F. Esgin*†, Oğuzhan Ersoy‡, Veronika Kuchta§, Julian Loss¶, Amin Sakzad*,
Ron Steinfeld*, Wayne Yang* and Raymond K. Zhao*
*Monash University, Australia
†CSIRO's Data61, Australia
‡Radboud University and Delft University of Technology, The Netherlands
§The University of Queensland, Australia
¶CISPA Helmholtz Center for Information Security, Germany

***Abstract*—In this work, we study the blockchain leader election problem. The purpose of such protocols is to elect a leader who decides on the next block to be appended to the blockchain, for each block proposal round. Solutions to this problem are vital for the security of blockchain systems. We introduce an efficient blockchain leader election method with security based solely on standard assumptions for cryptographic hash functions (rather than public-key cryptographic assumptions) and that does not involve a racing condition as in Proof-of-Work based approaches. Thanks to the former feature, our solution provides the highest confidence in security, even in the post-quantum era. A particularly scalable application of our solution is in the Proof-of-Stake setting, and we investigate our solution in the Algorand blockchain system. We believe our leader election approach can be easily adapted to a range of other blockchain settings.**

At the core of Algorand's leader election is a verifiable random function (VRF). Our approach is based on introducing a simpler primitive which still suffices for the blockchain leader election problem. In particular, we analyze the concrete requirements in an Algorand-like blockchain setting to accomplish leader election, which leads to the introduction of *indexed VRF* (iVRF). An iVRF satisfies modified uniqueness and pseudorandomness properties (versus a full-fledged VRF) that enable an efficient instantiation based on a hash function without requiring any complicated zero-knowledge proofs of correct PRF evaluation. We further extend iVRF to an *authenticated iVRF with forward-security*, which meets all the requirements to establish an Algorand-like consensus. Our solution is simple, flexible and incurs only a 32-byte additional overhead when combined with the current best solution to constructing a forward-secure signature (in the post-quantum setting).

We implemented our (authenticated) iVRF proposal in C language on a standard computer and show that our proposal significantly outperforms other quantum-safe VRF proposals in almost all metrics. Particularly, iVRF evaluation and verification can be executed in 0.02 ms, which is even faster than ECVRF used in Algorand.

***Keywords*—Blockchain, Leader Election, Verifiable Random Function, Post-Quantum, Algorand**

I. Introduction

In a blockchain system, a number of block creators work collectively on producing new blocks and appending them to the blockchain. Such systems rely on consensus protocols to agree on the proper sequencing of blocks. The main purpose of a consensus protocol is to elect a leader, who decides on the new block to be appended, and in some cases, to additionally elect a committee, who validates newly produced blocks. In Proof-of-Stake consensus protocols, for example, the leader is elected among the stakeholders with probability proportional to the amount of owned stakes.

Due to their critical roles, it is vital for the security of a blockchain system that leaders are elected from a set of honest users that may not be easily corrupted by an adversary. To accomplish such security requirements, there are various cryptographic techniques employed by different blockchain environments, e.g., [1]–[5]. For example, Algorand, aiming for an environment-friendly solution, employs a cryptographic sortition technique, where a user self-determines if they are elected as a leader using a Verifiable Random Function (VRF) [6]. The goal of the VRF is to introduce a way for each user to produce a *unique* lottery ticket whose validity can be publicly verified via a proof shared by the user. In the case a user has a winning ticket, they publish the ticket as well as a certifying proof, which together constitute the VRF output. The use of VRF in consensus protocols supports scaling and high performance, and allows the Algorand blockchain to support millions of users. VRFs are also used as a core cryptographic primitive in other blockchain systems such as Ouroboros Praos [1] (used in Cardano), Dfinity [4], Rangers Protocol [7] and Filecoin [8].

The VRF and signature solutions employed by Algorand, namely ECVRF [9] and Ed25519 [10], rely on the hardness of discrete logarithm problem (DLP), which is susceptible to quantum attacks. As discussed in [11], the lack of post-quantum security in an Algorand-like blockchain context may be catastrophic, leading to a complete adversarial re-write of the blockchain history. Particularly, the authors in [11] discuss that even if a forward-secure signature is used to secure the blockchain system, a quantum adversary may break DLP to recover users' master secret keys from their master public keys. As such, the adversary can corrupt any user at any time (including previous rounds) by generating the round keys from the master secret key even when the round secret keys are physically deleted by the users. Hence, it is important to migrate to post-quantum solutions, particularly for the consensus part, to make sure at least that the security

of prior rounds cannot be compromised[1].

A major bottleneck against making Algorand-like consensus protocols post-quantum is the lack of an *efficient* post-quantum replacement of its VRF. There have been attempts to construct practically efficient post-quantum VRFs, but all of them have significant disadvantages compared to our solution, which we discuss next.

The first practical post-quantum VRF proposal, named LB-VRF, was introduced by Esgin et al. in [11]. This construction is based on (module) lattices and is relatively efficient for the lattice setting. In particular, an LB-VRF proof is about 5 KB, which is somewhat larger than an ordinary signature based on the same assumptions (at around 2.5–3 KB). A significant limitation of LB-VRF is that it is only *few-time*, meaning that each VRF key pair can only be used to generate a few VRF outputs, particularly just one for the most efficient instantiation. This leads to significant issues because one has to find custom-designed methods to handle key refreshing almost every round (i.e., every 5 seconds). Moreover, since the new key needs to be communicated almost every round, the actual communication cost rises by the size of a public key to more than 8.3 KB. Another lattice-based VRF proposal, named LaV, was recently introduced by Esgin et al. in [12]. Despite the fact that LaV is a standard VRF (i.e., can directly replace ECVRF used in Algorand in terms of functionality), its main drawback is the proof size at 12 KB.

By employing hash-based cryptography, Buser et al. [13] introduced X-VRF, a post-quantum VRF construction based on the XMSS signature [14]. This VRF proposal is stateful (like XMSS), but this limitation is not a significant concern in the blockchain applications (at least for its use in Algorand). The main drawback of X-VRF (compared to our solution) is the significant communication overhead at about 3 KB. Another solution in [13] is a standard VRF proposal, named SL-VRF, but its communication cost is significantly larger, at about 40 KB. A summary of the performance of these existing VRF proposals is provided in Table I.

Given the above state of affairs for post-quantum VRFs, a natural question that we ask in this work is the following.

Do we really need a full VRF to realize an Algorand-like consensus? If not, what is precisely the tool needed for that purpose and how can we construct it from the most basic cryptographic primitives?

We emphasize the importance of simplicity, efficiency, sustainability and (long-term) reliability as we are after a solution that (i) can be readily deployed into Algorand's network, (ii) does not require wasting natural resources, and (iii) offers the strongest security in the post-quantum era. As a result, we build our solution for the VRF functionality based solely on hash functions.

Before going into more details of our solution, we present independent views on Bitcoin's, Algorand's, and our approaches to solving the leader election[2]/lottery problem on blockchain. These will be helpful in understanding our high-level approach without getting into technical details.

A view of Bitcoin's approach. A common feature of all three approaches that we discuss now is that the blockchain protocol generates a random "magic" number, say Q_n, at each protocol round n. In Bitcoin's approach, this magic number is used to select a random function H_{Q_n} from a large family of hash functions. Then, the idea of Bitcoin's Proof-of-Work (PoW) approach is to have users race *real-time* to find a "lucky" input, x, that maps under H_{Q_n} to one of the target values such as $\mathsf{H}_{Q_n}(x) < v$ for some threshold v. The main drawback, well-known in the community, is the tremendous energy consumption and the waste of resources due to the real-time racing as the more computational power one spends, the more chance they have in winning the race.

A view of Algorand's approach. Algorand aims to solve the unsustainability issue of Bitcoin's PoW by moving to a Proof-of-Stake (PoS) based approach. In Algorand, users fix their key pairs in advance of participating in the consensus, say $(\mathsf{pk}_i, \mathsf{sk}_i)$ for user i. Effectively, fixing the keys seals everyone's fate in that now each user has an internal secret function $\mathsf{H}_{\mathsf{sk}_i}$, which they cannot change since the corresponding public key pk_i acts as a commitment to $\mathsf{H}_{\mathsf{sk}_i}$. When the blockchain protocol generates the random magic number Q_n for round n, then each user locally computes $\mathsf{H}_{\mathsf{sk}_i}(Q_n)$ and checks whether the result is a "winning ticket". The probability of success in the latter check is based on the amount of stake one has. If the check is successful, then they can generate a (zero-knowledge) proof to show that they are indeed a winner of the lottery for round n.

Views of our approach. Our approach in a way is a new look at a combination of the above two approaches. In particular, we first want to deviate from real-time racing in Bitcoin while still making use of the magic number Q_n to choose a random function H_{Q_n} at each block generation round. Since we have these functions chosen at each round without a particular user's control,[3] what we do is simply ask every user (participating in the consensus) to commit to an input of their choice *in advance* of that round. In particular, at some round $t < n$ (or earlier), each user i commits to their input x_i on blockchain and simply wins the lottery if $\mathsf{H}_{Q_n}(x_i) = v$ for a target value v. To prove that user i indeed won the lottery, they simply publish x_i and everyone can straightforwardly check the claim by calculating $\mathsf{H}_{Q_n}(x_i)$. Overall, for improved efficiency, we ask users to commit to a set of inputs $x_{i,j}$'s early on and use the n-th value $x_{i,n}$ at round n, where the use of the n-th value can be verified so that users do not get to choose between multiple $x_{i,j}$'s.

[1] In recognition of the importance of post-quantum security, Algorand has recently (concurrent to our work) introduced *post-quantum secure* state proofs. See https://developer.algorand.org/docs/get-details/algorand_consensus/#state-proof-keys (accessed on July 26, 2022).

[2] We note that the high-level approaches we discuss (Bitcoin's, Algorand's and ours) do not guarantee a *single* leader election (and hence, the term 'leader election' is not used to mean 'single leader election'). Thanks to the flexibility of our approach, if multiple potential leaders arise in our protocol, one can employ, for example, the techniques in Algorand (discussed in Sec. VII) to recognise one of them as the true leader.

[3] Here, we are assuming the ideal case where Q_n is generated truly at random by some means. Of course, this is not possible to achieve in practice, but this issue is outside of our simplified discussion here. In our actual protocol, the magic number will be generated similar to Algorand and our approach is flexible enough to support different ways to generate Q_n.

TABLE I. PERFORMANCE COMPARISON OF OUR IVRF IN SEC. IV-A (WITH $N = 2^{18}$ EVALUATIONS) TO OTHER QUANTUM-SAFE VRF PROPOSALS AND (NON-QUANTUM-SAFE) ECVRF. THE SIZES ARE GIVEN IN BYTES AND THE TIMES ARE IN MILLISECONDS. FOR ECVRF, WE TAKE THE RESULTS REPORTED IN [13]. FOR OUR IVRF, THE VRF VALUE CAN BE COMPUTED FROM THE PROOF AND HENCE ITS SIZE IS GIVEN AS ZERO.

Scheme	Pub. Key Size	Proof Size	VRF Size	Keygen Time	Eval Time	Verify Time	# of Evaluations	Security Basis
SL-VRF [13]	48	40000	32	0.38	765.00	475.00	Unlimited	LowMC block cipher [15]
LaV [12]	6420	11980	124	-	-	-	Unlimited	Lattices
LB-VRF [11]	3320	4940	84	0.33	3.10	1.30	1	Lattices
X-VRF [13]	64	2720	32	426000.00	0.74	0.90	$N = 2^{18}$	Hash
Our iVRF	32	608	0	$<$ 3087.00	0.01	0.02	$N = 2^{18}$	Hash
ECVRF	32	80	32	0.05	0.10	0.10	Unlimited	Discrete log. (not quantum-safe)

A "dual" view of our approach is that each user i commits to a one-time function $\mathsf{H}_{x_{i,n}}$ for each round n in advance, and just evaluates $\mathsf{H}_{x_{i,n}}(Q_n)$ to check lottery winning similar to Algorand. However, the major difference of our approach from Algorand's is that we do not require a (zero-knowledge) proof to show the validity of computation and simply ask users to publish $\mathsf{H}_{x_{i,n}}$. The latter approach of ours significantly simplifies the functionality required from the cryptographic tool to accomplish leader election and enables a very efficient instantiation from the most basic primitives. We discuss our approach in more detail in Sec. I-B.

Forward security. Many blockchain systems such as Algorand and Ouroboros Praos [1] employ *forward-secure* digital signatures to maintain the security of prior rounds in case some stored user key is compromised at some point. Existing generic approaches to achieving forward security can be straightforwardly realized in the post-quantum setting by instantiating the underlying ordinary signatures using post-quantum ones.

One such generic approach is known as the MMM approach [16]. In the MMM 'sum' composition, a user creates N key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_N, \mathsf{sk}_N)$ of an ordinary signature and constructs a Merkle tree using the public keys as the tree leaves. To sign a message at a particular time i, the user communicates pk_i and an authentication path in addition to a signature on the actual message.

An alternative generic approach, adopted by Algorand, works as follows. A user first generates a key pair $(\mathsf{mpk}, \mathsf{msk})$ of an ordinary signature, and similarly round key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_N, \mathsf{sk}_N)$. Then, each round public key is signed with msk, denoted as σ'_i, and at a round i, the user communicates $(\sigma'_i, \mathsf{pk}_i)$ in addition to the signature on the actual message. Therefore, the communication difference between the Algorand's approach vs the MMM approach is the cost of a signature (σ'_i) vs an authentication path. The smallest signature length among the schemes selected for standardization by NIST for Post-Quantum Cryptography[4] is at about 700 bytes, which means even an authentication path for a Merkle tree with 2^{20} leaves is cheaper than such a post-quantum signature. Note that the latter Algorand approach also incurs more computation due to signing of round public keys.

Another alternative approach to constructing forward-secure post-quantum signature could be to use a post-quantum Identity-Based Signature (IBS) scheme. In this case, the IBS master secret key is used to derive signing keys for the round keys, where each round is assigned a different ID. The master secret key is deleted and round keys are stored and used to sign messages, and then each one is deleted as the corresponding round period expires. It is well known that such IBS can be constructed from a two-level Hierarchical Identity-Based Encryption (HIBE) scheme. An improved variant of the post-quantum lattice-based LATTE HIBE scheme [17] and its practical implementation are reported in a recent work [18]. However, the user private keys in the latter scheme are already longer than 3 KB (even for a one-level HIBE), and signatures will be even longer. This approach to forward security therefore leads to a longer communication cost compared to an MMM-based solution, which can be instantiated with communication of less than 2.2 KB for each message signed using the Falcon signature scheme [19]. We further note that Algorand's approach described above is an application of a folklore generic IBS construction via certification described, e.g., in [20].

Overall, in the post-quantum setting, the MMM approach already stands out as one of the best options and we adopt it in our work.

A. Our Contributions

Our main contribution in this work is the introduction of a simple, efficient, sustainable and post-quantum solution to blockchain leader election problem. We start by formalizing the concrete requirements of this problem, specifically tailored to the blockchain setting. This leads to notions of (many-time) indexed VRF (iVRF) and authenticated iVRF with forward security ('authenticated MT-iVRF' or 'authenticated iVRF', in short). The former is targeted at the blockchain leader election problem alone, while the latter combines all requirements to accomplish an Algorand-like consensus, where a forward-secure signature is needed. We believe our definitions capture the requirements in a real-life blockchain setting more closely, particularly matching the sequential (i.e., "indexed") nature of blockchain protocols.

Then, we introduce our solutions for (many-time) iVRF and authenticated iVRF that build on a cryptographic hash function, ordinary (t-time) signature (for a parameter t) and pseudorandom generator (PRG), which can be built from a hash function. We prove the security of our instantiations assuming the existence of a secure hash function, digital signature and PRG satisfying natural security requirements in the standard model (without random oracles). Since we do not require a random oracle, there is no complicated quantum random oracle model (QROM) analysis needed to argue security against (full) quantum adversaries.

We implemented our construction in C language on a standard computer using Falcon [19] as the post-quantum signature scheme (see Sec. V for more details). The performance results

[4]https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

together with a comparison with other practical post-quantum VRF proposals are provided in Table I. In the table, we also include ECVRF (used by Algorand) performance results as a reference point even though ECVRF is not quantum-safe. We can see from the table that our iVRF enjoys the smallest sizes across all components compared to other post-quantum schemes. Its evaluation and verification times also outperform all proposals in the table, including ECVRF. The only downside of our approach is its key generation time linear in the total number of allowed evaluations N, but this can be amortized over time and is only a one-time computation per N evaluations that can be straightforwardly parallelized. In Sec. V, we discuss a parameter setting of our iVRF, where the key generation takes about 10 hours on a single core and only needs to be done once a year in the Algorand setting. We now provide an overview of our solution and then discuss its advantages.

B. Overview of Solution

This section discusses an overview of how our final authenticated iVRF construction works in the Algorand setting. As mentioned earlier, our solution relies on having users commit to an ordered set of inputs (i.e., random strings), where the index of each input can be verified. We already know an excellent tool for this purpose: Merkle tree! More generally, we can use any (static) vector commitment.

Let us set N (a power of two) as a parameter that defines the number of rounds before a user key is refreshed. This parameter can be adjusted as desired, but users' (local) key generation time is linear in N due to the Merkle tree construction (see Table I for a concrete key generation time). We envision key refreshments of all users are synchronized in the following sense. The time periods are split into *epochs*, composed of N consecutive rounds, and d rounds before an epoch starts, for a delay parameter d, nodes wanting to get involved in consensus in that epoch are expected to commit to their Merkle tree root on blockchain. If a user performs this commitment at a later time, they still need to wait for d rounds before being able to join the consensus and their Merkle tree still needs to have N leaves[5].

Let us now see how the key generation is done. A user, Alice, generates N pseudorandom values $x_{0,0}, \ldots, x_{N-1,0}$ from a random seed s using a pseudorandom generator (PRG) and computes $x_{i,j} = \mathsf{H}^j(x_{i,0})$ (i.e., j recursive application of H on $x_{i,0}$) for $j = 1, \ldots, t-1$ and $i = 0, \ldots, N-1$, where t is a parameter denoting the number of iterations needed to reach agreement within a round (with high probability). She also generates N key pairs $(\mathsf{pk}_0, \mathsf{sk}_0), \ldots, (\mathsf{pk}_{N-1}, \mathsf{sk}_{N-1})$ of an ordinary (t-time) signature Σ using another random seed s' with the PRG. With these values, Alice now computes $x_{i,t} = \mathsf{H}(x_{i,t-1}, \mathsf{pk}_i)$ and constructs a Merkle tree with $(x_{0,t}, \ldots, x_{N-1,t})$ as the leaves (depicted in Fig. 1). Alice publishes the Merkle tree root on blockchain as her public key[6] (or commitment) and keeps the seeds (s, s') and the intermediate Merkle tree nodes as her secret key.

Now suppose we are at the j-th iteration of the i-th round of consensus in an epoch that Alice is able to participate. Let n denote the actual block round number with a "magic number" Q_n (note that $i \equiv n \bmod N$). To generate an authenticated iVRF output on an input message μ to be authenticated/signed, she outputs $v = \mathsf{H}(x_{i,j}, Q_n)$ as the VRF value, $\sigma = \Sigma.\mathsf{Sign}_{\mathsf{sk}_i}(\mu)$ as the signature and $\pi = (x_{i,j}, \mathsf{pk}_i, \mathsf{AP}_i)$ as the proof where AP_i denotes the Merkle tree authentication path w.r.t. the index i.[7]

Upon receiving an authenticated iVRF output $(v, \sigma, (x, \mathsf{pk}, \mathsf{AP}))$, the verification of iVRF works by first checking if $v = \mathsf{H}(x, Q_n)$ and v is below a threshold. Furthermore, it checks that σ is a valid signature on μ under pk. Finally, for $x' := \mathsf{H}(\mathsf{H}^j(x), \mathsf{pk})$, it checks that the Merkle tree root computed via x' and AP is equal to Alice's public key (or commitment) on blockchain for that epoch.

The intuition behind security is quite simple. Alice generated and committed to $x_{i,j}$'s before seeing Q_n. So, she cannot choose them to bias the output of $\mathsf{H}_{Q_n}(x_{i,n}) := \mathsf{H}(x_{i,j}, Q_n)$. Merkle tree commitment also ensures (computationally) that there is only a single valid $x_{i,j}$ that Alice can use at round n. Hence, assuming the randomness of Q_n, no user has a better advantage of winning the lottery. Of course, as in Algorand, we can adjust the winning condition based on the amount of stake to establish a PoS-based setting or the adjustment can be w.r.t. any other publicly available information.

The forward security of our approach is inherited from the 'sum' composition in the MMM paper [16]. Particularly, as in Algorand, we assume that the adversary cannot corrupt a user within a round (i.e., in 4-5 seconds). To allow such corruptions, we can simply ask users to build a Merkle tree with $N \cdot t$ leaves in the first place and consume t leaves at each round (even if the actual number of iterations in a round is less than t).

As mentioned above, we note that our approach in general can work with any (static) vector commitment, which would replace the role of Merkle tree in our description. Therefore, our approach can benefit from further improvements in the context of vector commitments.

C. Advantages of Our Approach

Simplicity and flexibility. Our approach supports the use of any ordinary signature and any hash function (satisfying natural properties), which are already supported by almost all blockchain applications. These tools have been studied for a long time, and their post-quantum variants are either already standardized or in standardization. Thanks to its simplicity and flexibility, we believe our approach can be easily adapted to work for a range of blockchain systems, for example, Cardano, Dfinity and Rangers Protocol. Note that our approach does not necessarily require the consensus protocol to be based on

[5]If a user joins late, they can actually leave a bottom left sub-tree 'empty' as that part will not be needed. For example, if a user joins 16 rounds late, then they can just pick the left-most tree node at level 4 (from the bottom) and $N - 16$ leaves to construct the Merkle tree. The sub-tree consisting of the first 16 leaves will not be needed.

[6]To prevent Alice from publishing multiple Merkle roots, we can simply have a flag bit in each account that states whether a user has published their Merkle root for that epoch. If that is the case, the verifiers would reject subsequent Merkle root commitments on blockchain.

[7]In the actual protocol, Alice in fact does not need to communicate v since it can be computed from $x_{i,j}$ and the public Q_n.

Proof-of-Stake, and it may be possible, for example, to adapt it to the Proof-of-Storage setting in Filecoin [8].

Confidence in (post-quantum) security. Hash functions (and symmetric primitives in general) are considered to be the most reliable solution to building quantum-safe cryptosystems. Therefore, our leader election solution is built on the safest alternative. For our full solution with authentication, one can choose the best tradeoff between security and efficiency for a specific system thanks to the flexibility of our approach.

Sustainability. Our approach does not have any racing condition and, therefore, does not lead to a tremendous waste of natural resources like Bitcoin.

Efficiency. Combined with one of the best solutions (i.e., the MMM approach) to achieving forward security in the post-quantum setting, the only additional communication cost of our iVRF-based approach is 32 bytes (i.e. the cost of sending the relevant round's $x_{i,j}$), which is a minimal cost one could expect to have. In comparison, combining an MMM-style forward-secure post-quantum signature with X-VRF [13], the smallest post-quantum VRF proposal, (under a common hash tree) would give an additional overhead due to the VRF functionality of about 2100 bytes. This overhead is almost two orders of magnitude more than the 32 byte overhead in our iVRF-based construction.

II. Preliminaries

For a finite set S, we denote by $x \xleftarrow{\$} S$ the sampling of a uniformly random element x of S. For a security parameter λ, we denote by $\mathsf{negl}(\lambda)$ a negligible function in λ, where $\mathsf{negl}(\lambda) := O(\mathsf{poly}(\lambda)/2^{\lambda})$.

A. Hash Functions and Merkle Tree

Definition 1 (Hash-Function Family): A (cryptographic) hash function family with security parameter λ is a set $\mathcal{H}$ of polynomial-time computable functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ that map an arbitrary-length message to an 2λ-bit hash value (digest). We define the following desirable security properties for a cryptographic hash family $\mathcal{H}$:

Everywhere Preimage Resistance (ePre): For this notion, we treat the domain of H as $\{0,1\}^{2\lambda} \times \{0,1\}^{\ell(\lambda)}$. Let $v \in \{0,1\}^{2\lambda}$. We define the following experiment $\mathsf{Exp\text{-}ePre}$ with any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$: Let $(v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{H})$ for $\mathsf{H} \xleftarrow{\$} \mathcal{H}$, $\mu \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$ and $y \leftarrow \mathcal{A}_2(\mu, \mathsf{st})$. $\mathcal{A}$ wins game $\mathsf{Exp\text{-}ePre}$ if $\mathsf{H}(y,\mu) = v$. We say that $\mathcal{H}$ satisfies Everywhere Preimage Resistance (ePre) if for any PPT adversary $\mathcal{A}$,

$$Adv_{ePre}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins } \mathsf{Exp\text{-}ePre}] \leq 2^{-\ell(\lambda)} + \mathsf{negl}(\lambda).$$

Collision Resistance (CR): We say that $\mathcal{H}$ satisfies Collision Resistance if, on input $\mathsf{H} \xleftarrow{\$} \mathcal{H}$, a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ outputs $(m, \hat{m})$ such that $\mathsf{H}(m) = \mathsf{H}(\hat{m})$ but $m \neq \hat{m}$, with at most $\mathsf{negl}(\lambda)$ probability.

Pseudorandom Function (PRF): For this notion, we treat the domain of H as $\{0,1\}^{2\lambda} \times \{0,1\}^*$ (note that the second input can be the empty string). Let $\mathcal{A}$ be a PPT adversary playing the following experiment $\mathsf{Exp\text{-}PRF}$:

1) Let $\mathsf{H} \xleftarrow{\$} \mathcal{H}$.
2) Let $y \xleftarrow{\$} \{0,1\}^{2\lambda}$.
3) Let F be a function chosen uniformly at random from the set of all functions from $\{0,1\}^*$ to $\{0,1\}^{2\lambda}$.
4) Let $b \xleftarrow{\$} \{0,1\}$.
5) $b' \leftarrow \mathcal{A}^{O_b(\cdot)}(\mathsf{H})$.

where $O_b(\cdot)$ is an oracle that on input $\mu \in \{0,1\}^*$ returns $\mathsf{H}(y,\mu) \in \{0,1\}^{2\lambda}$ if $b = 0$ and $F(\mu) \in \{0,1\}^{2\lambda}$ if $b = 1$. The adversary wins the game if $b = b'$.

We say that $\mathcal{H}$ satisfies the Pseudorandom Function (PRF) property if any PPT adversary $\mathcal{A}$ wins $\mathsf{Exp\text{-}PRF}$ with probability at most $\frac{1}{2} + \mathsf{negl}(\lambda)$.

We remark that all the above security properties are natural and standard assumptions on cryptographic hash functions used in practice. The flavour of Everywhere Preimage Resistance above (ePre) is a variant of the ePre one-wayness notion defined in [21], where the input μ above corresponds to the hash key in the ePre definition in [21]. In our variant, the adversary commits to the output value v before getting the hash key μ, whereas in [21] the value v is fixed at the beginning of the game.

We recall the construction of a Merkle Tree hash from a collision resistant hash family $\mathcal{H}$. Given a hash function family $\mathcal{H}$ of hash functions $\mathsf{H} : \{0,1\}^{2\lambda} \times \{0,1\}^{2\lambda} \to \{0,1\}^{2\lambda}$, a Merkle Tree hash MT consists of the following probabilistic polynomial-time (PPT) algorithms ($\mathsf{MT.Setup}$, $\mathsf{MT.Eval}$, $\mathsf{MT.AP}$, $\mathsf{MT.EvalAP}$):

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MT.Setup}(1^{\lambda}, N = 2^{\ell})$**:** Given the security parameter 1^{λ} and the power-of-2 number of Merkle tree leaves $N = 2^{\ell}$, sample a hash function $\mathsf{H} \leftarrow \mathcal{H}$ and output the public parameters $\mathsf{pp} = (\mathsf{H}, N)$.

$(\mathsf{root}, T) \leftarrow \mathsf{MT.Eval}(x_0, \ldots, x_{N-1})$**:** Given as input N Merkle tree leaf values $(x_0, \ldots, x_{N-1})$, compute the Merkle tree root (root) and other tree node values (T) as follows:

- $(x_0^{(\ell)}, \ldots, x_{N-1}^{(\ell)}) \leftarrow (x_0, \ldots, x_{N-1})$
- for $j = \ell - 1$ down to 0 (level index)
- for $i = 0, \ldots, N/2^{\ell-j}$ (node index)
- Let $x_i^{(j)} \leftarrow \mathsf{H}(x_{2i}^{(j+1)}, x_{2i+1}^{(j+1)})$
- end for
- end for
- $\mathsf{root} \leftarrow x_0^{(0)}$
- $T \leftarrow (x_i^{(j)})_{i,j}$

Return (root, T).

For every $j \in \{0, \ldots, \ell-1\}$ and $i \in \{0, N/2^{\ell-j} - 1\}$, the pair $(x_{2i}^{(j+1)}, x_{2i+1}^{(j+1)})$ are called *sibling* node values in the Merkle tree, and the node value $x_i^{(j)} = \mathsf{H}(x_{2i}^{(j+1)}, x_{2i+1}^{(j+1)})$ is called the parent node value of those sibling node values.

$\mathsf{AP}_i \leftarrow \mathsf{MT.AP}(i, T)$**:** Given as input a leaf node index $i \in \{0, \ldots, N-1\}$ and a node value tree $T = (x_i^{(j)})_{i,j}$,

compute the list of ℓ node values on the path P_i from the leaf node value $x_i^{(\ell)}$ to the root node value $x_0^{(0)}$ (not including the root), and return the *authentication path* AP_i consisting of ℓ *sibling* node values of the nodes in P_i.

$\mathsf{root} \leftarrow \mathsf{MT.EvalAP}(i, x_i, \mathsf{AP}_i)$**:** Given as input a leaf node index $i \in \{0, \ldots, N-1\}$, an i'th leaf node value x_i and an authentication path AP_i of ℓ sibling node values along the path P_i from the i'th leaf node to the root node, compute the node values of the nodes in P_i and return the root value root.

Lemma 1 (Merkle Tree Collision Resistance): If $\mathcal{H}$ is a collision-resistant hash family, then the Merkle Tree hash MT built from $\mathcal{H}$ satisfies the following collision-resistance property:

EvalAP Collision Resistance. Let $\mathsf{pp} \leftarrow \mathsf{MT.Setup}(1^\lambda, N)$. On input pp, a polynomial-time adversary $\mathcal{A}$ outputs $(i, x_i, \mathsf{AP}_i, x_i', \mathsf{AP}_i')$ such that $i \in \{0, \ldots, N-1\}$, $x_i' \neq x_i$ but $\mathsf{root}' = \mathsf{root}$ with at most $\mathsf{negl}(\lambda)$ probability, where $\mathsf{root}' \leftarrow \mathsf{MT.EvalAP}(i, x_i', \mathsf{AP}_i')$ and $\mathsf{root} \leftarrow \mathsf{MT.EvalAP}(i, x_i, \mathsf{AP}_i)$.

B. Pseudorandom Generator

We start with the following definition.

Definition 2: A (stateful) pseudorandom number generator G with security parameter λ is a pair of algorithms $(\mathsf{G.Key}, \mathsf{G.Next})$ and an integer Q, where G.Key is a probabilistic algorithm which takes no input and outputs an initial state $s \in \{0,1\}^\ell$, G.Next is a deterministic algorithm which, given the current state s, outputs a pair $(s', r) \leftarrow \mathsf{G.Next}(s)$, where s' is the new state and $r \in \{0,1\}^\lambda$ is the output and Q is the maximal number of outputs the pseudorandom number generator is allowed to produce.

The following security properties are desirable for a stateful pseudorandom generator G:

Pseudorandomness of G**:** A challenger first lets $s \xleftarrow{\$} \{0,1\}^\lambda$, $b \xleftarrow{\$} \{0,1\}$. All successive quires to $\mathcal{O}^{\mathsf{G.Next}}(s)$ should be indistinguishable from random, where $\mathcal{O}^{\mathsf{G.Next}}(\cdot)$ uses either $(s', r_0) \leftarrow \mathsf{G.Next}(s)$ or a random string $r_1 \xleftarrow{\$} \{0,1\}^\lambda$ to respond r_b to $\mathcal{A}$. The adversary $\mathcal{A}$ returns b' and wins the game if $b' = b$.

Forward security of G**:** The challenger generates a random initial secret input s and challenges the adversary $\mathcal{A}$ on its capacity to distinguish the real output of the pseudorandom number generator from random. In addition to the usual procedures detailed in the above pseudorandomness game, the adversary $\mathcal{A}$ has access to an oracle $\mathcal{O}^{\mathsf{GetState}}$ in which $\mathcal{A}$ has access to the current value of the state s. A G is called (T, Q, ε)-forward-secure, if for any adversary $\mathcal{A}$ running in time at most T, making at most Q calls to $\mathcal{O}^{\mathsf{G.Next}}(\cdot)$, followed by one call to an oracle, which is the last call $\mathcal{A}$ is allowed to make and gets the last state s, the advantage of $\mathcal{A}$ in this game is at most ε.

C. Digital Signatures and Forward Security

Definition 3 (Digital Signature): A signature scheme Σ consists of three probabilistic polynomial-time (PPT) algorithms $(\Sigma.\mathsf{Keygen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Verify})$ satisfying the following:

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Keygen}(1^\lambda)$**:** Given the security parameter 1^λ, output a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$.

$(\sigma, \mu) \leftarrow \Sigma.\mathsf{Sign}_{\mathsf{sk}}(\mu)$**:** Given as input a secret key sk and a message μ, output a signature-message pair (σ, μ).

$0/1 \leftarrow \Sigma.\mathsf{Verify}_{\mathsf{pk}}(\sigma, \mu)$**:** Given as input a public key pk, a signature σ and a message μ it outputs a bit $b = 1$ if the signature is valid and $b = 0$ otherwise.

It is required that for every λ, every key pair $(\mathsf{pk}, \mathsf{sk})$ output by the key generation algorithm, and every message $\mu \in \{0,1\}^*$ it holds that $\Sigma.\mathsf{Verify}_{\mathsf{pk}}(\Sigma.\mathsf{Sign}(\mathsf{sk}, \mu), \mu) = 1$.

a) Unforgeability against Chosen-Message Attacks (UF-CMA): A signature scheme $\Sigma = (\Sigma.\mathsf{Keygen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Verify})$ is unforgeable against chosen-message attacks if for all probabilistic polynomial-time adversaries, the probability to win the following experiment Exp-UF-CMA is $\mathsf{negl}(\lambda)$:

1) $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Keygen}(1^\lambda)$
2) $(\sigma, \mu) \leftarrow \mathcal{A}^{\mathcal{O}\Sigma.\mathsf{Sign}(sk, \cdot)}(\mathsf{pk})$
3) If $1 \leftarrow \Sigma.\mathsf{Verify}(\sigma, \mu, \mathsf{pk})$ and μ has not been queried to $\mathcal{O}\Sigma.\mathsf{Sign}(\mathsf{sk}, \cdot)$ return 1, else return 0.

We write $\Sigma.\mathsf{Keygen}(1^\lambda; r)$ to denote that the key generation function is run using a randomness r.

In [16], the authors showed a way of constructing a forward-secure signature from any (ordinary) signature using a 'sum' composition. Let N be the total time period for the forward-secure signature and assume for simplicity that N is a power of two. We recall below how this 'MMM approach' works for a given (one-time) signature scheme Σ.

MMM forward-secure signature approach (sum composition).

$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}_0) \leftarrow \mathsf{FSS.Keygen}(1^\lambda, N)$:
1) Set up $s \leftarrow \mathsf{G.Key}(1^\lambda)$
2) Compute $(s_i, r_i) \leftarrow \mathsf{G.Next}(s_{i-1})$ for $i = 0, \ldots, N-1$ where $s_{-1} := s$
3) Compute $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma.\mathsf{Keygen}(1^\lambda; r_i)$ for $i = 0, \ldots, N-1$.
4) Create a Merkle tree using $(\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1})$ as the leaves and denote its root by root.
5) Return $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}_0) = (\mathsf{root}, (s, 0))$.

$\widetilde{\mathsf{sk}}_i \leftarrow \mathsf{FSS.Update}(\widetilde{\mathsf{sk}}_{i-1})$: Parse $\widetilde{\mathsf{sk}}_{i-1} = (s_{i-1}, r_{i-1})$ and return $\mathsf{G.Next}(s_{i-1})$.

$\hat{\sigma} \leftarrow \mathsf{FSS.Sign}_{\widetilde{\mathsf{sk}}_i}(\mu, i)$: Parse $\widetilde{\mathsf{sk}}_i = (s_i, r_i)$ and derive $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Keygen}(1^\lambda; r_i)$. Compute $\sigma \leftarrow \Sigma.\mathsf{Sign}_{\mathsf{sk}}(\mu)$. Return $\hat{\sigma} = (\sigma, \mathsf{pk}, \mathsf{AP})$ where AP is the authentication path for index i.

$0/1 \leftarrow \mathsf{FSS.Verify}_{\widetilde{\mathsf{pk}}}(\hat{\sigma}, \mu, i)$:
1) Parse $\hat{\sigma} = (\sigma, \mathsf{pk}, \mathsf{AP})$.
2) If $\Sigma.\mathsf{Verify}_{\mathsf{pk}}(\sigma, \mu) = 0$, return 0.
3) Using pk and AP, compute a Merkle root root' w.r.t. the index i.
4) If $\mathsf{root}' = \widetilde{\mathsf{pk}}$, return 1. Otherwise, return 0.

In [16], the authors prove that the above construction is a proper forward-secure signature scheme. For a digital signature

scheme to be forward-secure, it should be computationally infeasible to forge a signature for a new message in a time period earlier than a time period, when a secret key was given/leaked to the adversary.

III. Formal Definitions of Many-Time Indexed VRF (MT-iVRF)

In this section, we discuss the formal definitions of a Many-Time Indexed VRF (MT-iVRF) and authenticated MT-iVRF with forward security. Particularly, we discuss the definitions of an authenticated MT-iVRF with forward security, and an MT-iVRF is simply a special case of that. For simplicity, we refer to our constructions as MT-iVRF and authenticated MT-iVRF (omitting 'with forward security' or even 'MT-' sometimes).

Our definitions are adapted from those of an ordinary VRF [6] and a forward-secure (or key-evolving) signature [22]. We also adapt the unbiasability definition of [11] into our setting. An important feature of our definitions is that we capture more closely the real-life setting in blockchain, where the protocol is inherently stateful and indexed (by round/iteration numbers). Our modifications over the prior definitions allow us to construct the desired tool efficiently from simple cryptographic primitives.

A. Syntax

Let N be the parameter denoting the maximum number of time periods (i.e., 'rounds') allowed (i.e., the iVRF is N-time), and t be the parameter denoting the maximum number of 'iterations' allowed within any given time period/round. In particular, we are here assuming a more generalized setting where each time period is split into further 'iterations', where the forward security will be required w.r.t. to the time periods (not iterations). Note that fixing $t = 1$ leads to the standard forward security setting in [22]. In our definitions of the authenticated MT-iVRF, we let $\mathsf{iAV.Eval}$ take two input messages μ_1, μ_2 to allow verifiable VRF evaluation on one message (μ_1) and authentication of another message (μ_2). The above two generalizations to the formal definitions are done to properly capture an Algorand-like blockchain setting.

In the functions below, we do not index the secret key in order not to clutter the presentation and the key update function $\mathsf{iAV.KeyUpd}$ simply periodically updates the secret key. The time period index i is clear from the descriptions of $\mathsf{iAV.Eval}$ and $\mathsf{iAV.Verify}$ functions as it is part of the input. An authenticated many-time indexed VRF, iAV, is given by the following five algorithms ($\mathsf{iAV.ParamGen}$, $\mathsf{iAV.Keygen}$, $\mathsf{iAV.KeyUpd}$, $\mathsf{iAV.Eval}$, $\mathsf{iAV.Verify}$):

- $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$: Given the security parameter λ, set up and return public parameters pp containing N. We assume that pp is an implicit input to the other algorithms.
- $(\mathsf{pk_{av}}, \mathsf{sk_{av}}) \leftarrow \mathsf{iAV.Keygen}(\mathsf{pp})$: Given the public parameters pp, return a public-secret key pair $(\mathsf{pk_{av}}, \mathsf{sk_{av}})$.
- $\mathsf{sk_{av}} \leftarrow \mathsf{iAV.KeyUpd}(\mathsf{sk_{av}})$: Given the given secret key $\mathsf{sk_{av}}$ for the previous period, update (overwrite) $\mathsf{sk_{av}}$ for the current time period.
- $(v, \sigma, \pi) \leftarrow \mathsf{iAV.Eval}_{\mathsf{sk_{av}}}(\mu_1, \mu_2, (i, j))$: Given input messages $\mu_1, \mu_2 \in \{0,1\}^{\ell(\lambda)}$ and a pair of indices (i, j) with $0 \leq i < N$ and $0 \leq j < t$, return a VRF value $v \in \{0,1\}^{m(\lambda)}$ w.r.t. μ_1, a signature σ w.r.t. μ_2 and an accompanying proof π.
- $0/1 \leftarrow \mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \mu_2, (i, j), v, \sigma, \pi)$: Given input messages μ_1, μ_2, a pair of indices (i, j) with $0 \leq i < N$ and $0 \leq j < t$, a VRF value v, a signature σ and a purported proof π, check using π if (v, σ) is correctly generated for the given $(\mu_1, \mu_2, (i, j))$ and $\mathsf{pk_{av}}$.

The (non-authenticated) MT-iVRF construction (in Sec. IV-A) does not have a signature σ and a second message μ_2. Hence, it only serves the VRF functionality without any authentication. The purpose of the parameters t and j is to formally match our application setting to Algorand. For applications not requiring them, one may simply fix $(t, j) = (1, 0)$.

B. Correctness and Security Definitions

Compared to an ordinary VRF, there are two main distinctions of our security definitions. First, uniqueness holds in the case of any (arbitrarily-generated) fixed input-index pair $(\mu_1, (i, j))$, rather than just any (arbitrarily-generated) fixed input μ_1. Secondly, pseudorandomness is satisfied against any challenge input-index pair (μ_1, i), where the index i is never queried to the $\mathsf{iAV.Eval}$ oracle $\mathcal{O}_{\mathsf{iAV.Eval}}$. In the case of ordinary VRFs, the adversary is not allowed to query the $\mathcal{O}_{\mathsf{iAV.Eval}}$ oracle on the challenge *input* μ_1.

Provability. For any $(v, \sigma, \pi) \leftarrow \mathsf{iAV.Eval}_{\mathsf{sk_{av}}}(\mu_1, \mu_2, (i, j))$ with $(\mathsf{pk_{av}}, \mathsf{sk_{av}}) \leftarrow \mathsf{iAV.Keygen}(\mathsf{pp})$, $\mathsf{sk_{av}} \leftarrow \mathsf{iAV.KeyUpd}^i(\mathsf{sk_{av}})$ and $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$, the algorithm $\mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \mu_2, (i, j), v, \sigma, \pi)$ outputs 1.

Computational Full Uniqueness (CFU). Let $\mathcal{A}$ be a polynomial-time adversary playing the following experiment Exp-CFU:

1) $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$.
2) $(\mu_1, i, j, \mathsf{pk_{av}}, \mu_2, v, \sigma, \pi, \hat{\mu}_2, \hat{v}, \hat{\sigma}, \hat{\pi}) \leftarrow \mathcal{A}(\mathsf{pp})$.

The adversary $\mathcal{A}$ wins the game if $\mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \mu_2, (i, j), v, \sigma, \pi) = \mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \hat{\mu}_2, (i, j), \hat{v}, \hat{\sigma}, \hat{\pi}) = 1$ and $v \neq \hat{v}$, with $0 \leq i < N$ and $0 \leq j < t$. An (N-time) authenticated MT-iVRF with forward security is said to satisfy *computational full uniqueness*, if the adversary $\mathcal{A}$ wins the above game with at most $\mathsf{negl}(\lambda)$ probability.

Pseudorandomness. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time adversary playing the following experiment Exp-PRand:

1) $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$,
2) $(\mathsf{pk_{av}}, \mathsf{sk_{av}}) \leftarrow \mathsf{iAV.Keygen}(\mathsf{pp})$,
3) $(\mu_1^*, \mu_2^*, i^*, j^*, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{iAV.Eval}}(\cdot)}(\mathsf{pk_{av}})$,
4) $\mathsf{sk_{av}} \leftarrow \mathsf{iAV.KeyUpd}^{i^*}(\mathsf{sk_{av}})$,
5) $(v_0, \sigma_0, \pi_0) \leftarrow \mathsf{iAV.Eval}_{\mathsf{sk_{av}}}(\mu_1^*, \mu_2^*, (i^*, j^*))$,
6) $v_1 \xleftarrow{\$} \{0,1\}^{m(\lambda)}$,
7) $b \xleftarrow{\$} \{0,1\}$,
8) $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{iAV.Eval}}(\cdot)}(v_b, \mathsf{st})$,

where $\mathcal{O}_{\mathsf{iAV.Eval}}(\cdot)$ is an oracle that on input a message-index tuple $(\mu_1, \mu_2, (i, j))$ outputs a VRF value v, a signature σ and a corresponding proof of correctness π. The adversary is restricted querying $\mathcal{O}_{\mathsf{iAV.Eval}}(\cdot)$ on index pairs (i, j) for j sequentially incrementing (i.e., $\mathcal{A}$ must query (i, j) first before being able to query $(i, j+1)$). Let $\mathcal{I}$ be the set of all index pairs queried by the adversary. The adversary wins the game if $b = b'$

and $(i^*, j^*) \notin \mathcal{I}$.

We say that an authenticated MT-iVRF with forward security is *pseudorandom* if any PPT adversary $\mathcal{A}$ wins Exp-PRand with probability at most $\frac{1}{2} + \mathsf{negl}(\lambda)$.

Forward-Secure Unforgeability. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time adversary playing the following experiment Exp-Forge:

1) $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$,
2) $(\mathsf{pk_{av}}, \mathsf{sk_{av}}) \leftarrow \mathsf{iAV.Keygen}(\mathsf{pp})$,
3) Set $i = 0$,
4) Until $\mathcal{A}$ stops or i reaches $N-1$: increment i by 1; set $\mathsf{sk_{av}} \leftarrow \mathsf{iAV.KeyUpd}(\mathsf{sk_{av}})$ and $\mathsf{st}_i \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{iAV.Eval}}(i;\cdot)}(\mathsf{pp}, \mathsf{pk_{av}})$,
5) $(\mu_1, \mu_2, k, j, v, \sigma, \pi) \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{pk_{av}}, \mathsf{sk_{av}}, \mathsf{st}_1, \ldots, \mathsf{st}_i)$,
6) $b \leftarrow \mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \mu_2, (k, j), v, \sigma, \pi)$,

where $\mathcal{O}_{\mathsf{iAV.Eval}}(i;\cdot)$ is an oracle that returns an iAV.Eval output w.r.t. the i-th secret key $\mathsf{sk_{av}}$ and time period i. $\mathcal{A}$ wins if $b = 1$, $1 \le k < i$, $0 \le j < t$ and μ_2 was not queried to $\mathcal{O}_{\mathsf{iAV.Eval}}$. We say that an indexed VRF is *forward-secure unforgeable* if

$$\Pr[\mathcal{A} \text{ wins Exp-Forge}] \le \mathsf{negl}(\lambda).$$

Unbiasability. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time adversary playing the following experiment Exp-Bias:

1) $\mathsf{pp} \leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$,
2) $(\mathsf{pk_{av}}, \mu_2, v, i, j, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp})$,
3) $\mu_1 \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$,
4) $(\sigma, \pi) \leftarrow \mathcal{A}_2(\mu_1, \mathsf{st})$,
5) $b \leftarrow \mathsf{iAV.Verify}_{\mathsf{pk_{av}}}(\mu_1, \mu_2, (i, j), v, \sigma, \pi)$.

$\mathcal{A}$ wins if $b = 1$. We say that an authenticated MT-iVRF with forward security is *unbiasable* if

$$\Pr[\mathcal{A} \text{ wins Exp-Bias}] \le 2^{-\ell(\lambda)} + \mathsf{negl}(\lambda).$$

IV. Our Constructions

We first introduce our many-time indexed VRF (MT-iVRF) which is constructed from a secure Merkle hash tree and which can evaluate up to N time periods (i.e., rounds). This scheme allows multiple iterations within a single round, matching the Algorand setting. After that, we present our final MT-iVRF scheme which also provides authentication and forward security.

A. Many-Time indexed VRF (MT-iVRF)

As discussed, the index has two parts as (i, j), where $i = 0, \ldots, N-1$ and $j = 0, \ldots, t$ for some public parameters $t, N \ge 1$. This construction is the setting depicted in Fig. 1 with no pk_i's. We use a cryptographic hash family $\mathcal{H}$ and a PRG $\mathsf{G} : \{0,1\}^\lambda \to (\{0,1\}^\lambda)^N$.

$\mathsf{iVRF.ParamGen}(1^\lambda)$: Pick a hash function $\mathsf{H} \xleftarrow{\$} \mathcal{H}$. Set parameters t and N for a power-of-2 N. Return $\mathsf{pp} = (\mathsf{H}, \mathsf{G}, N, t)$.

$\mathsf{iVRF.Keygen}(\mathsf{pp})$:

1) Set up $s \leftarrow \mathsf{G.Key}(1^\lambda)$.
2) Derive pseudorandom values $(x_{0,0}, \ldots, x_{N-1,0})$ by running G.Next iteratively on s.
3) Compute $x_{i,j+1} = \mathsf{H}(x_{i,j})$ for $i = 0, \ldots, N-1$ and $j = 0, \ldots, t-1$.
4) Construct a Merkle tree using $(x_{0,t}, \ldots, x_{N-1,t})$. Let root be the root of the tree.

Return $(\mathsf{pk_v}, \mathsf{sk_v}) = (\mathsf{root}, s)$.

$\mathsf{iVRF.Eval}_{\mathsf{sk_v}}(\mu, (i, j))$:

1) Derive $x_{i,0}$ from $s = \mathsf{sk_v}$ and update the state of the PRG G.
2) Compute $y = \mathsf{H}^{t-1-j}(x_{i,0})$.
3) Compute $v = \mathsf{H}(y, \mu)$.
4) Compute an authentication path AP_i w.r.t. the leaf index i.

Return v as the VRF value along with a proof $\pi = (y, \mathsf{AP}_i)$.

$\mathsf{iVRF.Verify}_{\mathsf{pk_v}}(\mu, (i, j), v, \pi)$:

1) Parse $\pi = (y, \mathsf{AP})$.
2) If $v \ne \mathsf{H}(y, \mu)$, return 0.
3) Compute $x_{i,t} = \mathsf{H}^{j+1}(y)$.
4) Compute a Merkle root, root′, using $x_{i,t}$ and AP w.r.t. the leaf index i.

If $\mathsf{root}' = \mathsf{pk_v}$, return 1. Otherwise, return 0.

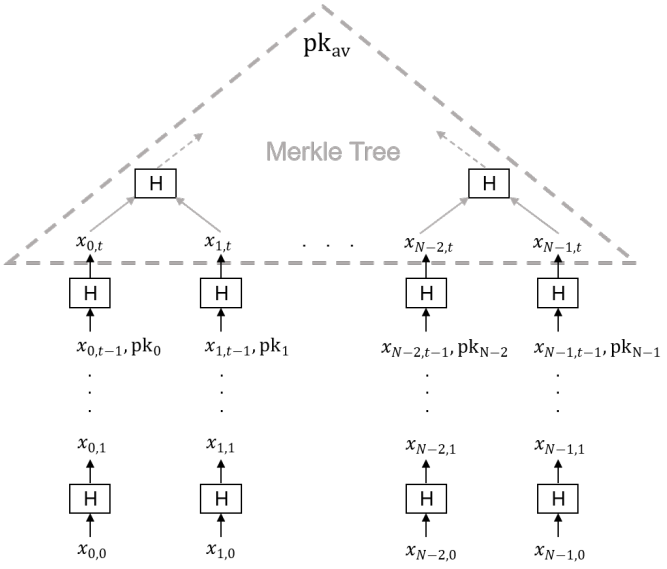

Fig. 1. Overall structure of our Authenticated MT-iVRF with Forward Security. The term $x_{i,0}$'s are pseudorandom strings generated from a seed. The pk_i's are (independent) public keys of an ordinary (t-time) signature.

B. Authenticated MT-iVRF with Forward Security

In the MMM (recursive) sum composition approach [16], described in Sec. II-C, to construct a forward-secure (FS) signature for N time periods, one makes use of a Merkle tree with N leaves, where each leaf corresponds to a random (one-time) signature key pair with the public keys used to construct the Merkle tree. Therefore, our MT-iVRF instantiation can be naturally combined with the MMM approach. The advantage in this case is that we will have a *single* Merkle tree (and a single tool) to realize both the VRF functionality as well as the forward-secure signature. This means that it is sufficient to communicate a *single* authentication path to authenticate both the keys for the FS signature and the evaluator's committed

values y used to compute the VRF value $v = \mathsf{H}(y, \mu)$ (where μ is the VRF input/message). As a result, the additional communication overhead of our final authenticated MT-iVRF construction over the MMM approach is minimal at just 32 bytes (recall that $v = \mathsf{H}(y, \mu)$ can be computed in the Algorand application, so need not be communicated).

Concretely, when constructing the Merkle tree, we set $x_{i,t} = \mathsf{H}(x_{i,t-1}, \mathsf{pk}_i)$ (instead of $x_{i,t} = \mathsf{H}(x_{i,t-1})$) in key generation, where $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}$ are independent public keys of a (t-time) signature scheme. As with the previous construction, we use a cryptographic hash family $\mathcal{H}$ of functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and a PRG $\mathsf{G} : \{0,1\}^\lambda \to (\{0,1\}^\lambda)^N$.

$\mathsf{iAV.ParamGen}(1^\lambda)$: Pick hash function $\mathsf{H} \xleftarrow{\$} \mathcal{H}$. Set parameters $t = \mathsf{poly}(\lambda)$ and $N = \mathsf{poly}(\lambda)$ for a power-of-2 N. Return $\mathsf{pp} = (\mathsf{H}, \mathsf{G}, N, t)$.

$\mathsf{iAV.Keygen}(\mathsf{pp})$:

1) Set up seeds $s, s' \leftarrow \mathsf{G.Key}(1^\lambda)$.
2) Derive pseudorandom values $(x_{0,0}, \ldots, x_{N-1,0})$ by running G.Next iteratively on s.
3) Compute $x_{i,j+1} = \mathsf{H}(x_{i,j})$ for $i = 0, \ldots, N-1$ and $j = 0, \ldots, t-2$.
4) Derive pseudorandom values $(r_0, \ldots, r_{N-1})$ by running G.Next iteratively on s'.
5) $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma.\mathsf{Keygen}(\mathsf{pp}; r_i)$ for $i = 0, \ldots, N-1$.
6) Compute $x_{i,t} = \mathsf{H}(x_{i,t-1}, \mathsf{pk}_i)$ for $i = 0, \ldots, N-1$.
7) Construct a Merkle tree using $(x_{0,t}, \ldots, x_{N-1,t})$. Let root be the root of the tree.

Return $(\mathsf{pk}_{\mathsf{av}}, \mathsf{sk}_{\mathsf{av}}) = (\mathsf{root}, (s, 0, s', 0))$.

$\mathsf{iAV.KeyUpd}(\mathsf{sk}_{\mathsf{av}})$: Parse $\mathsf{sk}_{\mathsf{av}} = (s, x, s', r)$ and update $\mathsf{sk}_{\mathsf{av}}$ as $(\mathsf{G.Next}(s), \mathsf{G.Next}(s'))$.

$\mathsf{iAV.Eval}_{\mathsf{sk}_{\mathsf{av}}}(\mu_1, \mu_2, (i, j))$:

1) Parse $\mathsf{sk}_{\mathsf{av}} = (s_i, x_{i,0}, s'_i, r_i)$.
2) Compute $y = \mathsf{H}^{t-1-j}(x_{i,0})$
3) Compute $v = \mathsf{H}(y, \mu_1)$.
4) Compute $\mathsf{pk} \leftarrow \Sigma.\mathsf{Keygen}(\mathsf{pp}; r_i)$
5) Compute $\sigma \leftarrow \Sigma.\mathsf{Sign}_{\mathsf{sk}}(\mu_2)$
6) Compute an authentication path AP_i w.r.t. the leaf index i.

Return v as the VRF value, σ as the signature and $\pi = (y, \mathsf{pk}, \mathsf{AP}_i)$ as the accompanying proof.

$\mathsf{iAV.Verify}_{\mathsf{pk}_{\mathsf{av}}}(\mu_1, \mu_2, (i, j), v, \sigma, \pi)$:

1) Parse $\pi = (y, \mathsf{pk}, \mathsf{AP})$.
2) If $v \neq \mathsf{H}(y, \mu_1)$, return 0.
3) If $\Sigma.\mathsf{Verify}_{\mathsf{pk}}(\sigma, \mu_2) = 0$, return 0.
4) Compute $y' = \mathsf{H}^j(y)$
5) Compute $x_i = \mathsf{H}(y', \mathsf{pk})$
6) Compute a Merkle root, root′, using x_i and AP w.r.t. the leaf index i.

If $\mathsf{root}' = \mathsf{pk}_{\mathsf{av}}$, return 1. Otherwise, return 0.

V. Performance Analysis and Implementation

Thanks to the simplicity of our scheme, the performance analysis can straightforwardly be done by counting the number of operations. For the hash operations, we look at the amount of bytes to be hashed rather than how many $\mathsf{H}(\cdot)$ operations appear in the description to capture a more accurate result, and consider 32-byte values for $x_{i,j}$'s, μ_i's and the hash output. We write $|\sigma|_B$ and $|\mathsf{pk}|_B$ to denote the byte-lengths of the ordinary signature and its public key.

iAV.Keygen consists of the following main operations:

1) $2N$ PRG iterations,
2) less than $(t+2) \cdot N \cdot 32 + N \cdot |\mathsf{pk}|_B$ bytes of hash calculations, and
3) N (ordinary) signature key generations.

Therefore, iAV.Keygen runtime is likely to be dominated by N (ordinary) signature key generations (for reasonable values of t), which is consistent with our implementation results discussed further below in this section. Note that key generation of Algorand's approach to forward security (as discussed in the introduction) involves invoking ordinary signature key generation *and* signing algorithms N times each. Hence, our iAV.Keygen runtime is likely to closely match (or may even be faster than) Algorand's approach. Note also that in the context of Algorand, keys for the next epoch can be generated progressively in the current epoch so that the key generation runtime is amortized.

Next, iAV.Eval consists of the following main operations:

1) at most $(t+1) \cdot 32$ bytes of hash calculations,
2) 1 (ordinary) signature key generation, and
3) 1 (ordinary) signing.

Note that the authentication path in iAV.Eval can simply be retrieved when the Merkle tree is stored.

Finally, iAV.Verify consists of the following main operations:

1) at most $(2\log N + t + 2) \cdot 32 + |\mathsf{pk}|_B$ bytes of hash calculations, and
2) 1 (ordinary) signature verification.

In terms of communication, iAV.Eval requires transmission of an ordinary signature σ, an ordinary signature public key pk, an authentication path ($\log N$ 32-byte strings) and a 32-byte string (and another 32-byte string if v needs to be communicated). So, in total, $|\sigma|_B + |\mathsf{pk}|_B + (\log N + 1) \cdot 32$ bytes of communication is needed.

The storage requirement by the evaluator is heavily dominated by the storage of the Merkle tree nodes (if they are stored). If the whole tree is stored, then about $32 \cdot 2N = 64N$ bytes of storage is needed. However, standard optimizations such as partial tree storage (together with progressive computation of missing nodes and deletion of used nodes) can be adopted. For example, by computing the two sibling leaf nodes together every two time periods, we can avoid storage of leaf nodes and reduce the storage requirement to $32N$ bytes.

We implemented our authenticated MT-iVRF with forward security (from Sec. IV-B) in C language (on a single Intel i7-7700K core at 4.2GHz).[8] We used Falcon-512 [19] to instantiate the post-quantum signature, SHA-256 for the hash function H, and AES-256 with the CTR-DRBG mode [23] (implemented by using the AES-NI hardware instructions [24]) for the PRG G. Falcon-512 has a public key of 897 bytes and a signature of 666 bytes. We used the Falcon variant with the ChaCha20 [25] seed expander provided by the latest Falcon

[8]The source code is available at https://gitlab.com/raykzhao/ivrf.

TABLE II. PERFORMANCE OF OUR AUTHENTICATED MT-IVRF WITH FORWARD SECURITY IN SEC. IV-B FOR DIFFERENT PARAMETERS.

(N,t)	Proof Size	Keygen	Eval	Verify	Key Lifetime in Algorand
$(2^{18},16)$	608 bytes	19.44 mins	4.63 ms	0.046 ms	2 weeks
$(2^{18},100)$	608 bytes	19.82 mins	4.72 ms	0.088 ms	2 weeks
$(2^{23},16)$	768 bytes	10.40 hours	4.63 ms	0.049 ms	>1 year
$(2^{23},100)$	768 bytes	10.45 hours	4.67 ms	0.092 ms	>1 year

TABLE III. RUNTIMES OF OUR EXECUTION OF FALCON SIGNATURE.

Keygen Time	Signing Time	Verify Time
4.45 ms	0.18 ms	0.023 ms

reference implementation[9], with both AVX2 and FMA instructions enabled. Clang 14.0.6 compiler was used to compile the code, with optimisation flags `-O3 -march=native`. We disabled Hyper-threading and Turbo Boost during the benchmarks.

In Tables II and III, we summarize the concrete performance results of our authenticated MT-iVRF with forward security proposal and our execution of Falcon-512 signature, respectively. When computing the proof sizes in Table II, we remove the costs due to the signature scheme to clearly show the impact of varying parameters since the cost due to the signature is fixed and arises from authentication, not the VRF functionality. We also note that when running $\mathsf{iAV.Eval}$ and $\mathsf{iAV.Verify}$, we set $j=0$ and $j=t-1$, respectively, to capture the *worst-case* running times.

Let us first analyze the impact of the parameter t. It is easy to see that the proof size is independent of t (as also evident from the theoretical analysis above) and that $\mathsf{iAV.Keygen}$ runtime increases very little even when t is increased to a large value like 100. The parameter t has also little impact on the $\mathsf{iAV.Eval}$ runtime, and $\mathsf{iAV.Verify}$ is still very fast even in the worst case with $t=100$. Therefore, we believe there is a lot of freedom for the choice of t.

Using the results of Tables II and III, it is easy to derive that the vast majority of $\mathsf{iAV.Keygen}$ time is spent on Falcon key generations. Similarly, the vast majority of $\mathsf{iAV.Eval}$ time is spent of Falcon key generation and signing. In fact, the slight variations in $\mathsf{iAV.Eval}$ for different N values are mainly due to the variations in the Falcon signature runtimes as $\mathsf{iAV.Eval}$ runtime is (almost) independent of N (neglecting the minor cost of retrieving $\log N$ tree nodes).

The runtimes of our execution of Falcon-512 (see Table III) are close to the reported runtimes on the official Falcon website[10]. Note that some of our runtimes in Table III are faster, due to the ChaCha20 seed expander (up to 8% faster for Keygen on Intel CPUs, as reported by the latest Falcon reference implementation) and higher CPU frequency in our benchmark platform (4.2GHz compared to 2.3GHz).

VI. SECURITY ANALYSIS

Provability of our authenticated MT-iVRF (with forward security) construction follows via straightforward investigation.

[9]https://falcon-sign.info/Falcon-impl-20211101.zip

[10]https://falcon-sign.info/

Theorem 1 (Computational Full Uniqueness): The authenticated MT-iVRF construction of Subsection IV-B satisfies Computational Full Uniqueness, if the hash function family $\mathcal{H}$ is Collision Resistant.

Proof: Let $\mathbf{Game}$ denote the Exp-CFU game running with a PPT adversary $\mathcal{A}$ on input $\mathsf{pp}=(\mathsf{H},\mathsf{G},N,t)\leftarrow \mathsf{iAV.ParamGen}(1^\lambda)$. We denote the adversary run time by $T_{\mathcal{A}}$. The adversary returns $(\mu_1,i,j,\mathsf{pk_{av}},\mu_2,v,\sigma,\pi=(y,\mathsf{pk},\mathsf{AP}),\hat{\mu}_2,\hat{v},\hat{\sigma},\hat{\pi}=(\hat{y},\hat{\mathsf{pk}},\hat{\mathsf{AP}}))$. Let W denote the event that $\mathcal{A}$ wins in $\mathbf{Game}$, i.e. we have (1) $v\neq\hat{v}$, where $v:=\mathsf{H}(y,\mu_1)$, $\hat{v}:=\mathsf{H}(\hat{y},\mu_1)$, and (2) $\mathsf{root}=\hat{\mathsf{root}}=\mathsf{pk_{av}}$, where $\mathsf{root}:=\mathsf{MT.EvalAP}(i,x_i,\mathsf{AP})$, $x_i:=\mathsf{H}(y',\mathsf{pk})$, $y':=\mathsf{H}^j(y)$ and $\hat{\mathsf{root}}:=\mathsf{MT.EvalAP}(i,\hat{x}_i,\hat{\mathsf{AP}})$, $\hat{x}_i:=\mathsf{H}(\hat{y}',\hat{\mathsf{pk}})$, $\hat{y}':=\mathsf{H}^j(\hat{y})$. If W happens, then from (1), we have $y\neq\hat{y}$.

We define three subevents that partition the event W. The subevent W_1 occurs if W occurs and $y'=\hat{y}'$. The subevent W_2 occurs if W occurs, W_1 does not occur (so $y'\neq\hat{y}'$) and $x_i=\hat{x}_i$. Finally, the subevent W_3 occurs if W occurs and W_1 and $W_{0,2}$ do not occur (so $x_i\neq\hat{x}_i$).

Subevent W_1 implies that $\mathsf{H}^j(y)=\mathsf{H}^j(\hat{y})$ whereas $y\neq\hat{y}$. Let $j'\leq j$ denote the smallest positive integer such that $\mathsf{H}^{j'}(y)=\mathsf{H}^{j'}(\hat{y})$. Then $(\mathsf{H}^{j'-1}(y),\mathsf{H}^{j'-1}(\hat{y}))$ is a collision for H. It follows that there exists an adversary breaking the collision-resistance of $\mathcal{H}$ with run-time at most $T_{\mathcal{A}}$ and success probability $\geq\Pr[W_1]$, so $\Pr[W_1]\leq\varepsilon_{\mathcal{H},CR}=\mathsf{negl}(\lambda)$, where the latter equality is from the assumed collision-resistance of $\mathcal{H}$, where $\varepsilon_{\mathcal{H},CR}$ is the maximal advantage of an adversary with run time $T_{\mathcal{A}}$ against collision-resistance of $\mathcal{H}$.

Similarly, subevent W_2 implies that $\mathsf{H}(y',\mathsf{pk})=\mathsf{H}(\hat{y}',\hat{\mathsf{pk}})$ whereas $y'\neq\hat{y}'$. Then $((y',\mathsf{pk}),(\hat{y}',\hat{\mathsf{pk}}))$ is a collision for H, so $\Pr[W_2]\leq\varepsilon_{\mathcal{H},CR}=\mathsf{negl}(\lambda)$.

Finally, subevent W_3 implies that $\mathsf{MT.EvalAP}(i,x_i,\mathsf{AP})=\mathsf{MT.EvalAP}(i,\hat{x}_i,\hat{\mathsf{AP}})$ whereas $x_i\neq\hat{x}_i$. Then $(i,x_i,\mathsf{AP},\hat{x}_i,\hat{\mathsf{AP}})$ is an EvalAP collision for the Merkle Tree hash MT built from $\mathcal{H}$. It follows from Lemma 1 and the collision-resistance of $\mathcal{H}$ that $\Pr[W_3]\leq\varepsilon_{\mathcal{H},CR}=\mathsf{negl}(\lambda)$.

We conclude that $\Pr[W]=\Pr[W_1]+\Pr[W_2]+\Pr[W_3]=\mathsf{negl}(\lambda)$, assuming the collision-resistance of $\mathcal{H}$. ∎

Theorem 2 (Pseudorandomness): The authenticated MT-iVRF construction of Subsection IV-B satisfies Pseudorandomness, if the pseudorandom number generator G satisfies Pseudorandomness (Def. 2) and the hash function family $\mathcal{H}$ satisfies the Pseudorandom Function (PRF) properties (Def. 1).

Proof: Let W be the event that the adversary wins, i.e. $b=b'$ and $(i^*,j^*)\notin\mathcal{I}$. We use a game based approach to prove the theorem.

$\mathbf{Game}_0$: This is identical to Exp-PRand except that the challenger picks at the beginning of the game a uniformly random guess $(\hat{i},\hat{j})\xleftarrow{\$}\{0,\ldots,N-1\}\times\{0,\ldots,t-1\}$ for the challenge indices (i^*,j^*) output by $\mathcal{A}_1$. Let E be the event that the guess is correct, i.e. $(\hat{i},\hat{j})=(i^*,j^*)$. We are interested in $\Pr[W]$ and hence we have that:

$$\begin{aligned}\Pr_{\mathbf{Game}_0}[W]&=\Pr_{\mathbf{Game}_0}[W\cap E]/\Pr_{\mathbf{Game}_0}[E]\\&=Nt\cdot\Pr_{\mathbf{Game}_0}[W\cap E].\end{aligned}$$

The first equality holds since W and E are independent in $\mathbf{Game}_0$. The second equality is deduced from $\Pr[E] = 1/(Nt)$ as the attacker's view is independent of what we chose here. In the following games we will trace both $\Pr[W \cap E]$ and $\Pr[E]$ at each game.

$\mathbf{Game}_1$: This game is identical to previous game except we replace G.Next output $(x_{0,0}, \ldots, x_{N-1,0})$ in line 1 of $\mathsf{iAV.Eval}_{\mathsf{sk_{av}}}$ and line 2 of $\mathsf{iAV.Keygen(pp)}$ algorithms with uniformly random and independent random elements in $\{0,1\}^{2\lambda}$. We have that $\Pr_{\mathbf{Game}_1}[E] \leq 1/(Nt) + \varepsilon_{\mathsf{G}}$, in which ε_{G} is the advantage of an adversary in distinguishing the outputs of G from random, as in Def. 2.

$\mathbf{Game}_2$: This game is identical to the previous game, except that we replace $x_{\hat{i},\hat{j}}$ in line 3 of $\mathsf{iAV.Keygen(pp)}$ by an independent uniformly random element $\hat{y}$ in $\{0,1\}^{2\lambda}$ (instead of $x_{\hat{i},\hat{j}} = H^{t-1-\hat{j}}(x_{i,0})$ used in $\mathbf{Game}_1$), and we also accordingly set $y := \hat{y}$ in line 2 of $\mathsf{iAV.Eval}_{\mathsf{sk_{av}}}$ when queried at $(i,j) = (\hat{i},\hat{j})$ and $y := \mathsf{H}^{\hat{j}-j}(\hat{y})$ when queried at $(i,j) = (\hat{i},j)$ for $j < \hat{j}$ by an independent uniformly random element in $\{0,1\}^{2\lambda}$ (note that queries with $i = \hat{i}$ and $j > \hat{j}$ do not occur if the events of interest E or $E \cap W$ occur). By applying the assumed Pseudorandom Function property (PRF) of $\mathcal{H}$ (with an empty string PRF input argument) at most t times, we get $\Pr_{\mathbf{Game}_2}[E] \leq 1/(Nt) + \varepsilon_{\mathsf{G}} + t \cdot \varepsilon_{\mathsf{PRF}}$, where $\varepsilon_{\mathsf{PRF}}$ is the maximal advantage of an adversary with run-time $T_{\mathcal{A}}$ against the PRF security of $\mathcal{H}$. Note that in this argument we replace the output of $\mathsf{H}^{\alpha}(\cdot)$ for $1 \leq \alpha \leq t$ with a random element because the PRF property of $\mathcal{H}$ can be applied sequentially to outputs of each iteration. More specifically, for a uniformly random input x, $\mathsf{H}(x)$ can be replaced by a random string according to the PRF property of $\mathcal{H}$. Now assuming the indistinguishability of $\mathsf{H}(x)$ from a random, one can replace $\mathsf{H}(\mathsf{H}(x)) = \mathsf{H}^2(x)$ with a random element. This process can be inductively iterated for any $\alpha \leq t$ to replace $\mathsf{H}^{\alpha}(x)$ with a random element assuming the indistinguishability of $\mathsf{H}^{\alpha-1}(x)$ from a uniformly random input to $\mathsf{H}(\cdot)$.

$\mathbf{Game}_3$: This game is identical to the previous game, except that we replace $x_{\hat{i},\hat{j}+1}$ in line 2 of $\mathsf{iAV.Keygen(pp)}$ by an independent uniformly random element $\hat{y}'$ in $\{0,1\}^{2\lambda}$ (instead of $x_{\hat{i},\hat{j}+1} = \mathsf{H}(\hat{y})$ used in $\mathbf{Game}_2$), and we also accordingly set $v := \hat{v}$ for another independent uniformly random element $\hat{v} \in \{0,1\}^{2\lambda}$ in line 3 of $\mathsf{iAV.Eval}_{\mathsf{sk_{av}}}$ when queried at $(i,j) = (\hat{i},\hat{j})$ (instead of $v = \mathsf{H}(\hat{y},\mu_1)$ in $\mathbf{Game}_2$) and $y := \mathsf{H}^{\hat{j}-j-1}(\hat{y}')$ in line 2 when queried at $(i,j) = (\hat{i},j)$ for $j < \hat{j}$ (instead of $y = \mathsf{H}^{\hat{j}-j}(\hat{y})$ in $\mathbf{Game}_2$). Notice that we can construct an adversary against the assumed Pseudorandom Function property (PRF) of $\mathcal{H}$ with key $\hat{y}$ as the first input that makes two PRF oracle queries with key $\hat{y}$ (namely at the empty string PRF input to simulate $x_{\hat{i},\hat{j}+1} = \mathsf{H}(\hat{y})$ and at the PRF input μ_1 to simulate $v = \mathsf{H}(\hat{y},\mu_1)$), such that the $b = 0$ case of the PRF game simulates the view of $\mathbf{Game}_2$ to $\mathcal{A}$ while the $b = 1$ case simulates the view of $\mathbf{Game}_3$ to $\mathcal{A}$. Hence, we get $\Pr_{\mathbf{Game}_3}[E] \leq 1/(Nt) + \varepsilon_{\mathsf{G}} + (t+1) \cdot \varepsilon_{\mathsf{PRF}})$, where $\varepsilon_{\mathsf{PRF}}$ is the maximal advantage of an adversary with run-time $T_{\mathcal{A}}$ against the PRF security of $\mathcal{H}$.

Now observe that in $\mathbf{Game}_3$, if E occurs then v_0 is perfectly indistinguishable from v_1 since they are both uniformly random and independent of $\mathcal{A}$'s view. It follows that $\Pr_{\mathbf{Game}_3}[W|E] = 1/2$. We further deduce that:

$$\begin{aligned}\Pr_{\mathbf{Game}_3}[W \cap E] &= \Pr_{\mathbf{Game}_3}[E]/2 \\ &\leq \frac{1}{2}(1/(Nt) + \varepsilon_{\mathsf{G}} + (t+1) \cdot \varepsilon_{\mathsf{PRF}})),\end{aligned}$$

where in the last equality we have used the above expression for $\Pr_{\mathbf{Game}_3}[E]$. Putting all these together, we have that:

$$\begin{aligned}\Pr[W]_{\mathbf{Game}_0} &= Nt \cdot \Pr_{\mathbf{Game}_0}[W \cap E] \qquad (1)\\ &\leq Nt \cdot (\Pr_{\mathbf{Game}_3}[W \cap E] + \varepsilon_{\mathsf{G}} + (t+1) \cdot \varepsilon_{\mathsf{PRF}}) \\ &\leq Nt \cdot (1/(2Nt) + 3\varepsilon_{\mathsf{G}}/2 + 3(t+1) \cdot \varepsilon_{\mathsf{PRF}}/2) \\ &= 1/2 + \varepsilon.\end{aligned}$$

Since $\varepsilon_{\mathsf{G}} = \mathsf{negl}(\lambda)$,$\varepsilon_{\mathsf{PRF}} = \mathsf{negl}(\lambda)$ and $Nt = \mathsf{poly}(\lambda)$, we conclude that $\varepsilon := (Nt/2) \cdot (3\varepsilon_{\mathsf{G}} + 3(t+1) \cdot \varepsilon_{\mathsf{PRF}}) = \mathsf{negl}(\lambda)$. ■

Theorem 3 (Forward-Secure Unforgeability): The authenticated MT-iVRF construction of Subsection IV-B is forward-secure, if the underlying signature Σ is t-time unforgeable against CMA and the stateful pseudorandom number generator G satisfies Forward-Security (Def. 2).

Proof: We conduct the proof by the following games from game $\mathbf{Game}_0$ to game $\mathbf{Game}_3$. For each game $\mathbf{Game}_i$, we use Win_i to denote the event that adversary $\mathcal{A}$ wins Exp-Forge in $\mathbf{Game}_i$. Without loss of generality, we assume that the adversary issues at most q queries to $\mathcal{O}_{\mathsf{iAV.Eval}}$.

$\mathbf{Game}_0$: This is identical to Exp-Forge.

$\mathbf{Game}_1$: This is identical to $\mathbf{Game}_0$ except we replace the output of G.Next in $\mathsf{iAV.KeyUpd}(\mathsf{sk_{av}})$ with random elements. It is obvious from the forward-security property of G that $|\Pr[\mathsf{Win}_1] - \Pr[\mathsf{Win}_0]| \leq \varepsilon$, where ε here is the advantage of an adversary in breaking the forward-security of G.

$\mathbf{Game}_2$: This is identical to $\mathbf{Game}_1$ except that the adversary can simulate (v,π) without knowing the private key $\mathsf{sk_{av}}$ of the signature. This is the case since we replaced the output of G.Next with random elements in previous game. Since there is no difference in the view of the attacker from the previous game, we have $\Pr[\mathsf{Win}_2] = \Pr[\mathsf{Win}_1]$. Since we have used N consecutive sum composition of Σ (that is $\Sigma^{\oplus}_{\log N}$ according to [16]) in our scheme, this game is now identical to an experiment with adversary $\mathcal{B}$ against the forward-security of $\Sigma^{\oplus}_{\log N}$. In particular, Theorem 1 of [16] implies that: $\Pr[\mathsf{Win}_2] \leq N \cdot \Pr[\mathcal{B}$ wins Exp-UF-CMA$]$.

Overall, we have that

$$\begin{aligned}\Pr[\mathsf{Win}_0] &\leq \Pr[\mathcal{A} \text{ wins Exp-Forge}] \\ &\leq N \cdot \Pr[\mathcal{B} \text{ wins Exp-UF-CMA}] + \varepsilon \leq \mathsf{negl}(\lambda),\end{aligned}$$

where the last inequality holds since $N = \mathsf{poly}(\lambda)$, and $\varepsilon = \mathsf{negl}(\lambda)$, $\Pr[\mathcal{B}$ wins Exp-UF-CMA$] = \mathsf{negl}(\lambda)$ by the assumed forward-security of G and t-time unforgeability of Σ, respectively. ■

Theorem 4 (Unbiasability): The authenticated MT-iVRF construction of Subsection IV-B satisfies Unbiasability, if the hash function family $\mathcal{H}$ satisfies Everywhere Preimage Resistance (ePre) in the sense of Def. 1.

Proof: Let $\mathbf{Game}$ denote the Exp-Bias game running with a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. We denote the adversary run time by $T_{\mathcal{A}}$. On input $\mathsf{pp} = \mathsf{H}$, where $\mathsf{H} \xleftarrow{\$} \mathcal{H}$, $\mathcal{A}_1$ returns $(\mathsf{pk}_{\mathsf{av}}, \mu_2, v, i, j, \mathsf{st})$, and on input (μ_1, st), where $\mu_1 \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$, $\mathcal{A}_2$ returns (σ, π), where $\pi = (y, \mathsf{pk}, \mathsf{AP})$. Let W denote the event that $\mathcal{A}$ wins in $\mathbf{Game}$, which implies that $\mathsf{H}(y, \mu_1) = v$. Then we can construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ breaking ePre of $\mathcal{H}$ with run-time at most $T_{\mathcal{A}}$ and advantage $Adv_{ePre}(\mathcal{B}) \geq \Pr[W]$, which implies that $\Pr[W] \leq 2^{-\ell(\lambda)} + \mathsf{negl}(\lambda)$ by the assumed ePre security of $\mathcal{H}$. Namely, on input $\mathsf{H} \xleftarrow{\$} \mathcal{H}$, $\mathcal{B}_1$ runs $\mathcal{A}_1$ on input H to get $(\mathsf{pk}_{\mathsf{av}}, \mu_2, v, i, j, \mathsf{st})$ and $\mathcal{B}_1$ returns (v, st), and on input (μ_1, st) for $\mu_1 \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$, $\mathcal{B}_2$ runs $\mathcal{A}_2$ on (μ_1, st) to get (σ, π), where $\pi = (y, \mathsf{pk}, \mathsf{AP})$, and $\mathcal{B}_2$ returns y. ■

VII. Application to Algorand

The Algorand protocol is a fork-free PoS protocol in which consensus is achieved using a Byzantine Agreement (BA) protocol [26], [27]. To prevent the adversary from adaptively corrupting parties who participate in the protocol, the parties who are actively running the protocol change after every step[11] of the protocol. This is achieved using a VRF that takes into account parties' stakes. The security of the signature primitives (including the VRF), and, by extension, of the BA protocol, rely on computational assumptions that are known to be broken in the context of a quantum adversary. Hence, to attain post-quantum security for this protocol, it is necessary to shift to alternate constructions for these primitives. The purpose of this section is to provide a high-level explanation of the relevant aspects of Algorand's BA protocol that should be replaced with post-quantum secure alternatives.

We begin by explaining how parties register their keys to the Algorand blockchain [28]. To create an account, a party registers its public account key apk to the blockchain using a registration transaction that includes the spending public key and the amount of coins belonging to the account owner. At each step of the BA protocol, parties individually run the sortition algorithm to check if they are eligible to participate in the consensus protocol. The sortition is implemented using a VRF (see below for how parties register their VRF keys).

At round n, a party computes the output of their VRF on input their VRF secret key vsk and a special quantity Q_n, which is derived from the blockchain. The main idea is that Q_n will be unpredictable to any party at the time of registering its public key to the blockchain. Therefore, it also should not be able to predict if it will be eligible to participate in the consensus protocol at round n. We elide a detailed description of how the protocol computes Q_n, as it is not relevant for explaining the sortition mechanism.

The initial quantity Q_0 is assumed to be randomly generated (as part of the genesis block). At round n, the previous round's magic number Q_{n-1} is used. Particularly, the *leader* of period j of round n is elected as the party P_i with the lowest VRF value (proportional to its stakes) satisfying[12] [13]

$$\mathsf{H}(\Sigma.\mathsf{Sign}_i(n, j, 1, Q_{n-1})) \leq p \cdot stake(P_i), \tag{2}$$

where p is a predefined threshold value and $stake(P_i)$ denotes the amount of stakes owned by P_i. More generally, parties determine their eligibility to participate in the protocol for a given step $s > 1$ (of period j) of the BA protocol by checking whether

$$\mathsf{H}(\Sigma.\mathsf{Sign}_i(n, j, s, Q_{n-1})) \leq p' \cdot stake(P_i), \tag{3}$$

where $p' \gg p$ is a predefined value.

Every period of the protocol is associated with a leader who is also elected via VRF. The BA protocol may have several periods (each consisting of a constant number of steps with rotating participants). Each potential round leader in round n proposes a block including new valid transactions together with its VRF proof for round n and broadcasts them to the network. After the blocks are proposed by the potential leaders, all participants of the protocol verify the correctness of the VRF outputs and choose the candidate block with the lowest VRF value. Once a block is accepted by collecting a strong majority of votes ($2/3$ of the votes) from the active protocol participants, parties move on to the next round.

A. Key Management in Algorand

In Algorand, there are four types of keys: *spending keys*, *VRF (selection) keys*, *voting keys* and (recently introduced) *state proof keys* [28]. Spending keys, also known as root keys, are used for sending and receiving coins by an account. An account is identified with the root public key apk of the spending key. Also, later on, the VRF, voting and state proof keys of the account are validated via the spending keys.

VRF keys are used to check if an account is selected for participation in the voting phase as the leader or, more generally, as an active participant in the BA protocol. Hence, all protocol messages are validated with a VRF proof.

Voting keys are used to authenticate participation during the voting phase of the BA agreement protocol. To achieve *forward security*, voting keys are periodically updated. This simple update works as follows: parties delete the latest used private key, and move to the next one. To avoid registering a new voting key each time, per epoch, a batch of ephemeral voting keys $(epk_1, esk_1), (epk_2, esk_2), ...$ (10,000 as the default value) are generated. These keys are authenticated using a signature relative to the root voting key apk, which is validated by the signature of the spending key. Thus, a batch of keys are committed and (partially) validated as follows whenever a new epoch begins. First, the user registers a new root key epk, which it validates by signing with the spending key of the root public key apk of the previous epoch. Each ephemeral voting key epk_i is later validated using epk.

[11] Note that a *round* of the Algorand protocol may include several *periods*, each consisting of a constant number of *steps*, namely 5 steps [26]. In our iVRF formal model, we use two indices (one for rounds and the other for iterations), and each step of a period in Algorand increments one iteration in our iVRF model (i.e., there is no need to introduce a third index in the formal model, which would unnecessarily complicate its presentation).

[12] The signature scheme used in Algorand's VRF (given in Eqn. 2) has the uniqueness property.

[13] Note that in the Algorand protocol given in [26], the period parameter j is omitted since each round is described over one period, whereas the current implementation of the protocol explained in [28] may include several periods per round.

In theory (to ensure full forward security), voting keys should be updated immediately after their use in a step of the BA protocol. However, in practice, they are updated once in every round [28]. This results in a slightly weaker form of security where a fully adaptive adversary may indefinitely stall progress of the protocol. However, such a strong adversary appears somewhat unrealistic in practice.

Recently, Algorand also introduced state proof keys that are used to generate post-quantum secure state proofs. Similar to voting keys, state proof keys consist of ephemeral keys that are renewed in epochs.

B. Our Modifications to the Algorand Protocol

We aim to achieve a post-quantum version of Algorand protocol by replacing cryptographic algorithms that are vulnerable to quantum attacks. In March 2022, concurrently and independently from our work, Algorand added *state proofs* to the protocol to improve its resilience against quantum attacks. In state proofs, a subset-sum-based hash function [29] and the Falcon signature algorithm [19] are used. However, the rest of the protocol mainly utilizes pre-quantum algorithms. Algorand uses SHA-512/256 and EdDSA [30], [31] as primary hash function and signature scheme, respectively.

The security of the Algorand protocol relies on a strong honest majority among each of the BA committees that actively run the protocol. With the above choices, the Algorand protocol achieves its desired security properties with overwhelming probability ($1 - 10^{-18}$ given appropriate network conditions) [2]. The committee and leader elections for the BA protocol are done via the VRF function. However, since the VRF function is instantiated with the pre-quantum algorithms, these election protocols are not post-quantum secure[14]. In our protocol, we can replace the signature scheme EdDSA with any post-quantum signature, and particularly propose to use Falcon signature since it has the minimal total size for a public key and a signature among those selected by the NIST for post-quantum signature standardization. Also, we replace the VRF function with MT-iVRF, which is defined in Section IV. In fact, as discussed before, we can realize both VRF and signature functionalities by a single tool, our authenticated MT-iVRF from Sec. IV-B.

As we mentioned previously, the election in the BA protocol is done via the VRF algorithm with the input of Q_n value, which should satisfy *uniqueness*, *pseudorandomness* and *unbiasability* (See Section 5.6 and proof of Lemma 5.11 in [26]). Our iVRF construction satisfies somewhat different notions of uniqueness, pseudorandomness, and unbiasability than the ones stated in [26]. First of all, our formal model in Section III is designed to collectively capture the VRF and forward secure signature requirements in an Algorand-like blockchain setting, where the protocol is inherently indexed (or timed). Particularly, we assume the protocol operates in rounds (indexed by i), periods within rounds (indexed by j), and each period consists of a constant number of steps. We require in an Algorand-like blockchain setting that at any particular step of a period of any round, each (computationally bounded) user can only produce a single valid VRF value, as captured by our formal uniqueness model. Similarly, we need pseudorandomness to hold against indices that have not been queried before. That is, it is fine for the outputs from previous steps/rounds to not satisfy pseudorandomness, as the past unique VRF values have already served their purpose and are no longer relevant. Moreover, by combining the forward-secure signature scheme with iVRF, our construction given in Section IV-B reduces the cost of the validation. More specifically, we require only one Merkle Tree authentication path, rather than two in the separate construction case.

Thanks to the blockchain-oriented design of our formal definitions, the security properties achieved by our iVRF are sufficient to ensure the required properties on the BA committee elections that are leveraged in Algorand's proofs. As we showed in Section VI, our iVRF proposal satisfies the aforementioned properties, and thereby can be substituted in the original Algorand protocol without impacting its security.

Finally, recall that our MT-iVRF construction includes t pseudorandom strings ($x_{i,j}$) per round. Here, the iteration parameter t can be chosen based on the network and security assumptions. The BA protocol of Algorand is expected to terminate within at most 2.5 periods (which corresponds to 16 steps in total) [27]. As discussed in Sec. V, the parameter t has very little impact on the computational performance (and no impact on the communication size) and, hence, can be safely adjusted without a significant compromise in performance.

C. Interpreting Our iVRF Performance for Algorand

It is easy to see that the iAV.Keygen runtime is heavily dominated by Falcon key generations, which are needed anyway for forward security. As discussed earlier, this key generation process can be amortized over time or paralellized. For example for $N = 2^{23}$, splitting the computation into 4 cores (and using 4 random seeds instead of one to generate signature keys) reduces the required *once-a-year* computation time to just 2.6 hours. Alternatively, whenever the user's device is turned on, the signature keys can be progressively computed (and hashed to avoid storing the whole key). Therefore, in terms of keygen procedure, there is effectively no computational overhead over what already needs to be done to achieve forward security. Note that the ephemeral key generation process that already exists in Algorand together with a little more of effort for the generation of $x_{i,j}$'s and the Merkle tree can serve as iAV.Keygen.

From Table II, we can see that an evaluation (including signing) can be done under 5 ms, which is well below the Algorand's round time at 4.5 seconds. Therefore, a committee member's local authenticated iVRF evaluations are not expected to lead to any slowdown. We can also conclude that 50,000 or more iAV.Verify executions can be performed within the time period of a round. In fact, as seen in Tables I and II, our verification (including signature validation) runs even faster than ECVRF verification used by Algorand. Therefore, we also do not expect any slowdown due to verification.

As discussed in Sec. V, for the storage of the Merkle tree nodes, we need about $32N$ bytes, meaning only 256 MB is needed even for $N = 2^{23}$. Note that the existing Algorand

[14]We note here that the uniqueness property of ECVRF used in Algorand does not rely on any computational assumption [9] and, therefore, is plausibly post-quantum (in ROM). However, the pseudorandomness property requires DDH assumption [9] and, therefore, ECVRF as a whole is not post-quantum.

protocol already has a similar storage requirement where N pairs of an ephemeral key and a signature (each pair of size about 96 bytes) are stored.

Perhaps the only significant cost introduced with the use of our authenticated MT-iVRF in the Algorand setting is the increased communication. As discussed before, the vast majority of the increased communication cost stems from the use of a post-quantum forward-secure signature and there is only a 32-byte additional cost due to the VRF functionality. Such an additional cost of increased communication appears unavoidable in the current state of affairs when post-quantum security is desired as evidenced by the increased sizes of all schemes standardized by NIST[15].

VIII. Conclusion

In this work, we introduced a simple and efficient method to realize the VRF functionality required in the blockchain setting for the leader election problem. Our solution does not involve a racing condition as in Bitcoin and can be instantiated from well-known basic (post-quantum) primitives. We believe that our approach can be readily deployed in the Algorand blockchain system as only minor modifications are needed.

Acknowledgment

This research was supported in part by ARC Discovery Project grants DP180102199 and DP220101234.

References

[1] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT (2)*, ser. LNCS, vol. 10821. Springer, 2018, pp. 66–98.

[2] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 51–68. [Online]. Available: https://doi.org/10.1145/3132747.3132757

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

[4] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," *CoRR*, vol. abs/1805.04548, 2018. [Online]. Available: http://arxiv.org/abs/1805.04548

[5] "Chia documentation," 2018, available at https://docs.chia.net/docs/03consensus/consensus_intro.

[6] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *FOCS*. IEEE Computer Society, 1999, pp. 120–130.

[7] "Rangersprotocol whitepaper," 2022, available at https://www.rangersprotocol.com/pdf/RangersProtocolWhitepaper.pdf.

[8] "What sets it apart: Filecoin's proof system," 2020, available at https://spec.filecoin.io/algorithms/expected_consensus/.

[9] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin, and S. Goldberg, "Making nsec5 practical for dnssec," Cryptology ePrint Archive, Report 2017/099, 2017, https://ia.cr/2017/099.

[10] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-speed high-security signatures," in *CHES*, ser. LNCS, vol. 6917. Springer, 2011, pp. 124–142.

[11] M. F. Esgin, V. Kuchta, A. Sakzad, R. Steinfeld, Z. Zhang, S. Sun, and S. Chu, "Practical post-quantum few-time verifiable random function with applications to algorand," in *Financial Cryptography (2)*, ser. LNCS, vol. 12675. Springer, 2021, pp. 560–578.

[15] https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

[12] M. F. Esgin, R. Steinfeld, D. Liu, and S. Ruj, "Efficient hybrid exact/relaxed lattice proofs and applications to rounding and vrfs," Cryptology ePrint Archive, Report 2022/141, 2022, https://ia.cr/2022/141.

[13] M. Buser, R. Dowsley, M. F. Esgin, S. K. Kermanshahi, V. Kuchta, J. K. Liu, R. Phan, and Z. Zhang, "Post-quantum verifiable random function from symmetric primitives in pos blockchain," Cryptology ePrint Archive, Report 2021/302, 2021, https://ia.cr/2021/302.

[14] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS - A practical forward secure signature scheme based on minimal security assumptions," in *PQCrypto*, ser. LNCS, vol. 7071. Springer, 2011, pp. 117–129.

[15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *EUROCRYPT (1)*, ser. LNCS, vol. 9056. Springer, 2015, pp. 430–454.

[16] T. Malkin, D. Micciancio, and S. K. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *EUROCRYPT*, ser. LNCS, vol. 2332. Springer, 2002, pp. 400–417.

[17] "Quantum-Safe Identity-based Encryption," The European Telecommunications Standards Institute, Sophia-Antipolis, France, Technical Report, 2019.

[18] R. K. Zhao, S. McCarthy, R. Steinfeld, A. Sakzad, and M. O'Neill, "Quantum-safe hibe: does it cost a latte?" Cryptology ePrint Archive, Paper 2021/222, 2021, https://eprint.iacr.org/2021/222. [Online]. Available: https://eprint.iacr.org/2021/222

[19] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, 2018.

[20] M. Bellare, C. Namprempre, and G. Neven, "Security proofs for identity-based identification and signature schemes," in *EUROCRYPT*, ser. LNCS, vol. 3027. Springer, 2004, pp. 268–286.

[21] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *FSE*, ser. Lecture Notes in Computer Science, vol. 3017. Springer, 2004, pp. 371–388.

[22] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 431–448.

[23] NIST, "Recommendation for random number generation using deterministic random bit generators," https://doi.org/10.6028/NIST.SP.800-90Ar1, 2015.

[24] S. Gueron, "Intel's new AES instructions for enhanced performance and security," in *FSE*, ser. Lecture Notes in Computer Science, vol. 5665. Springer, 2009, pp. 51–66.

[25] D. J. Bernstein, "Chacha, a variant of salsa20," https://cr.yp.to/chacha/chacha-20080128.pdf, 2008.

[26] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theor. Comput. Sci.*, vol. 777, pp. 155–183, 2019.

[27] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, "ALGORAND AGREEMENT: super fast and partition resilient byzantine agreement," *IACR Cryptol. ePrint Arch.*, p. 377, 2018.

[28] A. Foundation, "Algorand key specification," 2022, available at: https://github.com/algorandfoundation/specs/blob/master/dev/partkey.md.

[29] Y. Gilad, D. Lazar, and C. Peikert, "Subset-sum hash specification," 2021, available at: https://github.com/algorandfoundation/specs/blob/master/dev/cryptographic-specs/sumhash-spec.pdf.

[30] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," *RFC*, vol. 8032, pp. 1–60, 2017.

[31] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography*, ser. Lecture Notes in Computer Science, vol. 3958. Springer, 2006, pp. 207–228.