MUSEN: Aggregatable Key-Evolving Verifiable Random Functions and Applications

Bernardo David[1⋆], Rafael Dowsley[2], Anders Konring[3⋆⋆], and Mario Larangeira[45⋆⋆⋆]

[1] IT University of Copenhagen, Denmark
bernardo@bmdavid.com
[2] Monash University, Australia
rafael.dowsley@monash.edu
[3] Espresso Systems
anders@espressosys.com
[4] Tokyo Institute of Technology, Japan
mario@c.titech.ac.jp
[5] IOG, Singapore
mario.larangeira@iohk.io

Abstract. A Verifiable Random Function (VRF) can be evaluated on an input by a prover who holds a secret key, generating a pseudorandom output and a proof of output validity that can be verified using the corresponding public key. VRFs are a central building block of committee election mechanisms that sample parties to execute tasks in cryptographic protocols, *e.g.*, generating blocks in a Proof-of-Stake (PoS) blockchain or executing a round of MPC protocols. We propose the notion, and a matching construction, of an Aggregatable Key-Evolving VRF (A-KE-VRF) with the following extra properties: 1. Aggregation: combining proofs for several VRF evaluations of different inputs under different secret keys into a single constant size proof; 2. Key-Evolving: preventing adversaries who corrupt a party (learning their secret key) from "forging" proofs of past VRF evaluations. As an immediate application, we improve on the block size of PoS blockchains and on the efficiency of Proofs of Proof-of-Stake (PoPoS). Furthermore, the A-KE-VRF notion allows us to construct Encryption to the Future (EtF) and Authentication from the Past (AfP) schemes with a Key-Evolving property, which provides forward security. An EtF scheme allows for sending a message to a party who is randomly selected to execute a *role* in the future, while an AfP scheme allows for this party to authenticate their messages as coming from a past execution of this role. These primitives are essential for realizing the YOSO MPC Framework (CRYPTO'21).

⋆ This work was supported by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C) and 0165-00079B.
⋆⋆ This work was supported by the Independent Research Fund Denmark (IRFD) grant number 9040-00399B (TrA^{2}C).
⋆⋆⋆ This work was supported by JSPS KAKENHI Grant Number JP21K11882.

1 Introduction

The capability of publicly showing that one has generated a pseudorandom value has found numerous applications in cryptography. Verifiable Random Functions (VRF), the very cryptographic primitive that captures this capability, as proposed by Micali *et al.* [29], have been used in the setting of Proof-of-Stake (PoS) protocols [13, 24, 16]. Briefly, the randomness generated by the VRF primitive is employed to select leaders in the crucial *leader election* procedure which is ubiquitous in the PoS based systems. Likewise, it can used in the selection of a subset of participants (*i.e.*, a committee) to execute cryptographic protocols (*e.g.*, Multi-party computation (MPC) protocols in the YOSO model [21]).

The notion of Key-Evolving (KE) signatures was first formalized by Bellare and Miner [4]. The KE property addresses the threat that signatures generated a long time ago under a particular signing key can be forged at some point in the future if the signer is corrupted or leaks their signing key. A KE signature scheme assigns a specific signing key to each time slot and allows the signer to "evolve" the signing key for the current slot in order to obtain the signing key for the next slot, while keeping the public key static. Relying on secure erasures, such a scheme achieves forward security, since the signer can periodically evolve their signing key and erase the previous version (usually before outputting a signed message). Hence, in the event of a corruption and leakage of the secret key, the adversary can only forge messages from that point on. Surprisingly, even though VRFs are similar to digital signatures, they do not have a KE counterpart.

Another issue in many applications of VRFs is that very often protocols require multiple instances of VRFs to be constantly evaluated under different secret keys, thereby issuing multiple VRF proofs and ouput values. Thus, a convenient capability along with KE is *aggregation*. Namely, the ability to combine proofs for several VRF evaluations of different inputs under different secret keys into a single constant-sized proof that can be used to verify the outputs of all evaluations. While Key-Evolving Aggregate Signatures were proposed before [26, 25], no VRFs with equivalent properties exist.

VRFs for PoS and Proofs of PoS systems. Forward security is particularly interesting for PoS protocols for two main reasons: (1) these are protocols executed continuously for long periods of time; (2) during their execution, parties can become corrupted or simply stop taking security measures to protect signing keys. The biggest issue comes from adaptive adversaries who may instantly corrupt a party as soon as they publish a block and use their signing keys to generate alternative versions of the block. Moreover, there are more subtle issues such as that of *nothing at stake attacks*. Imagine a party who no longer has stake in a currently running protocol and thus has no incentive to spend resources to properly secure or to erase their old signing key. Such a party can inadvertently leak their signing key or even sell it for a profit. Current PoS blockchain protocols [13, 24, 16] employ KE signature schemes and secure erasures to achieve forward security and prevent such attacks.

Furthermore, it is well known that the size of the block is crucial not only in PoS, but also Proof-of-Work (PoW) systems. Any extra space available in blocks is welcome [14, 27]. The ability to aggregate values inside the block opens an opportunity to lower both the size of a single block and the storage requirements for multiple blocks, a natural application for blockchains.

Encryption to the future. The recent You Only Speak Once (YOSO) MPC [21] framework work addresses the issues of adaptive security and scalability in Multi-party Computation (MPC) protocols by using different anonymous committees chosen uniformly at random to execute each round of the protocol. The idea is that smaller committees can execute each round of the protocol instead of requiring all parties to execute all rounds. In order to prevent an adaptive adversary from immediately corrupting a a large enough fraction of a committee and breaking security, the parties in each committee remain anonymous until they act. The YOSO MPC framework assumes ideal secure channels that allow for sending messages to anonymous parties chosen uniformly at random. In order to concretely realize this model, an Encryption to the Future (EtF) primitive that allows for sending messages to anonymous parties chosen at random was introduced in another recent work by Campanelli *et al.* [8].

The Encryption to the Future (EtF) notion and constructions introduced in [8] and further extended in [11] require all ciphertexts to be posted on a PoS blockchain ledger. This is necessary both for sending the ciphertext towards its anonymous receiver(s), for establishing a notion of time and for random anonymous receiver selection using the PoS blockchain's intrinsic leader election mechanism. However, this introduces a grave issue pertaining to forward security: if a party is corrupted at some point in the future, the adversary can readily access and decrypt all ciphertexts for which this party was elected as receiver in the past (plus all of those in the future). This issue also extends to the notion of Authentication from the Past (AfP) introduced in [8], which allows a party selected to perform a certain role to sign messages on behalf of this role. Thus, an adversary who corrupts a party is able to sign messages on behalf of roles for which this party has been selected in the past.

1.1 Our Contributions

Our central contribution is the UC notion of A-KE-VRF along with a matching construction based on BLS signatures [6]. Our construction allows for both aggregation of VRF proofs and for a key evolving property that results in forward security. Moreover, our construction can be endowed with a output extension feature similar to that of [3], where a single VRF evaluation on a given input yields several independent pseudorandom outputs. This construction lends itself to several applications, such as improving the efficiency of PoS blockchains and adding forward security to Encryption to the Future and Authentication from the Past. Now we further detail features of the novel notion and construction.

Pros and Cons of game vs simulation based definitions: When defining our notion of A-KE-VRF, we must provide a definition that can be readily integrated with protocols that use this sort of primitive while capturing all the properties we want. Definitions of VRFs with a unbiasability property were provided in [16, 3]. Game based definitions of forward secure VRFs have been proposed in [17], although they have weaker unbiasability guarantees than the aforementioned simulation based definitions. Game based definitions make each individual property very clear and may be easier for people not familiar with UC to understand. However, we need a lot of different properties, so we end up with many different security games, making the presentation of definitions and the security analysis cumbersome. Moreover, we lose the ability of analysing security under composability. A simulation based definition can be captured in a single ideal functionality and gives us the chance to prove composability. Hence, we opt for a UC ideal functionality and show in Appendix B that it implies the game based properties of [17]. However, proving universal composability for our candidate construction requires carefully dealing with the random oracle.

Aggregation with signatures: We observe that an A-KE-VRF is a key evolving signature (KES) scheme with EUF-CMA security. When used as a signature, only the proof must be provided and the output can be discarded allowing for succinct aggregation of signatures into aggregated VRF proofs. This observation allows for the substitution of the KES, which has typically larger keys and signatures. The aggregation and signature properties, if combined, allow for using an A-KE-VRF both as a VRF and as a KE signature scheme when producing blocks in a blockchain system (more on that later regarding PoS applications).

Providing multiple outputs per evaluation: When instantiating a PoS blockchain via the Ouroboros Praos approach, we need to perform two independent VRF evaluations: one for leader election and another for implementing a bounded bias random beacon. It is important to have independent evaluations because otherwise there would be too much bias on the random beacon output, which in turn would affect future leader elections. These independent evaluations are obtained by evaluating the same VRF under the same keys on inputs appended with different suffixes to partition the VRF domain (*e.g.*, given epoch randomness r evaluate the VRF on input $r|\texttt{LEADER}$ to get the leader election output and then on input $r|\texttt{BEACON}$ to get the beacon protocol message). However, this results in two separate proofs, which we would have to aggregate. A different approach is proposed in [3], where one VRF evaluation is used to derive many independent outputs via randomness extraction using a random oracle. While this approach can be generically applied to our results, our construction can directly derive multiple outputs from one VRF proof (*i.e.*, a BLS signature), by defining each output as $y_i = H_i(\sigma)$ where $H_i(\cdot)$ is a different instance of the random oracle and σ is a VRF proof.

Applications to Proof of Stake (PoS). We concretely show how A-KE-VRFs can be used to obtain a smaller version of block headers in PoS protocols such as [13, 24, 16], which employ VRF-based secret leader election to select the parties who produce each block. In a nutshell, we describe two versions of the block header using A-KE-VRF properties to reduce storage requirements: (1) has two VRF proofs, and does not rely on KES primitive (it relies on the VRF signature-like property cited earlier), (2) relies on the aggregation of both proofs into a single one. It should be noted that in case of (2), the adversary may impose a larger bias in the leader selection, therefore a more conservative set of parameters should be devised for a safe protocol. This parameter estimation is out of the scope of this work.

Applications to Proof of PoS. Our A-KE-VRF can also be used to improve the concrete complexity of the Proof of PoS scheme by Agrawal et al. [2]. Here we use the aggregation in order to provide smaller representations of the blockchain state that must be shown when providing a Proof of PoS.

Applications to Encryption to the Future. We introduce definitions for both EtF and AfP with forward security, *i.e.* with the extra guarantee that adversaries who corrupt a party cannot decrypt EtF ciphertexts encrypted towards roles for which this party was selected in the past nor generate AfP tags on behalf of such roles. We build on the property of evolving keys of the A-KE-VRFs we propose to provide matching constructions.

1.2 Related Works

VRFs were first introduced in [29] and UC notions of this primitive have been proposed in [16] and in [3]. A primitive akin to a VRF but tailor-made for the Algorand blockchain protocol with key evolution/forward security was proposed in [17], which does not consider aggregation of VRF proofs or general definitions of VRFs. Key Evolving signatures (which yield forward security) were introduced in [4] and subsequently improved in [1, 23, 7]. Sequentially aggregatable and forward secure signatures were introduced in [26, 25], which does not consider general aggregation or VRF properties. An aggregated lottery scheme suitable for Proof-of-Stake blockchain protocols was proposed in [18]. This scheme is also analysed in the UC framework but is constructed from aggregatable commitments, allowing for an even more compact representation of the aggregated lottery results than what we can achieve by aggregating VRF proofs while keeping VRF outputs on the block. However, it does not offer forward security, which is paramount for our applications to Encryption to the Future and also allows for further improvements in the Proof-of-Stake application. While notions of forward secure (aggregatable) signatures and forward secure VRF-like primitives have been proposed in current literature, none of these primitives allow for obtaining both aggregation and forward security while achieving the pseudorandomness and unbiasability properties required from modern VRFs.

Independent work: A similar notion and constructions of key-homomorphic aggregate VRF are independently introduced in [28], which also suggests an application to reducing storage in Proof-of-Stake blockchains. However, their notion and constructions are not proven secure under arbitrary composition and do not encompass forward security. We remark that our construction is computationally more efficient that the construction based on bilinear pairings shown in [28]. On the other hand, a LWE based construction is also presented in [28].

2 Preliminaries

We denote sampling an element x uniformly at random from a set $\mathcal{X}$ as $x \xleftarrow{\$} \mathcal{X}$, and the range of integers $a, a+1, a+2, \ldots, b$ as $[a, b]$.

Assumptions. Our scheme is defined over groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, g_T \in \mathbb{G}_T$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which is non-degenerate. Our A-KE-VRF scheme is secure in the random oracle model given that the BLS signature scheme [6] is existentially unforgeable under chosen message attacks (*i.e.* EUF-CMA secure). Since the EUF-CMA security of BLS holds under the Gap-Diffie-Hellman (GDH) problem on bilinear pairings as defined in [6], our scheme's security also reduces to GDH.

Verifiable Random Functions. We will use A-KE-VRF to build forward secure Encryption to the Future (EtF) and Authentication from the Past (AfP) schemes following the approach of [8]. These EtF and AfP schemes will require a concrete representation of the VRF algorithms with game-based security definitions. Hence, in Appendix B we present standard game-based definitions for VRFs and show that our UC security notion implies these definitions.

Encryption to the Future (EtF)/Authentication from the Past (AfP). EtF allows for encrypting messages towards an anonymous party randomly selected to play a *role* at some point in the future. AfP allows for this party to sign a message on behalf of this role. We recall in verbatim form the definitions of EtF and AfP from [8] in Appendices D and E.

The UC Security Framework. We prove our A-KE-VRF scheme secure in the Universal Composability framework of [9], which allows for reasoning about security under arbitrary composition. This is paramount when using the A-KE-VRF as part of other protocol (*e.g.* in our applications to PoS blockchains). We briefly recall the UC framework and the random oracle functionality $\mathcal{F}_{\text{RO}}$ in Appendix A and refer interested readers to [9] for further details.

3 Aggregatable Key-Evolving VRF

We present an ideal functionality for A-KE-VRFs and then provide a construction that UC-realizes this functionality. We show in Appendix B that this UC notion of A-KE-VRF implies game based definitions such as those of [17].

The first thorough treatment of signatures in the UC model was introduced in [10]. Subsequently, a line of work ([16, 3]) presented designs of VRF functionalities also in the UC model. We depart from [3] and now provide some intuition

behind the design of the functionality depicted in Figures 1 and 2 and how we extend the functionality of [3] with both aggregation and key-evolving features.

- *Aggregation.* We add an interface that allows for a party who knows evaluations of the VRF on different inputs with different outputs and proofs to aggregate the proofs into a single constant-size proof. Additionally, we add a specific aggregate proof verification interface that takes as input the tuples of verification keys, inputs and outputs, verifying if an aggregate proof is valid with respect to these tuples;
- *Key-Evolving.* A key-evolving signature functionality is presented in [16] and we adopt a similar approach such that an evaluation is associated with a specific time period k_{ctr} where $1 \leq k_{ctr} \leq T$ and T is the total number of key updates. An interesting modelling challenge is to allow for evaluations on multiple messages between key updates and, at the same time, not letting an adaptive adversary break forward security by corrupting a party immediately after one evaluation is done (before key update) to forge a signature for that period. We model this by using a vector of messages $\bar{m}$ which are evaluated atomically, leaving the adversary no opportunity to adaptively corrupt before the counter has been incremented;
- *Generic Range Extension.* As in [3], we are able to extend the range of the output of the VRF. If we assume that the output of the VRF evaluation is $(y, \pi) \leftarrow \mathsf{VRF.Eval}_{sk}(x)$ where $y \in \mathcal{Y}$, we merely apply a constant[6] number of random oracles $H_1, \ldots H_c$ to the output $y \in \mathcal{Y}$ such that the extended VRF output becomes $\mathsf{VRF\text{-}EXT.Eval}_{sk}(x) = (H_1(y), \ldots, H_c(y))$;
- *Consistent Verification.* A subtle detail in [3] makes the ideal adversary able to make the verification algorithm output different answers on the same message and proof. In short, the functionality does not store the answer if the verification is rejected, thus, opens the possibility that the ideal adversary later can make the functionality accept an identical record. To ensure consistency, we adopt an approach similar to [10] and store a "blacklist" of proofs that have been rejected.

3.1 Construction $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ (Figure 3)

Our construction, presented in Figure 3, is inspired by aggregation of BLS signatures [5] with the property of forward security as presented in [26, 25]. Roughly, to use such signatures as VRF outputs, we simply observe that BLS signatures are unpredictable and unique and the use of random oracles provides the desired pseudorandom outputs. Note that the construction in Figure 3 is not susceptible to the "rogue public key attack". This is because the evaluation includes both the public key vk and time period j, effectively, making the messages pairwise distinct. We formally state the security of $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ in Theorem 1.

[6] As in [3], we are motivated by the application where $c = 2$ and output (y_T, y_ρ) such that y_T corresponds to leader election and y_ρ supplies randomness for the upcoming epoch.

Functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ (1 of 2)

Functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an ideal adversary $\mathcal{S}$. It is parameterized by the total number of key updates T and maintains initially empty tables $M[\cdot,\cdot,\cdot]$ (VRF evaluations), $M_{\mathsf{Agg}}[\cdot]$ (aggregated proofs) and sets $L_{eval} \leftarrow \emptyset$ (honest VRF evaluations), $\mathcal{PK} \leftarrow \emptyset$ (public key information).

Key Generation. Upon receiving (KeyGen, sid) from $P \in \mathcal{P}$.

If $(P,\cdot,\cdot) \in \mathcal{PK}$, then ignore the request. Otherwise, send $(\mathsf{KeyGen}, sid, P)$ to $\mathcal{S}$ and, upon receiving $(\mathsf{VerKey}, sid, P, vk)$ from $\mathcal{S}$, check if $\forall(\cdot, vk', \cdot) \in \mathcal{PK} : vk \neq vk'$. If this is the case, then set $k_{ctr} \leftarrow 1$ and $\mathcal{PK} \leftarrow \mathcal{PK} \cup \{(P, vk, k_{ctr})\}$ and return $(\mathsf{VerKey}, sid, vk)$ to P. Otherwise, ignore the request.

Eval and Prove. Upon receiving $(\mathsf{EvalProve}, sid, vk, j, \bar{m})$ from P, verify that $\bar{m} \in \mathcal{X}^{\ell}$, $(P, vk, k_{ctr}) \in \mathcal{PK}$ and $k_{\mathsf{ctr}} \leq j \leq T$. If not, then ignore the message. Otherwise, set $k_{\mathsf{ctr}} = j + 1$ and send $(\mathsf{EvalProve}, sid, vk, j, \bar{m})$ to $\mathcal{S}$. Upon receiving the response $(\mathsf{EvalProve}, sid, vk, j, \bar{m}, \bar{\pi})$, parse $\bar{m}$ and $\bar{\pi}$ as $(m_1, \ldots, m_\ell)$ and $(\pi_1, \ldots, \pi_\ell)$, respectively, and do the following for $i \in \{1, \ldots, \ell\}$:

1. If $\exists M[vk', j', m'] = (y', S', Q')$ such that $\pi_i \in S'$ and $(vk', j', m') \neq (vk, j, m_i)$, then ignore the message.
2. Else if $M[vk, j, m_i] = \perp$, then sample a random $y_i \leftarrow \mathcal{Y}$ and set the value $M[vk, j, m_i] \leftarrow (y_i, \{\pi_i\}, \emptyset)$. Assign $L_{eval} \leftarrow L_{eval} \cup \{(vk, j, m_i, y_i)\}$.
3. Else if $M[vk, j, m_i] = (y_i, S, Q) \neq \perp$, set $M[vk, j, m_i] \leftarrow (y_i, S \cup \{\pi_i\}, Q)$.

Finally, collect all $\bar{y} = (y_1, \ldots, y_\ell)$ and respond with $(\mathsf{Evaluated}, sid, j, \bar{y}, \bar{\pi})$.

Malicious Eval. Upon receiving $(\mathsf{Eval}, sid, vk, j, m)$ from $\mathcal{S}$, if $m \notin \mathcal{X}$, proceed as:

1. If $\exists (P, vk', k_{ctr}) \in \mathcal{PK}$ where $vk' = vk$ and P is not corrupted: if $M[vk, j, m] = (y, S, Q)$ s.t. $S \neq \emptyset$, return $(\mathsf{Evaluated}, sid, y)$ to $\mathcal{S}$, else, ignore the message.
2. If $\exists (P, vk', k_{ctr}) \in \mathcal{PK}$ where $vk' = vk$ and P is corrupted: if $M[vk, j, m] = \perp$, set $M[vk, j, m] \leftarrow (y, \emptyset, \emptyset)$ where $y \xleftarrow{\$} \mathcal{Y}$ and return $(\mathsf{Evaluated}, sid, y)$.

Verification. Upon receiving a message $(\mathsf{Verify}, sid, vk, m, j, y, \pi)$ from a party V.

If $j \notin [1, T]$, set $f = 0$. Otherwise, let $(y', S, Q) \leftarrow M[vk, j, m]$ and proceed:

1. If $\exists (\cdot, vk', \cdot) \in \mathcal{PK}$ where $vk' = vk$ and $y' = y$ and $\pi \in S$, set $f \leftarrow 1$.
2. Else, if $\exists (P, vk', \cdot) \in \mathcal{PK}$ where $vk' = vk$ with P honest and $\pi \notin S$ or $y \neq y'$, set $f \leftarrow 0$ and, if $\pi \notin S$, set $Q \leftarrow Q \cup \{\pi\}$.
3. Else, if $y = y'$ and $\pi \in S$, set $f \leftarrow 1$. If $\pi \in Q$: Set $f \leftarrow 0$.
4. Else, send $(\mathsf{Verify}, sid, vk, j, m, y, \pi, L_{eval})$ to $\mathcal{S}$ and upon receiving the response $(\mathsf{Verified}, sid, vk, j, m, y, \phi)$, when $\phi = 0$ set $Q \leftarrow Q \cup \{\pi\}$; and when $\phi = 1$ set $S \leftarrow S \cup \{\pi\}$; lastly set $M[vk, j, m] \leftarrow (y, S, Q)$ and $f \leftarrow \phi$.

Finally, output $(\mathsf{Verified}, sid, vk, j, m, y, \pi, f)$ to V.

Adversarial Leakage. On input $(\mathsf{PastEval}, sid)$ from $\mathcal{S}$, return L_{eval} to $\mathcal{S}$.

Aggregate. Upon receiving $(\mathsf{Aggregate}, sid, (vk_1; j_1; m_1; y_1; \pi_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell; \pi_\ell))$ from any party A, then do the following:

1. Send the message $(\mathsf{Aggregate}, sid, (vk_1; j_1; m_1; y_1; \pi_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell; \pi_\ell))$ to the simulator $\mathcal{S}$ and await the response $(\mathsf{Aggregated}, sid, \pi_{\mathsf{Agg}})$.
2. Store the aggregated proof with the corresponding tuples such that $M_{\mathsf{Agg}}[(vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell)] \leftarrow (\{\pi_{\mathsf{Agg}}\}, \emptyset)$.
3. Return the message $(\mathsf{Aggregated}, sid, \pi_{\mathsf{Agg}})$.

Fig. 1. Ideal functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ (1 of 2).

Functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ (2 of 2)

AggVerification. Upon receiving $(\mathsf{AggVerify}, sid, (vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell), \pi_{\mathsf{Agg}})$ from any party V, check that $1 \leq j_i \leq T$, $\forall i \in \{1, \ldots, \ell\}$. If this check fails, set $f \leftarrow 0$ and skip the steps below. Otherwise, let $(S_{\mathsf{Agg}}, Q_{\mathsf{Agg}}) \leftarrow M_{\mathsf{Agg}}[(vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell)]$ and $(y'_i, S_i, Q_i) \leftarrow M[vk_i, j_i, m_i]$, $\forall i \in \{1, \ldots, \ell\}$ and proceed as follows:

1. If $\pi_{\mathsf{Agg}} \in S_{\mathsf{Agg}}$ and $\exists (P_i, vk'_i, \cdot) \in \mathcal{PK}$ with $vk_i = vk'_i$ and $y_i = y'_i$, $\forall i \in \{1, \ldots, \ell\}$, set $f \leftarrow 1$.
2. If $\exists i \in \{1, \ldots, \ell\}$ such that $\exists (P_i, vk'_i, \cdot) \in \mathcal{PK}$ where $vk_i = vk'_i$ and P_i is honest but $y_i \neq y'_i$, set $M_{\mathsf{Agg}}[(vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell)] \leftarrow (S_{\mathsf{Agg}}, Q_{\mathsf{Agg}} \cup \{\pi_{\mathsf{Agg}}\})$ and $f \leftarrow 0$.
3. If $\pi_{\mathsf{Agg}} \in S_{\mathsf{Agg}}$ and $y_i = y'_i$, $\forall i \in \{1, \ldots, \ell\}$, set $f \leftarrow 1$; if $\pi_{\mathsf{Agg}} \in Q_{\mathsf{Agg}}$ set $f \leftarrow 0$.
4. If $\forall i \in \{1, \ldots, \ell\}$ it holds that $(vk_i; j_i; m_i; y_i)$ is such that $\exists (P_i, vk'_i, \cdot) \in \mathcal{PK}$ where $vk_i = vk'_i$ and P_i is honest, $y_i = y'_i$ and $\pi_{\mathsf{Agg}} \notin Q_{\mathsf{Agg}} \cup S_{\mathsf{Agg}}$, then send $(\mathsf{AggVerify}, sid, (vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell), \pi_{\mathsf{Agg}}, L_{eval})$ to $\mathcal{S}$. Upon receiving the response $(\mathsf{AggVerified}, sid, (vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell), \phi)$, if $\phi = 0$ set $Q_{\mathsf{Agg}} \leftarrow Q_{\mathsf{Agg}} \cup \{\pi_{\mathsf{Agg}}\}$; if $\phi = 1$ set $S_{\mathsf{Agg}} \leftarrow S_{\mathsf{Agg}} \cup \{\pi_{\mathsf{Agg}}\}$. Finally set $M_{\mathsf{Agg}}[(vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell)] \leftarrow (S_{\mathsf{Agg}}, Q_{\mathsf{Agg}})$ and $f \leftarrow \phi$.
5. Else, set $f \leftarrow 0$ and set $Q_{\mathsf{Agg}} \leftarrow Q_{\mathsf{Agg}} \cup \{\pi_{\mathsf{Agg}}\}$.

Output $(\mathsf{AggVerified}, sid, (vk_1; j_1; m_1; y_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell), \pi_{\mathsf{Agg}}, f)$ to V.

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ (2 of 2).

Theorem 1. *Protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ UC-realizes $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ in the $\mathcal{F}_{\mathrm{RO}}$-Hybrid model assuming honest parties are able to perform secure erasures and that the BLS signature [6] is EUF-CMA secure (i.e. the Gap-Diffie-Hellman assumption).*

Proof. Consider an environment $\mathcal{Z}$ with input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ where λ denotes the security parameter. We will argue that for *any* such environment[7] $\mathcal{Z}$ and any adversary $\mathcal{A}$, running in polynomial time in λ, there exist a simulator $\mathcal{S}$ such that it holds that $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}, \mathcal{S}, \mathcal{Z}} \approx \mathrm{EXEC}_{\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}, \mathcal{A}, \mathcal{Z}}$.

Describing the Simulator. To argue the above indistinguishability, we describe a simulator $\mathcal{S}$ for every real-world adversary $\mathcal{A}$. The simulator $\mathcal{S}$ interacts with the functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and an internal copy of $\mathcal{A}$, producing towards the environment a transcript that is indistinguishable from that of the real protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ interacting with the adversary. Our approach is similar to that of [3] where specific events that causes the simulator to abort are identified through the description of the simulator and a subsequent analysis can then bound the probability of such an event happening resulting in a simulator that fails with negligible probability. During the simulation, $\mathcal{S}$ emulates towards the internal copy of the adversary $\mathcal{A}$ random oracles represented by functions H_0, H_1 and H_2. Finally, $\mathcal{S}$ has tables T_{pk} and T_{sk} storing honest public and private keys,

[7] Since we follow the work of [9], we also consider "appropriate" PPT environments and protocols. Meaning that an environment is assumed to be both balanced and identity-bounded and protocols are subroutine-respecting.

Protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$

$\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ is parameterized by a security parameter λ, a number of key updates T, message space $\mathcal{X}$, output range $\mathcal{Y}$, and a bilinear map represented by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$. We use as setup random oracles $H_0 : \mathbb{Z}_p \to \mathbb{Z}_p$, $H_1 : \mathbb{G}_2 \times [1,T] \times \mathcal{X} \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \times [1,T] \times \mathbb{G}_1 \to \mathcal{Y}$ (represented by $\mathcal{F}_{\mathrm{RO}}$).

Key Generation. On input (KeyGen, sid), if such a request has already been received, then ignore the message. Otherwise, P generates a sequence of key pairs $((sk_1, vk_1), \ldots, (sk_T, vk_T))$ by initially sampling $sk_1 \xleftarrow{\$} \mathbb{Z}_p$ and computing $sk_{j+1} \leftarrow H_0(sk_j)$ and setting $vk_j \leftarrow g_2^{sk_j}$ for $j \in \{1, \ldots, T\}$. It then sets the counter $k_{ctr} \leftarrow 1$, stores sk_1 locally as the secret key, securely erases $sk_2, \ldots, sk_T$ and outputs $(\mathsf{VerKey}, sid, vk)$ where $vk = (vk_1, \ldots, vk_T)$.

Eval and Prove. On input $(\mathsf{EvalProve}, sid, vk, \bar{m}, j)$, P first checks that $\bar{m} \in \mathcal{X}^\ell$ for some $\ell \in \mathbb{N}$ and $k_{ctr} \leq j \leq T$. P then executes these steps until $k_{ctr} = j$: (1) Compute $sk_{k_{ctr}+1} \leftarrow H_0(sk_{k_{ctr}})$ (2) Securely erase $sk_{k_{ctr}}$ (3) Set $k_{ctr} \leftarrow k_{ctr} + 1$. It then computes $h_i \leftarrow H_1(vk|j|m_i)$ and $\pi_i \leftarrow h_i^{sk_j}$ for $i \in \{1, \ldots, \ell\}$ and the VRF output is derived by $y_i = H_2(vk|j|\pi_i) \in \mathcal{Y}$. Finally, P computes $sk_{k_{ctr}+1} \leftarrow H_0(sk_{k_{ctr}})$, securely erases $sk_{k_{ctr}}$, sets $k_{ctr} \leftarrow k_{ctr} + 1$ and outputs $(\mathsf{Evaluated}, sid, j, \bar{y}, \bar{\pi})$, where $\bar{y} = (y_1, \ldots, y_\ell)$ and $\bar{\pi} = (\pi_1, \ldots, \pi_\ell)$.

Verification. On input $(\mathsf{Verify}, sid, j, m, y, \pi, vk)$: Start by parsing vk as $(vk_1, \ldots, vk_T)$ and evaluate $h \leftarrow H_1(vk|j|m)$. Then, verify that: (1) $1 \leq j \leq T$; (2) $H_2(vk|j|\pi) = y$; and (3) $e(\pi, g_2) = e(h, vk_j)$. If all checks pass, then set $f \leftarrow 1$. Otherwise, set $f \leftarrow 0$. Finally, output $(\mathsf{Verified}, sid, vk, j, m, y, \pi, f)$.

Aggregation. Upon receiving $(\mathsf{Aggregate}, sid, (vk_1; j_1; m_1; y_1; \pi_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell; \pi_\ell))$, output $(\mathsf{Aggregated}, sid, \pi_{\mathsf{Agg}})$ where $\pi_{\mathsf{Agg}} = \left(\prod_{i=1}^{\ell} \pi_i, \pi_1, \ldots, \pi_\ell\right)$. Notice that outputs y_i can be represented by π_i as $y_i = H_2(vk_i|j_i|\pi_i)$, so π_{Agg} represents all (π_i, y_i) pairs.

AggVerification. Upon receiving $(\mathsf{AggVerify}, sid, (vk_1; j_1; m_1; y_1'), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell'), \pi_{\mathsf{Agg}})$, parse $vk_i = vk_{i,1}, \ldots, vk_{i,n}$ and set $f = 1$ if and only if:

$$e(\pi_{\mathsf{Agg}}, g_2) = \prod_{i=1}^{\ell} e(H_1(vk_i|j_i|m_i), vk_{i,j_i}), H_2(vk_i|j_i|\pi_i) = y_i', \forall\, i \in \{1, \ldots, \ell\}$$

Output $(\mathsf{AggVerified}, sid, (vk_1; j_1; m_1; y_1'), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell'), \pi_{\mathsf{Agg}}, f)$.

Fig. 3. Concrete protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$.

respectively, and T_{H_0}, T_{H_1}, T_{H_2}, T_{VRF} meant for bookkeeping with respect to random oracles and mirroring the insides of the functionality.

Upon receiving messages from $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$

– On $(\mathsf{KeyGen}, sid, P_i)$: Choose a random $sk_1 \leftarrow \mathbb{Z}_p$ and set $vk_1 = g_2^{sk_1}$. Obtain the key pairs $((sk_1, vk_1), \ldots, (sk_T, vk_T))$ in the same way as in the protocol but by repeatedly invoking the emulated random oracle H_0. Denote the keys $(sk, vk) \leftarrow ((sk_1, \ldots, sk_T), (vk_1, \ldots, vk_T))$. If $\exists j : T_{pk}[j] = (\cdot, vk)$, then abort. Otherwise, set $T_{pk}[i] \leftarrow (P_i, vk)$ and $T_{sk} \leftarrow (P_i, sk)$. Finally, return $(\mathsf{VerKey}, sid, P_i, vk)$ to $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$.

– On $(\mathsf{EvalProve}, sid, vk, j, \bar{m})$: Get the entry (P_i, vk) of the honest P_i from T_{pk} and parse $\bar{m}$ as $(m_1, \ldots, m_\ell)$, then run the following four-step loop $\forall i \in \{1, \ldots, \ell\}$:
  1. If there exists $(vk, j, m_i, \cdot, S) \in T_{\mathsf{VRF}}$ with $S \neq \emptyset$, then set $\pi_i \leftarrow \pi$ where[8] $\pi \in S$ and break. Otherwise, move to next step.
  2. Invoke the emulated random oracle $h_i \leftarrow \mathsf{H}_1(vk|j|m_i)$ and set $\pi_i \leftarrow h_i^{sk_j}$.
  3. Check that π_i is unique by checking that no other tuple $(vk', j', m', \cdot, S')$ exists in T_{VRF} such that $\pi \in S'$. If this is not the case, then abort.
  4. If $(vk, j, m_i, \cdot, \cdot) \notin T_{\mathsf{VRF}}$, then insert $(vk, j, m_i, \bot, \{\pi_i\})$ into T_{VRF}.

  Return $(\mathsf{EvalProve}, sid, vk, j, \bar{m}, \bar{\pi})$ with $\bar{m} = (m_1, \ldots, m_\ell)$ and $\bar{\pi} = (\pi_1, \ldots, \pi_\ell)$.
– On $(\mathsf{Verify}, sid, vk, j, m, y, \pi, L_{eval})$: First parse $vk = (vk_1, \ldots, vk_T)$, then
  1. Invoke the emulated random oracle $h \leftarrow \mathsf{H}_1(vk|j|m)$ and $\pi' \leftarrow h^{sk_j}$;
  2. Verify the proof by checking if $\pi = \pi'$ and that $e(\pi, g_2) = e(h, vk_j)$. If the checks goes through set $f_\pi \leftarrow 1$. Otherwise, set $f_\pi \leftarrow 0$;
  3. If $f_\pi = 0$ then return $(\mathsf{Verified}, sid, vk, j, m, 0)$ to $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and break;
  4. If $(vk, j, m, \cdot, \cdot) \notin T_{\mathsf{VRF}}$ but $T_{\mathsf{H}_2}[vk, j, \pi] \neq \bot$, then abort. If $(vk, j, m, \cdot, \cdot) \in T_{\mathsf{VRF}}$ then invoke the emulated random oracle and $y' \leftarrow \mathsf{H}_2(vk|j|\pi)$;
  5. If $y = y'$ then retrieve the record $(vk, j, m, y, S) \in T_{\mathsf{VRF}}$ and update the entry $(vk, j, m, y, S \cup \{\pi\})$ and return $(\mathsf{Verified}, sid, vk, j, m, 1)$. Otherwise, return $(\mathsf{Verified}, sid, vk, j, m, 0)$.
– On $(\mathsf{Aggregate}, sid, (vk_1; j_1; m_1; y_1; \pi_1), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell; \pi_\ell))$: Compute the aggregated proof $\pi_{\mathsf{Agg}} = \prod_{i=1}^{\ell} \pi_i$. Return $(\mathsf{Aggregated}, sid, \pi_{\mathsf{Agg}})$.
– On $(\mathsf{AggVerify}, sid, (vk_1; j_1; m_1; y_1'), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell'), \pi_{\mathsf{Agg}}, L_{eval})$: Parse the key $vk_i = vk_{i,1}, \ldots, vk_{i,n}$ and set $\phi = 1$ if and only if the following holds:

$$e(\pi_{\mathsf{Agg}}, g_2) = \prod_{i=1}^{\ell} e(H_1(j_i|m_i), vk_{i,j_i}),\ H_2(vk_i|j_i|m_i) = y_i', \forall\, i \in \{1, \ldots, \ell\}.$$

  Return $(\mathsf{AggVerified}, sid, (vk_1; j_1; m_1; y_1'), \ldots, (vk_\ell; j_\ell; m_\ell; y_\ell'), \phi)$.

Interacting with Random Oracles

– Emulating H_0: On input $s \in \mathbb{Z}_p$. If $T_{\mathsf{H}_0}[s] \neq \bot$ then return $T_{\mathsf{H}_0}[s]$. Otherwise, sample uniformly random $r \in \mathbb{Z}_p$ and set $T_{\mathsf{H}_0}[s] \leftarrow r$. Finally, return $T_{\mathsf{H}_0}[s]$.
– Emulating H_1: On input $t \in \{0,1\}^\ell$. If $T_{\mathsf{H}_1}[t] \neq \bot$ then return $T_{\mathsf{H}_1}[t]$. Otherwise, sample a group element $r \leftarrow \mathbb{G}_1$ uniformly at random and set $T_{\mathsf{H}_1}[t] \leftarrow r$. Finally, return $T_{\mathsf{H}_1}[t]$.
– Emulating H_2: On input $u \in \mathbb{G}_1$ do the following:
  1. If this is an internal invocation from the simulator the list of honest evaluations L_{eval} is provided by the functionality as an argument in Verify. Otherwise, the same list can be retrieved by calling $\mathsf{PastEval}$;

[8] Even if $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ supports multiple proofs per (vk, j, m)-tuple, the simulator can safely assume $|S| \leq 1$. This is due to the uniqueness of the underlying BLS signatures.

2. Identify the set Z of (vk, j, m)-tuples that give rise to the group element u. More concretely, first define a function $\mathsf{key\text{-}seq}(vk)$ as

$$\mathsf{key\text{-}seq}(vk) = \{(sk_j)_{j\in\{1,\ldots,T\}} \mid T_{\mathsf{H}_0}[sk_{j-1}] = sk_j \wedge g^{sk_j} = vk_j\}.$$

That is, obtain the vector of keys $(sk_1, \ldots, sk_T)$ extracted from T_{H_0} such that $g^{sk_j} = vk_j$. Then, given $u = h \wedge h^{sk_j}$, identify the set

$$Z_{vk} = \{(vk, j, m) \mid \mathsf{key\text{-}seq}(vk) = (sk_1, \ldots, sk_T) \wedge T_{\mathsf{H}_1}[vk|j|m] = u\};$$

3. If $Z_{vk} = \emptyset$:
Check if $T_{\mathsf{H}_2}[vk|j|u] \neq \perp$. If so, set $y \leftarrow T_{\mathsf{H}_2}[vk|j|u]$. Otherwise, assign y to a uniformly random element in $\mathcal{Y}$;
4. If $Z_{vk} = \{(vk, j, m)\} \wedge (vk, j, m, \cdot, \cdot) \in T_{\mathsf{VRF}}$:
If there is already an entry $(vk, j, m, y', \cdot) \in T_{\mathsf{VRF}}$ then we simply use $y \leftarrow y'$. Otherwise, the entry is of the form $(vk, j, m, \perp, \cdot)$ and we can identify the tuple $(m, j, \tilde{y}) \in L_{eval}$, update $(vk, j, m, \perp, \cdot)$ to $(vk, j, m, \tilde{y}, \cdot)$ and set $y \leftarrow \tilde{y}$;
5. If $Z_{vk} = \{(vk, j, m)\} \wedge (vk, j, m, \cdot, \cdot) \notin T_{\mathsf{VRF}}$:
Send $(\mathsf{Eval}, sid, vk, j, m)$ to $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and receive the $(\mathsf{Evaluated}, sid, y')$. Set $y \leftarrow y'$ and insert $(vk, j, m, y, \emptyset)$ into T_{VRF}.

Finally, set $T_{\mathsf{H}_2}[u] \leftarrow y$ and respond with y.

Upon corruption of party P_i. Upon corruption of P_i, the simulator first identifies all honest evaluations under P_i's keys:

$$O_{vk} = \left\{ (vk, j, m, y) : \begin{array}{c} \mathsf{key\text{-}seq}(vk) = (sk_1, \ldots, sk_T) \wedge \\ T_{\mathsf{H}_1}[vk|j|m] = h \quad \wedge \\ h^{sk_j} = u \quad \wedge \\ T_{\mathsf{H}_2}[vk|j|u] = y \end{array} \right\}.$$

Then, it requests the list of honest evaluations L_{eval} from the functionality and check if any entries from the functionality need to be programmed into H_2. That is, for each tuple in L_{eval} with (j, m, y), check that a corresponding tuple (vk, j, m, y) exists in O_{vk}. If not, then program the oracle by setting $T_{\mathsf{H}_2}[vk|j|u] = y$. Otherwise, do nothing. Then, it marks P_i as corrupted in $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and provides the adversary the key sk_{j+1} where j represents the maximum of the j-values found in T_{VRF} (the time period of the most recent evaluation).

Analysis of the simulator. We analyse the simulator in Appendix C.

4 Applications to Succinct Proof-of-Stake Blockchains

We start by reviewing the Ouroboros Praos block design [16]. We show how A-KE-VRFs can be used to achieve smaller block headers in Ouroboros Praos. Later, we outline further concrete optimizations that only apply to our construction $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$, *i.e.*, requiring non-black-box modifications. Finally, we discuss how A-KE-VRFs be used to obtain more efficient Proofs of Proof-of-Stake [20, 2].

A Regular VRF based PoS Block Design. In PoS blockchain protocols such as Ouroboros Praos [16], VRFs are used for two purposes: 1. Leader Election, where a VRF output/proof is used to verify that a party is selected to generate a block; 2. Random Beacon, where a VRF output/proof is used to generate randomness for leader election in the next epoch. Moreover, a KE signature is used for block authentication, *i.e.*, ensuring that a block is created by an elected leader in a way that an adaptive adversary cannot forge a conflicting block after corrupting the leader. By using the A-KE-VRF it is possible to aggregate the two VRF proofs. Moreover, it is also possible to substitute the KE signature on the block with a third VRF proof.

Ouroboros Praos Block Design Review. A block B for a slot number sl containing data d is produced by a party $\mathcal{P}$ by publishing the tuple $(\mathsf{st}, d, \mathsf{sl}, crt, \rho, \sigma)$, where st is the hash of the previous blocks. The tuple $crt = (\mathcal{P}, y, \pi)$ contains the party's identity $\mathcal{P}$, a VRF output y and a vrf proof π. The pair $\rho = (y_\rho, \pi_\rho)$ contains a VRF output y_ρ and a VRF proof π_ρ. Finally, σ is a KE signature on the value $(\mathsf{st}, d, \mathsf{sl}, crt, \rho)$ for the time slot sl, generated with the signing key for a particular point in time specified by sl. Concretely, in [16], the KE signature scheme [23] is used, while the VRF is instantiated by a new UC-secure unbiasable construction. Lastly, the design [16] is introduced in the hybrid world execution. Namely, it is described with ideal functionalities for the VRF and KE signature instances.

4.1 Succinct Block Headers: Concrete Optimizations

Now we describe optimization techniques to our construction $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ by detailing the following four approaches.

1. Generic VRF output extension via [3];
2. Non-black-box output extension for $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$;
3. Heuristic construction with higher adversarial bias: deriving VRF outputs from aggregated proof in $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$;
4. Compressing public keys into Merkle Trees.

Generic VRF output extension via [3]. We argue that our construction is compatible to the extension technique in [3]. Note that the core idea in [3] is that a k-tuple of VRF outputs is generated by performing $y_i \leftarrow H(i||y)$ for $1 \leq i \leq k$ given an arbitrary proof/value VRF pair (y, π). We observe that our construction, defined in Section 3 as $y \leftarrow H_2(vk||j||\pi)$ and $\pi \leftarrow h^{sk_j}$ for $h \leftarrow H_1(vk||j||m)$ and slot j, is suitable to the same derivation technique for $(y_1, \ldots, y_k)$. That is $y_i = H_3(i||H_2(vk||j||\pi))$, given an extra hash function H_3. We point out that in our construction we derive new outputs by relying on a single proof π. This feature is used to decrease the size of the block as it is described next. The final block design, as presented in [3], is $(\mathsf{st}, d, \mathsf{sl}, crt, \rho, \sigma)$, with $crt = (\mathcal{P}, y, \pi)$, $\rho = (y_\rho, \pi_\rho)$, and the signature σ generated by the KE signature scheme [23].

Musen: Non-black-box output extension for $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$. Although our notion of A-KE-VRF is compatible with the generic output extension of [3], however our

concrete construction can be modified for more efficient output extension. We leverage the generation of the VRF output values by a series of hash values from the original proof π, instead of the original VRF output y. We stress that this change offers a smaller block, given that it is possible to generate VRF values y and y_ρ given the single proof π. Concretely, $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$, described in Figure 3, can be modified to compute independent pseudorandom VRF outputs $y_i = H_3(vk|i|\pi) \in \mathcal{Y}$ for $i \in \{1, \ldots, \ell\}$ and a random oracle instance $H_3()$. Table 1 highlights the size differences. Our construction requires storing only the proof π, therefore y and y_ρ can be derived for validation. Furthermore, our design uses the VRF construction to compute a signature π_σ. Finally, the block design is $(\mathsf{st}, d, \mathsf{sl}, crt^*, \rho^*, \sigma^*)$, such that $crt^* = \mathcal{P}$, $\rho^* = \pi$, and $\sigma^* = \pi_\sigma$, which is a VRF proof used as a signature.

Heuristic Musen: Heuristic construction with higher adversarial bias. Our previous construction contains two VRF proofs, *i.e.*, π and π_σ, and the latter is used as a signature. Given the existing aggregation procedure, it is a natural direction to explore the possibility of aggregating both proofs into a new proof denoted π_{agg}, *i.e.*, the aggregated proof of π and π_σ, as it would lead to an even smaller block header. In such a construction, the VRF output values are generated by computing $y_i = H_3(i|vk|j|\pi_{agg}) \in \mathcal{Y}$ for $i \in \{1, \ldots, \ell\}$ as before. The final block structure is $(\mathsf{st}, d, \mathsf{sl}, crt^*, \rho^{**}, \sigma^*)$, such that $crt^* = \mathcal{P}$, $\rho^{**} = \cdot$, and $\sigma^{**} = \pi_{agg}$. This results in a smaller header, as illustrated in Table 1.

Note that π_σ is generated with the new block content as the input for the VRF. We stress that the data is arbitrarily chosen by a corrupted participant which can choose multiple versions of the block content in order to pick the most favorable one among several options of π_σ. The adversary's goal is to generate a VRF output y that increases the chances of a corrupted party being selected as the block leader. We remark that although the adversary cannot arbitrarily choose the value of y (given it is an output of a random oracle), by having multiple choices to choose from, it introduces a bias in the distribution of the value which translates in an edge in being chosen as slot leader.

We conjecture that such bias is manageable and could be proven secure, albeit with more conservative set of parameters which may hinder the overall protocol efficiency. The study of the new and secure set of parameters is out of the scope of this work, and left as a suggestion for future work.

Compressing public keys. We now focus our attention to the size of the public key. We recall that in our construction vk is the tuple $(vk_1, \ldots, vk_T)$ for the time slot $1 \leq j \leq T$. We now detail a Merkle Tree based optimization to make a small key version. For completeness, assume the Merkle Tree is three algorithms, $(\mathsf{M.gen}, \mathsf{M.prove}, \mathsf{M.ver})$. Given the public key vk, then $\mathsf{M.gen}(vk) \rightarrow \mathsf{M}_{root}$, thus $\mathsf{M.prove}$ outputs the membership proof M_j, that is $\mathsf{M.prove}(\mathsf{M}_{root}, vk_j) \rightarrow \mathsf{M}_j$. The verification is performed as $\mathsf{M.ver}(\mathsf{M}_{root}, \mathsf{M}_j, vk_j) \rightarrow 1$. In order to decrease the size of vk, it is necessary to set $vk' \leftarrow \mathsf{M}_{root}$ and $\pi' \leftarrow (\pi, \mathsf{M}_j, vk_j)$. Thus, the VRF proof π' in (j, m, y, π', vk') is verified by computing $h \leftarrow H_1(\mathsf{M}_{root}|j|m)$,

Protocol \ Types	Block	Proofs	Size (in bits)
Ouroboros Praos [16]	$(\mathsf{st}, d, \mathsf{sl}, crt, \rho, \sigma)$	$crt = (\mathcal{P}, y, \pi)$ $\rho = (y_\rho, \pi_\rho)$ $\sigma = \sigma$	$\lvert\mathcal{P}\rvert$+926 926 3584
Badertscher *et al.* [3]	$(\mathsf{st}, d, \mathsf{sl}, crt, \rho', \sigma)$	$crt = (\mathcal{P}, y, \pi)$ $\rho' = y_\rho$ $\sigma = \sigma$	$\lvert\mathcal{P}\rvert$+926 463 3584
MUSEN [THIS WORK]	$(\mathsf{st}, d, \mathsf{sl}, crt^*, \rho^*, \sigma^*)$	$crt^* = \mathcal{P}$ $\rho^* = \pi$ $\sigma^* = \pi_\sigma$	$\lvert\mathcal{P}\rvert$ 463 463
HEURISTIC MUSEN [THIS WORK]	$(\mathsf{st}, d, \mathsf{sl}, crt^*, \rho^{**}, \sigma^{**})$	$crt^* = \mathcal{P}$ $\rho^{**} = \cdot$ $\sigma^{**} = \pi_{agg}$	$\lvert\mathcal{P}\rvert$ 0 463

Table 1. Note $(\mathsf{st}, d, \mathsf{sl})$ is fixed, however the block proofs (crt, ρ, σ) differ. We remark that the Musen versions of the block header does not contain a single VRF output value as they can be derived from the single proof π in a non-black-box access to our VRF construction $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$. We assume that: (1) one pairing group element to be 463 bits as well as the output of hash function, (2) the label $\mathcal{P}$ has fixed $|\mathcal{P}|$ bits of length, and (3) KE signature is 448 bytes long according to Cardano Specification [15]. However, the size of a pairing group element can be as small as 382 bits for the BLS12-381 curve.

and checking the following three equalities: (1) $\mathsf{M.ver}(\mathsf{M}_{root}, \mathsf{M}_j, vk_j) = 1$, (2) $y = H_2(\mathsf{M}_{root}|j|\pi)$; and (3) $e(\pi, g_2) = e(h, vk_j)$.

This technique shortens the public key vk size to only the root of the Merkle Tree, a significant advantage from the much longer $(vk_1, \ldots, vk_T)$. We increased the size of the original proof π with two extra values, namely (M_j, vk_j). However, the trade-off between vk and vk' may be worth the size increase of the proof.

4.2 Better Proofs of Proof-of-Stake

The construction of Proofs of Proof-of-Stake is an important aspect in the Proof-of-Stake ecosystem to allow efficient sidechains and superlight clients for Proof-of-Stake protocols [20, 2]. The first construction was presented by Gazi et al. [20] and built a scheme based on Ad-Hoc Threshold Multisignatures (ATMS), which can themselves be built based on regular digital signatures, aggregate signatures, STARKs or bulletproofs - with different tradeoffs among the constructions in terms of computational/communication complexities and required assumptions.

A key idea is that for every epoch of the Proof-of-Stake protocol there is a synchronization committee that is responsible for signing the state commitment of that epoch. Moreover, there is handover information that will determine the committee members for the next epoch (the specifics of how committee members are sampled depends on the particular Proof-of-Stake protocol being used, please check [20, 2] for details), and the committee members of the current epoch sign the handover information, thus allowing the committee of one epoch to inaugurate the committee representing the next epoch. The state commitment/handover messages are accepted as long as there are signatures from the majority of the committee members for that epoch. And it is assumed that for

every epoch of the Proof-of-Stake protocol there is a honest majority in the committee, so that this recurrent process of signing handover information and state commitment can work without security problems.

The recent work of Agrawal et al. [2] make improvements by building a scheme based on Merkle Trees that achieves sublinear communication and computational complexities based on the assumption that there are some provers that are always online in the chain and available to help the verifiers, and that at least one of those provers is honest and will provide the correct information.

The main idea is that each prover presents a succinct representation of the sequence of committees, denoted a handover tree, by hashing the information of each epoch's committee and then inserting the hashes into a Merkle tree whose root is sent to the verifier. In case that no divergence exists in the roots provided by the provers, one of the provers can be used to get the full information about the current committee members together with a proof that its hash is the last element of the handover tree. And this information can be checked by the verifier in order to guarantee the correctness of the information about the current committee members. And from there, the signature of the members of this committee can be used to verify the state commitment (also represented using Merkle tree to get an efficient and secure representation). Using our techniques it is possible to aggregate the individual signatures of committee members on the state commitment (while keeping tracking of who signed it); and in cases where a VRF is used to determine the committee membership (i.e. Ouroboros Praos [16]) signatures and VRF proofs that are needed to verify membership can be aggregated as described in previous sections.

In case there is any divergence in the Merkle tree roots provided by the different provers, the verifier will use an elimination tournament and play disagreeing parties against each other in matches until all remaining provers are in agreement. In each match of this tournament the verifier recursively asks two disagreeing provers to recursively provide information about the children of the handover tree until the first point of disagreement is located. This point of this disagreemnt is then checked: both provers are asked to reveal the committee members of that epoch and the previous one, as well as the signatures on the handover information by the committee members from the previous epoch. This information, together with the handover tree, can then used to eliminate at least one dishonest prover involved in that match, and no honest prover is ever eliminated. Please check Agrawal et al. [2] for details. Also here our techniques can be used to aggregate the signatures on the handover information, and in cases where a VRF is used to determine the committee membership to aggregate both the signatures and VRF proofs that are necessary for the checks.

5 Forward Secure Encryption to the Future

In order to argue about forward security for EtF and AfP, we first recall that semantic security for EtF (Definitions 10 and 11) and unforgeability for AfP (Definition 12) as introduced in [8] are defined with respect to a lottery predicate

(Definition 9). The lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) \in \{0,1\}$ lets a party who has lottery witness $\mathsf{sk}_{L,i}$ check whether it was the winner for role P in slot sl according to a blockchain $\mathbf{B}$, *i.e.* the party selected to play that role at that slot by the intrinsic leader election mechanism of a PoS blockchain protocol. A sender can generate an EtF ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m)$ containing message m encrypted towards the winner of role P in slot sl according to an initial blockchain $\mathbf{B}$. A party whose lottery witness $\mathsf{sk}_{L,i}$ satisfies $1 \leftarrow \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$ with respect to a future blockchain $\tilde{\mathbf{B}}$ evolved from $\mathbf{B}$[9] can use $\mathsf{sk}_{L,i}$ to decrypt ct and obtain message $m \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk}_{L,i})$. Intuitively, semantic security for an EtF ciphertext for role P and slot sl holds against adversaries whose lottery witness $\mathsf{sk}_{L,A}$ is such that $0 \leftarrow \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,A})$ (*i.e.* an adversary who did not win role P at slot P with respect to a valid evolution $\tilde{\mathbf{B}}$ of the initial blockchain $\mathbf{B}$ cannot learn anything except the length of m by seeing ct). Analogously, unforgeability of AfP signatures on behalf of role P at slot sl with respect to blockchain $\tilde{\mathbf{B}}$ holds against adversaries who did not win role P at slot sl of a blockchain evolution $\tilde{\mathbf{B}}$ of $\mathbf{B}$ (*i.e.* the adversary cannot sign on behalf of role P of slot sl). Notice that both security guarantees hinge on the fact that the adversary does not know a witness $\mathsf{sk}_{L,i}$ that satisfies $1 \leftarrow \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$ with respect to a future blockchain $\tilde{\mathbf{B}}$ evolved from $\mathbf{B}$ for the role P, slot sl and initial blockchain $\mathbf{B}$ used as parameters for the EtF ciphertext.

Defining Forward Security for EtF and AfP. We focus on defining and constructing a notion of witness evolving lottery predicates that allow parties to evolve their witness so that obtaining past versions of the witness from its current versions is computationally hard. We do not modify Definition 9 of a lottery predicate but rather define an extra witness evolution algorithm to be used in tandem with the original lottery predicate. Later on, this will be naturally leveraged to ensure that previous EtF ciphertexts generated for roles for which a party has been selected cannot be decrypted using the current version of the witness. Similarly, AfP tags cannot be generated on behalf of roles for which the party has been selected in the past using current versions of the party's witness.

Recall that, according to the model of blockchain protocol execution for EtF and AfP presented in [8] and summarized in Appendix D, each party P_i is represented by a pair $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$ associated with public data $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$. In the original formulation of lottery mechanisms from [8] a static witness $\mathsf{sk}_{L,i}$ is used by each party throughout the entire execution. Now, we take advantage of the slot numbers sl already considered in the original model to define a notion of evolving witness where every slot sl is related to an unpredictable slot-specific witness $\mathsf{sk}_{L,i,\mathsf{sl}}$ that can be obtained from the previous slot's witness $\mathsf{sk}_{L,i,\mathsf{sl}-1}$. Notice that the sl parameter in the subscript of our generalized lottery scheme is already taken as input by the lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) \in \{0,1\}$

[9] The security model for EtF considers blockchain protocols for which it is possible to non-interactively check whether a blockchain $\tilde{\mathbf{B}}$ has evolved from an initial blockchain $\mathbf{B}$ via an honest execution of the underlying blockchain protocol. See Appendix D for details.

and used as a parameter for EtF and AfP algorithms, so it is not necessary to modify the syntax and security definitions for these component. While we keep the lottery predicate itself unchanged, we will add a witness update algorithm $\mathsf{sk}_{L,i,\mathsf{sl}+1} \leftarrow \mathsf{Update}(\mathsf{sk}_{L,i,\mathsf{sl}})$ that takes as input a lottery witness $\mathsf{sk}_{L,i,\mathsf{sl}}$ for slot sl and outputs a new witness $\mathsf{sk}_{L,i,\mathsf{sl}+1}$ for slot $\mathsf{sl}+1$. While the witness evolves $\mathsf{sk}_{L,i,\mathsf{sl}}$, the signature key $\mathsf{Sig.sk}_i$ and the public data $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$ are static, so no modification to the underlying blockchain protocol is necessary apart from having parties invoke the witness update algorithm for every slot, so they obtain their updated witness.

Definition 1 (Witness Evolving Lottery Predicate). *A witness evolving lottery predicate is a pair of PPT functions* $(\mathsf{lottery}, \mathsf{Update})$ *defined as follows:*

- $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) \in \{0,1\}$ *takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* P *and a lottery witness* $\mathsf{sk}_{L,i}$ *and outputs 1 if and only if the party owning* $\mathsf{sk}_{L,i}$ *won the lottery for the role* P *in slot* sl *with respect to the blockchain* $\mathbf{B}$*.*
- $\mathsf{sk}_{L,i,\mathsf{sl}+1} \leftarrow \mathsf{Update}(\mathsf{sk}_{L,i,\mathsf{sl}})$ *takes as input a lottery witness* $\mathsf{sk}_{L,i,\mathsf{sl}}$ *and outputs a fresh* unpredictable *lottery witness* $\mathsf{sk}_{L,i,\mathsf{sl}+1}$ *for slot* $\mathsf{sl}+1$*.*

  Security. *An adversary* $\mathcal{A}$ *who is given a blockchain* $\mathbf{B}$ *that is at a slot* sl *in the context of a blockchain protocol* Γ^V *as defined in Appendix D.1 and any lottery witness* $\mathsf{sk}_{L,i,\mathsf{sl}}$ *can only find a lottery witness* $\mathsf{sk}_{L,i,\mathsf{sl}'}$ *for a slot* $\mathsf{sl}' < \mathsf{sl}$ *that satisfies* $1 \leftarrow \mathsf{lottery}(\mathbf{B}, \mathsf{sl}', \mathsf{P}, \mathsf{sk}_{L,i,\mathsf{sl}'})$ *for any role* P *with probability negligible in* κ *for a computational security parameter* κ*.*

Forward security for EtF and AfP with Witness Evolving Lottery Predicate and Secure Erasures. It is straightforward to show that any EtF or AfP scheme instantiated with a witness evolving lottery predicate achieves forward security given secure erasures. Indeed, the semantic security definition for EtF (Definition 11) and the unforgeability definition for AfP (Definition 12) prevent any adversary who does not know a witness $\mathsf{sk}_{L,i}$ such that $1 \leftarrow \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$ from breaking these security guarantees. When we instantiate such schemes with a witness evolving lottery predicate, we can use the $\mathsf{sk}_{L,i,\mathsf{sl}+1} \leftarrow \mathsf{Update}(\mathsf{sk}_{L,i,\mathsf{sl}})$ algorithm to obtain a new lottery witness $\mathsf{sk}_{L,i,\mathsf{sl}+1}$ for slot $\mathsf{sl}+1$ from the current witness $\mathsf{sk}_{L,i,\mathsf{sl}}$ and then securely erase $\mathsf{sk}_{L,i,\mathsf{sl}}$. Given the security of the witness evolving lottery scheme (Definition 1), it is infeasible for an adversary who obtains $\mathsf{sk}_{L,i,\mathsf{sl}}$ to learn any witness $\mathsf{sk}_{L,i,\mathsf{sl}'}$ for slot $\mathsf{sl}' < \mathsf{sl}$ such that $1 \leftarrow \mathsf{lottery}(\mathbf{B}, \mathsf{sl}', \mathsf{P}, \mathsf{sk}_{L,i,\mathsf{sl}'})$ for any role P except with negligible probability. On the other hand, an adversary who breaks semantic security for EtF or unforgeability for AfP with respect to a slot $\mathsf{sl}' < \mathsf{sl}$ when only given access to a lottery witness $\mathsf{sk}_{L,i,\mathsf{sl}}$ must be able to learn a witness $\mathsf{sk}_{L,i,\mathsf{sl}'}$ such that $1 \leftarrow \mathsf{lottery}(\mathbf{B}, \mathsf{sl}', \mathsf{P}, \mathsf{sk}_{L,i,\mathsf{sl}'})$[10], which would break the security of the witness evolving lottery predicate. Hence, any EtF or AfP scheme instantiated with a witness evolving lottery predicate achieves forward security given

[10] Formally, this requires witnesses to be extractable given the blockchain $\mathbf{B}$ and the EtF ciphertexts, which is achieved by the cWE construction of near future EtF from [8] recalled in Appendix E

secure erasures. A concrete instantiation can be obtained via the constructions [8] (recalled in Appendix E), which can be instantiated for any lottery predicate.

Construction of Witness Evolving Lottery Predicate Our main construction is a direct application of our notion of A-KE-VRF, leveraging the key evolving property to construct a witness evolving lottery predicate. The main idea of this construction is to instantiate the lottery predicate based on oblivious leader selection via VRFs from Ouroboros Praos [16] (recalled in Appendix E) using our A-KE-VRF scheme instead of a standard VRF scheme. In this lottery predicate, the lottery witness is the VRF secret key iteself. Hence, since we can evolve the VRF secret key, we can directly derive a witness evolving lottery predicate. Given that our A-KE-VRF has the same properties as the Ouroboros Praos VRF (plus the key evolution), it fits into this lottery construction in a straightforward manner. Moreover, since we only require forward security from the VRF (and not aggregation) to obtain this property, the forward secure VRF from [17] could also be used to construct this lottery predicate and instantiate forward secure EtF and AfP following the same approach.

Forward Security for the EtF construction of [8]. If we depart from the EtF construction of from [8] based on Witness Encryption of Commitments (cWE) recalled in Appendix E.3, we can also get forward security by erasing the randomness used for the commitments to witnesses for the cWE scheme. The key observation is that a party may only decrypt a cWE ciphertext if it knows both the witness $\mathsf{sk}_{L,i}$ inside its commitment $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$ and the randomness ρ_i used to generate cm_i. Since this EtF construction from [8] is essentially a direct application of a cWE, where each party publishes commitments cm_i to their witnesses $\mathsf{sk}_{L,i}$ so that an encryptor can create cWE ciphertexts containing the desired messages towards each cm_i for the relation that checks whether $1 \leftarrow \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$. Later on, if a party is selected for a given role at a certain slot with respect to the blockchain state, they can decrypt the cWE ciphertext using $\mathsf{sk}_{L,i}$ and ρ_i. Hence, one can adapt this specific construction as follows: post commitment $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$, retrieve/decrypt EtF ciphertext from the blockchain and decrypt using $\mathsf{sk}_{L,i}, \rho$, securely erase ρ_i, post new commitment $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i')$ using a fresh randomness ρ_i'. This would prevent an adversary from decrypting past ciphertexts upon corruption, since cWE decryption requires both $\mathsf{sk}_{L,i}$ *and* ρ_i. To obtain forward security for an AfP scheme associated to this EtF scheme (described in Appendix E.4), we further modify the AfP construction of [8] to require that the signer proves knowledge not only of $\mathsf{sk}_{L,i}$ but also of ρ_i for the cm used to generate the EtF ciphertext.

References

1. Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages

116–129. Springer, Heidelberg, December 2000.
2. Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. Proofs of proof-of-stake with sublinear complexity. Cryptology ePrint Archive, Report 2022/1642, 2022. https://eprint.iacr.org/2022/1642.
3. Christian Badertscher, Peter Gazi, Iñigo Querejeta-Azurmendi, and Alexander Russell. A composable security treatment of ECVRF and batch verifications. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part III*, volume 13556 of *LNCS*, pages 22–41. Springer, Heidelberg, September 2022.
4. Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.
5. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.
6. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.
7. Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129. Springer, Heidelberg, November / December 2011.
8. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 151–180. Springer, Heidelberg, December 2022.
9. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
10. Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pages 219–233. IEEE, 2004.
11. Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 651–680. Springer, Heidelberg, December 2022.
12. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.
13. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
14. Coindesk. Cardano Network Developers Increase Block Size by 10%. https://www.coindesk.com/tech/2022/04/27/cardano-network-developers-increase-block-size-by-10/, 2022. [Online; accessed 20-September-2023].
15. Jared Corduan, Polina Vinogradova, and Matthias Gudemann. A Formal Specification of the Cardano Ledger. https://github.com/input-output-hk/cardano-ledger/releases/latest/download/shelley-ledger.pdf, 2019. [Online; accessed 20-September-2023].

16. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.
17. Muhammed F. Esgin, Oguzhan Ersoy, Veronika Kuchta, Julian Loss, Amin Sakzad, Ron Steinfeld, Wayne Yang, and Raymond K. Zhao. A new look at blockchain leader election: Simple, efficient, sustainable and post-quantum. Cryptology ePrint Archive, Report 2022/993, 2022. `https://eprint.iacr.org/2022/993`.
18. Nils Fleischhacker, Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Jackpot: Non-interactive aggregatable lotteries. Cryptology ePrint Archive, Paper 2023/1570, 2023. `https://eprint.iacr.org/2023/1570`.
19. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.
20. Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy*, pages 139–156. IEEE Computer Society Press, May 2019.
21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.
22. Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Heidelberg, November 2017.
23. Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, August 2001.
24. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.
25. Di Ma and Gene Tsudik. Extended abstract: Forward-secure sequential aggregate authentication. In *2007 IEEE Symposium on Security and Privacy*, pages 86–91. IEEE Computer Society Press, May 2007.
26. Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. Cryptology ePrint Archive, Report 2007/052, 2007. `https://eprint.iacr.org/2007/052`.
27. Bitcoin Magazine. What is the Bitcoin block size limit? `https://bitcoinmagazine.com/guides/what-is-the-bitcoin-block-size-limit`, 2022. [Online; accessed 20-September-2023].
28. Giulio Malavolta. Key-homomorphic and aggregate verifiable random functions. Cryptology ePrint Archive, Paper 2024/643, 2024. `https://eprint.iacr.org/2024/643`.
29. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.
30. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

A UC Security Framework

We work in the Universal Composability framework of [9], to which we refer interested readers for further details. In this security framework a protocol execution is represented by a group of ITIs called the *main machines*. These machines constitute the protocol together with two additional ITIs called the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

The main machines executing the protocol are associated with a protocol session id (SID) and each of the machines can be uniquely identified within a protocol session by a so-called party id (PID). A protocol execution is subject to the rules set by an appropriate *control function*. In particular, during an execution the environment $\mathcal{Z}$ orchestrates the machines only by providing inputs and receiving the outputs.

The adversary $\mathcal{A}$ is allowed to communicate using so-called backdoor tapes with other ITIs in the same session. Here, a corruption by $\mathcal{A}$ can be modeled merely as the ITI receiving a special corruption message from $\mathcal{A}$ and subsequently acting in an appropriate manner dictated by the protocol.

Consider an environment $\mathcal{Z}$ with input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ where λ denotes the security parameter. $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ denotes the probability distribution ensemble $\{\mathrm{EXEC}_{\diamond,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$ and represents the binary outputs of the environments with initial input being some polynomial function of the security parameter.

Analogously let $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the probability distribution ensemble $\{\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(z)\}_{z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$. When Π is randomized, indistinguishability is guaranteed even when the distribution of outputs of Π are viewed jointly with the output of $\mathcal{A}$ (and similarly for $\mathcal{F}$ and $\mathcal{S}$). We say a protocol Π UC-realizes a functionality $\mathcal{F}$ when for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$, running in polynomial time in λ, it holds that $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$.

Setup Assumptions Our construction of A-KE-VRF is in the random oracle model, which is formalized in the UC framework as $\mathcal{F}_{\mathrm{RO}}$-hybrid model, *i.e.*, where parties have access to functionality $\mathcal{F}_{\mathrm{RO}}$ as setup.

Functionality $\mathcal{F}_{\mathrm{RO}}$

Functionality $\mathcal{F}_{\mathrm{RO}}$ interacts with parties $P_1, \ldots, P_n$ and ideal adversary $\mathcal{S}$. $\mathcal{F}_{\mathrm{RO}}$ is parameterized by an image set $\mathcal{Y}$ and maintains an initially empty list H.

Query. Upon receiving a message $(\textsc{Query}, sid, m)$, $\mathcal{F}_{\mathrm{RO}}$ proceeds as follows:

- If there is no pair (m, y) in H, sample $y \xleftarrow{\$} \mathcal{Y}$ and add (m, y) to H.
- Retrieve the pair (m, y) from H and return $(\textsc{Response}, sid, y)$.

Fig. 4. Random Oracle Functionality $\mathcal{F}_{\mathrm{RO}}$

B Game based Definitions of VRFs and relations to UC A-KE-VRF

We recall standard game based definitions of VRFs and argue that our UC notion of A-KE-VRF implies these game based notions.

Definition 2 (Verifiable Random Function [29]). *Let* $\mathsf{ParamGen}$,KeyGen, Eval, Verify *be polynomial-time algorithms with the following description:*
$\mathsf{ParamGen}(1^\lambda)$: *On input a security parameter* 1^λ*, the algorithm outputs global, public parameters pp.*
$\mathsf{KeyGen}(pp)$: *On input public parameters pp the algorithm outputs a pair* (pk, sk) *named a public and secret key, respectively.*
$\mathsf{Eval}(sk, x)$: *On input a secret key sk and input* $x \in \mathcal{X}$*, the algorithm outputs a pair* (y, π) *where* $y \in \mathcal{Y}$ *is the VRF evaluation value and* π *is the proof of correct evaluation of* y*.*
$\mathsf{Verify}(pk, y, x, \pi)$: *On input public key pk, VRF evaluation value* y*, input* x *and proof of correct evaluation* π*, the algorithm outputs a bit* $b \in \{0, 1\}$*.*
Provability *For any pair* $(y, \pi) \leftarrow \mathsf{Eval}(sk, x)$ *such that* $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ *and* $pp \leftarrow \mathsf{ParamGen}(1^\lambda)$*, it holds that* $\mathsf{Verify}(pk, y, x, \pi)$ *outputs* 1*.*
Pseudorandomness *Let* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be a polynomial-time adversary. A VRF scheme satifies pseudorandomness if* $\mathcal{A}$ *when engaged in the experiment* $\mathsf{Exp\text{-}PRand}$ *(described below) wins the game with probability* $\frac{1}{2} + \mathsf{negl}(\lambda)$*.*

The standard definition of VRF does not take into consideraiton the notion of key evolution and forward security. Although the scheme in [17] satisfies a notion of forward security, it is tailor made for a specific blockchain protocol, which hinders our goal of providing a general construction. Hence, we present an informal definition of key evolving VRF.

Definition 3 (Key Evolving VRF (Informal)). *A VRF scheme (Definition 2) satisfying the following additional key-evolving property is called an forward secure VRF. This scheme is define by adding a key update algorithm* $\mathsf{sk}_{i+1} \leftarrow \mathsf{Update}(\mathsf{sk}_i)$ *that takes as input a VRF secret key* sk_i *and outputs an* unpredictable *new secret key* sk_{i+1}*. In a key evolving VRF, an adversary* $\mathcal{A}$ *who has a key* sk_i *cannot forge a proof* π *or obtain a VRF output* y *with respect to a previous key* $\mathsf{sk}_{i'}$ *such that* $i' < i$ *for any input* x *that has not been previously evaluated under secret key* $\mathsf{sk}_{i'}$ *except with negligible probability.*

Given our UC notion of A-KE-VRF, it is easy to see that any scheme that UC-realizes $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$. The argument is that any adversary that breaks the definitions above, can be used in a black box way to construct an environment that distinguishes a real world execution of the VRF scheme from an ideal world execution of $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$. Indeed, an adversary that succeeds in forging a VRF proof, can be used by an environment to construct such a forgery for an arbitrary secret key sk and input x, then forcing the ideal world execution to evaluate the VRF on x (where this evaluation would be simulated), identifying the forgery. The same argument goes for pseudorandomness, since an adversary who break

the pseudorandomness property can be used by the environment to generate a non-pseudorandom output y for a given input x, which can be used to distinguish the real world execution from the ideal world execution when the same input x is evaluated in the ideal world (which would provide a pseudorandom output). As for forward security, an adversary breaking forward security can be used to generate an output y for an input x that has not been previously evaluated under a secret key sk, which again helps distinguishing executions when the environment instructs the ideal execution to evaluate the VRF on input x (which would not be possible).

C Security Analysis of A-KE-VRF Construction (Remainder of proof of Theorem 1)

We show that the simulator $\mathcal{S}$ described in Section 3.1 interacts with the functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and an internal copy of $\mathcal{A}$, producing towards the environment a transcript that is indistinguishable from that of the real protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ interacting with the adversary $\mathcal{A}$.

Analysis of Simulator. We recall the two probability ensembles, namely $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}},\mathcal{S},\mathcal{Z}}$, representing the environment $\mathcal{Z}$ interacting with the functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and the simulator $\mathcal{S}$ (*ideal world*), and $\mathrm{EXEC}_{\Pi_{\mathsf{A\text{-}KE\text{-}VRF}},\mathcal{A},\mathcal{Z}}$ where the environment interacts with the adversary $\mathcal{A}$ in the context of the real protocol $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ (*real world*). In the following, we argue that the two ensembles are, in fact, indistinguishable as long as the simulator does not abort. To that end, we describe the inputs that $\mathcal{Z}$ can provide as well as the resulting view of $\mathcal{Z}$ in the real and ideal world, respectively.

Key Generation. When an honest evaluator $P \in \mathcal{P}$ is instructed by the environment to do a key generation the functionality first check if the party P is already registered with a key and otherwise such a key is generated by the simulator. From the view of the environment, the outputs are identically distributed since vk is produced in the same manner by the simulator (in the ideal world) and the honest party (in the real world). The simulator's attempt to program the random oracle $\mathsf{H_0}$ at one of the values $sk_1, \ldots, sk_T$ may fail because an entry, $T_{\mathsf{H_0}}[sk_j]$ for some $j \in \{1, \ldots, T\}$, has already been recorded. However, this happens only if the adversary can predict the value of sk_1 sampled by the simulator or if a collision happens when producing the hash sequence $sk_1, \ldots, sk_T$.

Eval and Prove. During evaluation the environment provides as input a vector of messages to be evaluated $\bar{m} \in \mathcal{X}^\ell$ under a verification key vk in period j. In both the real and ideal world the same verification is done to ensure that j is in the right range. Now, every m_i for $i \in \{1, \ldots, \ell\}$ is treated the same both in the ideal and real world so (w.l.o.g.) consider an evaluation of a single message $m = m_i$ (if the adversary can distinguish for some m_i, it can distinguish for all $\bar{m}$). If no proof has been computed for a tuple (vk, j, m), then the new proof

π is computed exactly the same way by the honest evaluator in the protocol and by the simulator, respectively. The only possible failure is when a collision happens. That is, the same proof π has already been produced for a different tuple (vk', j', m'). This happens with negligible probability. If the tuple (vk, j, m) has already been evaluated, π is found in T_{VRF} by the simulator resulting in the same real world distribution. One exception is the case where the proofs set S has size $|S| > 1$. This can lead to ambiguity as to which proof in S the simulator should return to faithfully simulate the real world execution. Fortunately, this is not a problem since $\Pi_{\mathsf{A\text{-}KE\text{-}VRF}}$ deals with unique proofs (BLS-signatures), thus the simulator always has only a single proof in S to choose from.

Finally, the output values y are distributed uniformly at random in $\mathcal{Y}$ in both worlds. In the real world this is ensured by the random oracle and in the ideal world this is sampled by the functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$. If an honest party asks for an evaluation on (vk, j, m) (on behalf of the environment), the simulator will not know the value y that was sampled inside $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and received by the environment along with the proof π. The environment can now instruct the adversary $\mathcal{A}$ to query the tuple by invoking the random oracle $y' \leftarrow \mathsf{H}_2(vk|j|\pi)$ and check if $y = y'$. Or, alternatively, the environment can provide input $(\mathsf{Verify}, sid, vk, j, m, y, \pi)$ and check if the tuple correctly verifies and thus outputs $f = 1$. If this "strategy" succeeds, the environment will obviously be able to distinguish. To avoid such a scenario, the emulation of H_2 ensures consistency between T_{H_2} and $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$. In particular, the simulator obtains the list L_{eval} from the $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ and makes sure that T_{H_2} is programmed and consistent with VRF outputs generated inside the functionality.

Verification. Consider the party V that provides the tuple (vk, j, m, y, π) for verification. We define a function $\mathsf{Valid} : \mathbb{G}_2 \times \{0,1\}^\ell \times [T] \times \mathcal{Y} \times \mathbb{G}_1 \rightarrow \{\mathsf{true}, \mathsf{false}\}$ that takes the contents of the above tuple as input and returns true if and only if: $1 \leq j \leq T$, $H_2(vk|j|\pi) = y$ and $e(\pi, g_2) = e(h, vk_j)$ where H_2 has different connotation depending on the (real/ideal) context. We also define failure events such that F_{col1} and F_{col2} refers to a collision happening in H_1 and H_2, respectively. And $F_{\mathcal{O}}$ refers to the (unfortunate) sampling of the identity element from $\mathbb{G}_1$ by H_1. Now, the input/output distribution of the Verify interface in the real world can be described precisely. Assume that f is the output of the Verify interface and (vk, j, m, y, π) defines the input, then conditioned on F_{col1}, F_{col2} and $F_{\mathcal{O}}$ *not* happening the relationship $f = 1 \iff \mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{true}$ holds.

From now we argue that the same relationship holds when the environment is interacting with the simulator $\mathcal{S}$ and functionality $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$.

(a) *Correctness.* When the functionality receives a tuple which is already defined in its internal database M and where the key vk is such that $(\cdot, vk, \cdot) \in \mathcal{PK}$, then it immediately responds with $f = 1$. Since the proof π is present in S, the evaluator was honest and the proof was generated by the simulator during $\mathsf{EvalProve}$. Thus, it satisfies $\mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{true}$

(b) *Unforgeability.* If the functionality receives a tuple with $vk = vk'$ where $(P, vk', \cdot) \in \mathcal{PK}$ and P being honest but with a proof π that is not to be found in any set S in M, then the functionality immediately rejects

$f = 0$. Since $vk = vk'$, the only way that $\mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{true}$ is if the environment can produce a valid π without knowledge of the secret key sk_j and thus breaking the EUF-CMA property of BLS signatures which we assume happens with negligible probability.

(c) *Consistency.* This follows from the fact that if $\pi \in S$ then we may assume that it is an honest evaluation that satisfies $\mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{true}$ and if $\pi \in Q$ then the proof was deemed invalid during a forgery "attack".

(d) *Simulator's Decision.* In this case the functionality sends the tuple (vk, j, m, y, π) along with L_{eval}. The simulator now decides the value of f and it uses the exact same algorithm as the real-world protocol to decide $f = 0$ so by construction $\mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{false}$. What remains is to identify the probability that the simulator will abort if $\mathsf{Valid}(vk, j, m, y, \pi) = \mathsf{true}$. This happens when $(vk, j, m, \cdot, \cdot) \notin T_{\mathsf{VRF}}$ but $T_{\mathsf{H}_2}[vk|j|\pi] \neq \perp$. In other words, H_2 is already programmed on a (valid) entry defined by the tuple (vk, j, m) but no entry is found in T_{VRF}.

Aggregation and Aggregate Verification. The analysis follows similarly from the standard verification simulation. Notice that $\mathcal{S}$ responds to queries from $\mathcal{F}_{\mathsf{A\text{-}KE\text{-}VRF}}$ for aggregation and aggregated verification by executing the same protocol steps as an honest party in the real world protocol. Hence, the simulation is indistinguishable unless $\mathcal{A}$ finds a collision for one of the random oracles or forges an aggregate BLS signature. Since we have already established that collisions can only be found with negligible probability and the BLS scheme is proven secure, it follows that $\mathcal{S}$'s behavior in these procedures is computationally indistinguishable from the real world execution with $\mathcal{A}$.

Corruption of P_i. An honest party $P_i \in \mathcal{P}$ in the real world performs secure erasures between evaluations. Thus, we can assume that upon corruption the adversary is only given the key $sk_{i,j+1}$ (where j represents the time period of the latest evaluation). In the ideal world, the simulator retrieves $sk_{i,j+1}$ using its local tables T_{sk} and T_{H_0}. Since the simulator has computed the keys in exactly the same way as the protocol, the view of the adversary upon corrupting the evaluator is indistinguishable (ignoring RO collisions). Finally, since the y-values sampled by the functionality (from honest evaluations) are added to T_{H_2} *before* handing over the control of the honest party to the adversary, then the state of the random oracle is consistent with the view of the adversary upon corruption.

This concludes our argument and establishes Theorem 1. □

D Defining Encryption to the Future (EtF) and Authentication from the Past (AfP)

In this appendix, we repeat the summary of the model for Encryption to the Future (EtF) and Authentication from the Past (AfP) from [8] as presented in [11] in almost verbatim form.

D.1 The Blockchain Model

We use the model for Encryption to the Future (EtF) from [8], which defines this primitive with respect to a blockchain ledger that has an built-in lottery mechanism. Before presenting the definition of EtF and related concepts, we recall the model for blockchain ledgers from [22], which is used to state the definitions of [8] and that captures properties of natural Proof-of-Stake (PoS) based protocols such as [16]. In this section, we give an overview of the framework from [22] for arguing about PoS blockchain protocol security as presented in [8].

Blockchain Structure. A genesis block $B_0 = ((\mathsf{Sig.pk}_1, \mathsf{aux}_1, \mathsf{stake}_1), \ldots, (\mathsf{Sig.pk}_n, \mathsf{aux}_n, \mathsf{stake}_n), \mathsf{aux})$ associates each party P_i to a signature scheme public key $\mathsf{Sig.pk}_i$, an amount of stake stake_i and auxiliary information aux_i (*i.e.* any other relevant information required by the blockchain protocol). As in [16], we assume that the genesis block is generated by an initialization functionality $\mathcal{F}_{\mathsf{INIT}}$ that registers all parties' $\mathsf{Sig.pk}_i, \mathsf{aux}_i$ when the execution starts and assigns stake_i for P_i. Within the execution model of [22], $\mathcal{F}_{\mathsf{INIT}}$ is executed by the environment. A blockchain $\mathbf{B}$ relative to a genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots $\mathsf{sl}_1, \ldots, \mathsf{sl}_m$ such that $B_i = (\mathsf{sl}_j, H(B_{i-1}), \mathsf{d}, \mathsf{aux})$, where sl_j indicates the time slot that B_i occupies, $H(B_{i-1})$ is a collision resistant hash of the previous block, d is data and aux is auxiliary information required by the blockchain protocol (*e.g.* a proof that the block is valid for slot sl_j). We denote by $\mathbf{B}^{\lceil \ell}$ the chain (sequence of blocks) $\mathbf{B}$ where the last ℓ blocks have been removed and if $\ell \geq |\mathbf{B}|$ then $\mathbf{B}^{\lceil \ell} = \epsilon$. Also, if $\mathbf{B}_1$ is a prefix of $\mathbf{B}_2$ we write $\mathbf{B}_1 \preceq \mathbf{B}_2$. For the sake of simplicity, we identify each party P_i participating in the protocol by its public key $\mathsf{Sig.pk}_i$.

Blockchain Protocol Execution. Let the blockchain protocol

$$\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$$

be guarded by a validity predicate V. The algorithms can be described as follows:

- $\mathsf{UpdateState}(1^\lambda) \rightarrow \mathsf{bst}$ where bst is the local state of the blockchain along with metadata.
- $\mathsf{GetRecords}(1^\lambda, \mathsf{bst}) \rightarrow \mathbf{B}$ outputs the longest sequence $\mathbf{B}$ of valid blocks (wrt. V).
- $\mathsf{Broadcast}(1^\lambda, m)$ Broadcast the message m over the network to all parties executing the blockchain protocol.

An execution of a blockchain protocol Γ^V proceeds by participants running the algorithm $\mathsf{UpdateState}^V$ to get the latest blockchain state, $\mathsf{GetRecords}$ to extract the ledger data structure from a state and $\mathsf{Broadcast}$ to distribute messages which are added to the blockchain if accepted by V. An execution is orchestrated by an environment $\mathcal{Z}$ which classifies parties as either honest or corrupt. All honest parties executes $\Gamma^V(1^\lambda)$ with empty local state bst and all corrupted parties are controlled by the adversary $\mathcal{A}$ who also controls network including delivery of messages between all parties.

– In each round all honest parties receive a message m from $\mathcal{Z}$ and potentially receive incoming network messages delivered by $\mathcal{A}$. The honest parties may do computation, broadcast messages and/or update their local states.
– $\mathcal{A}$ is responsible for delivering all messages sent by honest parties to all other parties. $\mathcal{A}$ cannot modify messages from honest parties but may delay and reorder messages on the network.
– At any point $\mathcal{Z}$ can communicate with adversary $\mathcal{A}$ or use $\mathsf{GetRecords}$ to retrieve a view of the local state of any party participating in the protocol.

The result is a random variable $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ denoting the joint view of all parties (i.e. all inputs, random coins and messages received) in the above execution. Note that the joint view of all parties fully determines the execution. We define the view of the adversary as $\mathsf{view}_{\mathcal{A}}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$ and the view of the party P_i as $\mathsf{view}_{P_i}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$. If it is clear from the context which execution the argument is referring to, then we just write view_i. We assume that it is possible to take a snapshot *i.e.*, a view of the protocol after the first r rounds have been executed. We denote that by $\mathsf{view}^r \leftarrow \mathsf{EXEC}_r^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$. Furthermore, we can resume the execution departing from this view and continue until round $\tilde{r}$ resulting in the full view including round $\tilde{r}$ denoted by $\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma^V}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$.

We let the function $\mathsf{stake}_i = \mathsf{stake}(\mathbf{B}, i)$ take as input a local blockchain $\mathbf{B}$ and a party P_i and output a number representing the stake of party P_i wrt. to blockchain $\mathbf{B}$. Let the sum of stake controlled by the adversary be $\mathsf{stake}_{\mathcal{A}}(\mathbf{B})$, the total stake held by all parties $\mathsf{stake}_{\mathsf{total}}(\mathbf{B})$ and the adversaries relative stake is $\mathsf{stake\text{-}ratio}_{\mathcal{A}}(\mathbf{B})$. We also consider the PoS-fraction $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell)$ as the amount of unique stake whose proof is provided in the last ℓ mined blocks. More precisely, let $\mathcal{M}$ be the index i corresponding to miners P_i of the last ℓ blocks in $\mathbf{B}$ then

$$\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell) = \frac{\sum_{i \in \mathcal{M}} \mathsf{stake}(\mathbf{B}, i)}{\mathsf{stake}_{\mathsf{total}}}.$$

A note on corruption For simplicity in the above execution we restrict the environment to only allow static corruption while the execution described in [30] supports adaptive corruption with erasures.

A note on admissible environments [30] specifies a set of restrictions on $\mathcal{A}$ and $\mathcal{Z}$ such that only compliant executions are considered and argues that certain security properties holds with overwhelming probability for these executions. An example of such a restriction is that $\mathcal{A}$ should deliver network messages to honest parties within Δ rounds.

Blockchain Properties. In coming sections we will define what it means to encrypt to a future state of the blockchain. First, we need to ensure what it means for a blockchain execution to have evolved from one state to another. We recall that running a protocol Γ^V with appropriate restrictions on $\mathcal{A}$ and

$\mathcal{Z}$ will yield certain compliant executions $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ where some security properties will hold with overwhelming probability. An array of prior works, including [19, 30], have converged towards a few security properties that characterizes blockchain protocols. These include *Common Prefix* or *Chain Consistency*, *Chain Quality* and *Chain Growth*. From these basic properties, a number of stronger properties were derived in [22]. Among them, is the *Distinguishable Forking* property which will be the main requirement when introducing the EtF scheme.

Definition 4 (Common Prefix). *Let $\kappa \in \mathbb{N}$ be the common prefix parameter. The chains $\mathbf{B}_1$, $\mathbf{B}_2$ possessed by two honest parties P_1 and P_2 in slots $\mathsf{sl}_1 < \mathsf{sl}_2$ satisfy $\mathbf{B}_1^{\lceil\kappa} \preceq \mathbf{B}_2$.*

Definition 5 (Chain Growth). *Let $\tau \in (0, 1]$, $s \in$ $mathbbN$ and let $\mathbf{B}_1$, $\mathbf{B}_2$ be as above with the additional restriction that $\mathsf{sl}_1 + s \leq \mathsf{sl}_2$. Then $\mathsf{len}(\mathbf{B}_2) - \mathsf{len}(\mathbf{B}_1) \geq \tau s$ where τ is the speed coefficient.*

Definition 6 (Chain Quality). *Let $\mu \in (0, 1]$ and $\kappa \in \mathbb{N}$. Consider any set of consecutive blocks of length at least κ from an honest party's chain $\mathbf{B}_1$. The ratio of adversarial blocks in the set is $1 - \mu$ where μ is the quality coefficient.*

Definition 7 (Distinguishable Forking). *A blockchain protocol Γ satisfies $(\alpha(\cdot), \beta(\cdot), \ell_1(\cdot), \ell_2(\cdot))$-distinguishable forking property with adversary $\mathcal{A}$ in environment $\mathcal{Z}$, if there exists negligible functions, negl$(\cdot)$, $\delta(\cdot)$ such that for every $\lambda \in \mathbb{N}, \ell \geq \ell_1(\lambda), \tilde{\ell} \geq \ell_2(\lambda)$ it holds that*

$$\Pr\left[\begin{array}{c} \alpha(\lambda) + \delta(\lambda) < \beta(\lambda) \wedge \\ \mathsf{suf\text{-}stake\text{-}contr}^{\tilde{\ell}}(\mathsf{view}, \beta(\lambda)) = 1 \wedge \\ \mathsf{bd\text{-}stake\text{-}fork}^{(\ell,\tilde{\ell})}(\mathsf{view}, \alpha(\lambda) + \delta(\lambda)) = 1 \end{array} \middle| \mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^\lambda) \right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

Evolving Blockchains. In an EtF scheme, the future is defined with respect to a future state of the underlying blockchain. In particular, we want to make sure that the initial chain $\mathbf{B}$ has "correctly" evolved into the final chain $\tilde{\mathbf{B}}$. Otherwise, the adversary can easily simulate a blockchain where it wins a future lottery and finds itself with the ability to decrypt. Fortunately, the *Distinguishable Forking* property from [22] allows us to distinguish a sufficiently long chain in an honest execution from a fork generated by the adversary by looking at the combined amount of stake proven in such a sequence of blocks. This property is used to construct a predicate called $\mathsf{evolved}(\cdot, \cdot)$. First, let $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with validity predicate V and where the $(\alpha, \beta, \ell_1, \ell_2)$-*distinguishable forking* property holds. And let $\mathbf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$ and $\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.

Definition 8 (Evolved Predicate). *An evolved predicate is a polynomial time function* evolved *that takes as input blockchains $\mathbf{B}$ and $\tilde{\mathbf{B}}$*

$$\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) \in \{0,1\}.$$

It outputs 1 if and only if $\mathbf{B} = \tilde{\mathbf{B}}$ *or the following holds (i)* $V(\mathbf{B}) = V(\tilde{\mathbf{B}}) = 1$*; (ii)* $\mathbf{B}$ *and* $\tilde{\mathbf{B}}$ *are consistent i.e.* $\mathbf{B}^{\lceil\kappa} \preceq \tilde{\mathbf{B}}$ *where* κ *is the common prefix parameter; (iii) Let* $\ell' = |\tilde{\mathbf{B}}| - |\mathbf{B}|$ *then it holds that* $\ell' \geq \ell_1 + \ell_2$ *and* $\mathsf{u\text{-}stakefrac}(\tilde{\mathbf{B}}, \ell' - \ell_1) > \beta$*.*

Blockchain Lotteries. The vast majority of PoS-based blockchain protocols has an inbuilt lottery scheme for selecting parties to generate blocks. In this lottery any party can win the right to generate a block for a certain slot with a probability proportional to its relative stake in the system. In the model from [8], a party can decrypt an EtF ciphertext if it wins this lottery. It can be useful to conduct multiple independent lotteries for the same slot sl, which is associated to a set of roles $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Depending on the lottery mechanism, each pair $(\mathsf{sl}, \mathsf{P}_i)$ may yield zero, one or multiple winners. A party with access to the blockchain can locally determine whether it is the lottery winner for a given role by executing a procedure using its lottery witness $\mathsf{sk}_{L,i}$ related to $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$, which may also give the party a proof of winning for others to verify. The definition below from [8] details what it means for a party to win a lottery.

Definition 9 (Lottery Predicate). *A lottery predicate is a polynomial time function* $\mathsf{lottery}$ *that takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* P *and a lottery witness* $\mathsf{sk}_{L,i}$ *and outputs 1 if and only if the party owning* $\mathsf{sk}_{L,i}$ *won the lottery for the role* P *in slot* sl *with respect to the blockchain* $\mathbf{B}$*.*
Formally, we write $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) \in \{0,1\}$*.*

It is natural to establish the set of lottery winning keys $\mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{P}}$ for parameters $(\mathbf{B}, \mathsf{sl}, \mathsf{P})$. This is the set of eligible keys satisfying the $\mathsf{lottery}$ predicate.

D.2 Modelling EtF

We are now ready to present the model of [8] for encryption to the future winner of a lottery (*i.e.* EtF). The blocks of an underlying blockchain ledger and their relative positions in the chain are used to specify points in time. Intuitively, this notion allows for creating ciphertexts that can only be decrypted by a party that is selected to perform a certain role R at a future slot sl according to a lottery scheme associated with a blockchain protocol (*i.e.* a party that has a lottery secret key $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$).

Definition 10 (Encryption to the Future). *A pair of PPT algorithms* $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ *in the the context of a blockchain* Γ^V *is an EtF-scheme with evolved predicate* $\mathsf{evolved}$ *and a lottery predicate* $\mathsf{lottery}$*. The algorithms work as follows*

Encryption. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m)$ *takes as input an initial blockchain* $\mathbf{B}$*, a slot* sl*, a role* P *and a message* m*. It outputs a ciphertext* ct *- an encryption to the future.*

Decryption. $m/\perp \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk})$ *takes as input a blockchain state* $\tilde{\mathbf{B}}$*, a ciphertext* ct *and a secret key* sk *and outputs the original message* m *or* $\perp$*.*

Correctness. An EtF-scheme is said to be correct if for honest parties i and j, there exists a negligible function μ such that

$$\left| \Pr \left[\begin{array}{l} \mathsf{view} \leftarrow \mathit{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m) \\ \mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1 \\ \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}) = 1 \end{array} : \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk}) = m \right] - 1 \right| \leq \mu(\lambda).$$

Security. Security is defined with a game $\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Algorithm 1, where a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ execute an underlying blockchain protocol with an environment $\mathcal{Z}$ as described in Appendix D.1. In this game, $\mathcal{A}$ chooses a blockchain $\mathbf{B}$, a role P for the slot sl and two messages m_0 and m_1 and sends it all to $\mathcal{C}$, who chooses a random bit b and encrypts the message m_b with the parameters it received and sends ct to $\mathcal{A}$. $\mathcal{A}$ continues to execute the blockchain until an evolved blockchain $\tilde{\mathbf{B}}$ is obtained and outputs a bit b'. If the adversary is a lottery winner for the challenge role P in slot sl, the game outputs a random bit. If the adversary is not a lottery winner for the challenge role P in slot sl, the game outputs $b \oplus b'$. The reason for outputting a random guess in the game when the challenge role is corrupted is as follows. Normally the output of the IND-CPA game is $b \oplus b'$ and we require it to be 1 with probability 1/2. This models that the guess b' is independent of b. This, of course, cannot be the case when the challenge role is corrupted. We therefore output a random guess in these cases. After this, any bias of the output away from 1/2 still comes from b' being dependent on b.

Algorithm 1 $\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ $\mathcal{A}$ executes Γ with $\mathcal{Z}$ until round r
$(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}})$ ▷ $\mathcal{A}$ outputs challenge parameters
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m_b)$
$\mathsf{st} \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}}, \mathsf{ct})$ ▷ $\mathcal{A}$ receives challenge ct
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ Execute from view^r until round $\tilde{r}$
$(\tilde{\mathbf{B}}, b') \leftarrow \mathcal{A}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}}, \mathsf{st})$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ **then** ▷ $\tilde{\mathbf{B}}$ is a valid evolution of $\mathbf{B}$
  if $sk^{\mathcal{A}}_{L,j} \notin \mathcal{W}_{\tilde{\mathbf{B}},\mathsf{sl},\mathsf{P}}$ **then** ▷ $\mathcal{A}$ does not win role P
    return $b \oplus b'$
return $\hat{b} \xleftarrow{\$} \{0,1\}$

Definition 11 (IND-CPA Secure EtF). *An EtF-scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ in the context of a blockchain protocol Γ executed by PPT machines $\mathcal{A}$ and $\mathcal{Z}$ is said*

to be IND-CPA *secure if, for any* $\mathcal{A}$ *and* $\mathcal{Z}$*, there exists a negligible function* μ *such that for* $\kappa \in \mathbb{N}$*:*

$$\left|2 \cdot \Pr\left[\mathrm{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}} = 1\right] - 1\right| \leq \mu(\lambda).$$

ECW as a Special Case of EtF. In this work, we focus on a special class of EtF called ECW where the underlying lottery is always conducted with respect to the current blockchain state. This has the following consequences

1. $\mathbf{B} = \tilde{\mathbf{B}}$ means that $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ is trivially true.
2. The winner of role P in slot sl is already defined in $\mathbf{B}$.

Notice that in ECW there is no need for checking if the blockchain has 'correctly' evolved and all lottery parameters (*e.g.* stake distribution and randomness extracted from the blockchain) are static. Hence, when constructing an ECW scheme, the lottery winner is already decided at encryption time. While an ECW is simpler to realize than a more general EtF, it is shown in [8] that ECW can be used to instantiate YOSO MPC and then be transformed into EtF given an identity based encryption scheme.

D.3 Authentication from the Past (AfP)

When the winner of a role S sends a message m to a future role R then it is typically also needed that R can be sure that the message m came from a party P which, indeed, won the role S. This concept is formalized as an AfP scheme as follows.

Definition 12 (Authentication from the Past). *A pair of PPT algorithms* $\mathcal{U} = (\mathsf{Sign}, \mathsf{Ver})$ *is a scheme for authenticating messages as a winner of a lottery in the past in the context of blockchain* Γ *with lottery predicate* $\mathsf{lottery}$ *such that:*

Authenticate. $\sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}, m)$ *takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* S *and a message* m*. It outputs a signature* σ *that authenticates the message* m*.*

Verify. $\{0,1\} \leftarrow \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m)$ *uses the blockchain* $\tilde{\mathbf{B}}$ *to ensure that* σ *is a signature on* m *produced by the secret key winning the lottery for slot* sl *and role* S*.*

Furthermore, an AfP-scheme has the following properties:

Correctness.

$$\left|\Pr\left[\begin{array}{l} \mathsf{view} \leftarrow \mathit{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \quad \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \quad \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}, m) \\ \quad \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}) = 1 \\ \quad \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}) = 1 \end{array} : \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m) = 1\right] - 1\right| \leq \mu(\lambda).$$

In other words, an AfP on a message from an honest party with a view of the blockchain $\mathbf{B}$ *can attest to the fact that the sender won the role* S *in slot*

sl. *If another party, with blockchain* $\tilde{\mathbf{B}}$ *agrees, then the verification algorithm will output 1.*

Security. *The EUF-CMA game detailed in 2 is used to define the security of an AfP scheme. In this game, the adversary has access to a signing oracle* $\mathcal{O}_{\mathsf{AfP}}$ *which it can query with a slot* sl*, a role* S *and a message* m_i*, obtaining AfP signatures* $\sigma_i = \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_j, m_i)$ *where* $\mathsf{sk}_j \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{S}}$ *i.e.* $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_j) = 1$*. The oracle maintains the list of queries* $\mathcal{Q}_{\mathsf{AfP}}$*. Formally, an AfP-scheme* $\mathcal{U}$ *is said to be EUF-CMA secure in the context of a blockchain protocol* Γ *executed by PPT machines* $\mathcal{A}$ *and* $\mathcal{Z}$ *if there exists a negligible function* μ *such that for* $\kappa \in \mathbb{N}$*:*

$$\Pr\left[\mathrm{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}} = 1\right] \leq \mu(\lambda).$$

Algorithm 2 $\mathrm{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$

$\mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ $\mathcal{A}$ executes Γ with $\mathcal{Z}$
$(\mathbf{B}, \mathsf{sl}, \mathsf{S}, m', \sigma') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}_{\mathcal{A}})$
if $(m' \in \mathcal{Q}_{\mathsf{AfP}}) \vee (\mathsf{sk}^{\mathcal{A}}_{L,j} \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{S}})$ **then** ▷ $\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}$ won or queried illegal m'
  return 0
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^{r},\tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ Execute from view^{r} until round $\tilde{r}$
$\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(\mathsf{view}^{\tilde{r}}_{i})$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ **then**
  if $\mathsf{Ver}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \sigma', m') = 1$ **then** ▷ $\mathcal{A}$ successfully forged an AfP
    return 1
return 0

AfP Privacy. The specific privacy property we seek is that an adversary, observing AfP tags from honest parties, cannot use this information to enhance its chances in predicting the winners of lotteries for roles for which an AfP tag has not been published.

Definition 13 (AfP Privacy). *An AfP scheme* $\mathcal{U}$ *with corresponding lottery predicate* $\mathsf{lottery}$ *is private if a PPT adversary is unable to distinguish between the scenarios defined in 3 and 4 with more than negligible probablity in the security parameter.*

Scenario 0 ($b = 0$) *In this scenario (3) the adversary is first running the blockchain* Γ *together with the environment* $\mathcal{Z}$*. At round* r *the adversary is allowed to interact with the oracle* $\mathcal{O}_{\mathsf{AfP}}$ *as described in 12. The adversary then continues the execution until round* $\tilde{r}$ *where it ouputs a bit* b'*.*

Scenario 1 ($b = 1$) *This scenario (4) is identical to scenario 0 but instead of interacting with* $\mathcal{O}_{\mathsf{AfP}}$*, the adversary interacts with a simulator* $\mathcal{S}$*.*

Algorithm 3 $b = 0$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}_r^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
$\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}_{\mathcal{A}}^r)$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}_{\mathcal{A}}^{\tilde{r}})$

Algorithm 4 $b = 1$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}_r^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
$\mathcal{A}^{\mathcal{S}}(\mathsf{view}_{\mathcal{A}}^r)$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathcal{S}}(\mathsf{view}_{\mathcal{A}}^{\tilde{r}})$

We let $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ *denote the game where a coinflip decides whether the adversary is executed in scenario 0 or scenario 1. We say that the adversary wins the game (i.e.* $\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}} = 1$*) iff* $b' = b$*. Finally, an AfP scheme* $\mathcal{U}$ *is called private in the context of the blockchain* Γ *and underlying lottery predicate* $\mathsf{lottery}$ *if the following holds for a negligible function* μ*.*

$$\Pr\left[\mathrm{Game}^{\mathsf{ID\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}} = 1\right] \leq 1/2 + \mu(\lambda).$$

E Constructing EtF and AfP

In this appendix, we present the generic constructions of EtF and AfP introduced in [8] in almost verbatim form.

E.1 Building Blocks

First, we introduce the definitions of the main building blocks as presented in [8] in almost verbatim form.

Commitment Schemes. We recall the syntax for a commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ below:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ outputs a commitment key. The commitment key ck defines a message space $\mathcal{S}_m$ and a randomizer space $\mathcal{S}_r$.
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho) \to \mathsf{cm}$ outputs a commitment given as input a message $\mathsf{s} \in \mathcal{S}_m$ and randomness $\rho \in \mathcal{S}_r$.

We require a commitment scheme to satisfy the standard properties of *binding* and *hiding*. It is binding if no efficient adversary can come up with two pairs $(\mathsf{s}, \rho), (\mathsf{s}', \rho')$ such that $\mathsf{s} \neq \mathsf{s}'$ and $\mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho) = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}'; \rho')$ for $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$. The scheme is hiding if for any two $\mathsf{s}, \mathsf{s}' \in \mathcal{S}_m$, no efficient adversary can distinguish between a commitment of s and one of s'.

Extractability. In the construction of ECW from cWE shown in [8], the commitments must satisfy an additional property which allows to *extract* message and randomness of a commitment. In particular we assume that our setup outputs both a commitment key and a trapdoor td and that there exists an algorithm Ext such that $\mathsf{Ext}(\mathsf{td}, \mathsf{cm})$ outputs (s, ρ) such that $\mathsf{cm} = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$. We remark

we can generically obtain this property by attaching to the commitment a NIZK argument of knowledge that shows knowledge of opening, i.e., for the relation $\mathcal{R}^{\text{opn}}(\mathsf{cm}_i; (\mathsf{s}, \rho)) \iff \mathsf{cm}_i = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$.

Witness Encryption over Commitments (cWE). Here, we describe witness encryption over commitments (cWE), a relaxed notion of witness encryption (WE) introduced in [8], whose definition we present in almost verbatim form. In witness encryption parties encrypt to a public input for some NP statement. In cWE we have two phases: first parties provide a (honestly generated) commitment cm of their private input s. Later, anybody can encrypt to a public input for an NP statement which *also* guarantees correct opening of the commitment. Importantly, in applications, the first message in our model can be reused for many different invocations. As observed in [8], cWE is substantially weaker than WE and can be constructed based on standard assumptions via a generic construction from 2-round Oblivious Transfer and Garbled Circuits.

The type of relations we consider are of the following form: a statement $\mathsf{x} = (\mathsf{cm}, C, y)$ and a witness $\pi^{\mathsf{cw}} = (\mathsf{s}, \rho)$ are in the relation (i.e., $(\mathsf{x}, \pi^{\mathsf{cw}}) \in \mathcal{R}$) iff "$\mathsf{cm}$ commits to some secret value s using randomness ρ, and $C(\mathsf{s}) = y$". Here, C is a circuit in some circuit class $\mathcal{C}$ and y is the expected output of the function. Formally, we define witness encryption over commitments as follows:

Definition 14 (Witness encryption over commitments). *Let* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ *be a non-interactive commitment scheme. A cWE-scheme for witness encryption over commitments with circuit class* $\mathcal{C}$ *and commitment scheme* C *consists of a pair of algorithms* $\Pi_{cWE} = (\mathsf{Enc}, \mathsf{Dec})$*:*

Encryption phase. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, \mathsf{x}, m)$ *on input a commitment key* ck*, a statement* $\mathsf{x} = (\mathsf{cm}, C, y)$ *such that* $C \in \mathcal{C}$*, and a message* $m \in \{0,1\}^*$*, generates a ciphertext* ct*.*

Decryption phase. $m/\perp \leftarrow \mathsf{Dec}(\mathsf{ck}, \mathsf{ct}, \pi^{\mathsf{cw}})$ *on input a commitment key* ck*, a ciphertext* ct*, and a witness* π^{cw}*, returns a message* m *or* $\perp$*.*

A cWE should satisfy *correctness* and *semantic security* as defined below.

(Perfect) Correctness. An honest prover with a statement $\mathsf{x} = (\mathsf{cm}, C, y)$ and witness $\pi^{\mathsf{cw}} = (\mathsf{s}, \rho)$ such that $\mathsf{cm} = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$ and $C(\mathsf{s}) = y$ can always decrypt with overwhelming probability. More precisely, a cWE with circuit class $\mathcal{C}$ and commitment scheme C has perfect correctness if for all $\kappa \in \mathbb{N}$, $C \in \mathcal{C}$, $\mathsf{ck} \in \mathsf{Range}(\mathsf{C.Setup})$, $\mathsf{s} \in \mathcal{S}_m$, randomness $\rho \in \mathcal{S}_r$, commitment $\mathsf{cm} \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho)$, and bit message $m \in \{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, C(\mathsf{s})), m); m' \leftarrow \mathsf{Dec}(\mathsf{ck}, \mathsf{ct}, (\mathsf{s}, \rho)) \ : \ m = m'\right] = 1$$

(Weak) Semantic Security. Intuitively, encrypting with respect to a false statement (with honest commitment) produces indistinguishable ciphertexts. Formally, there exists a negligible function μ such that for all $\kappa \in \mathbb{N}$, all

auxiliary strings aux and all PPT adversaries $\mathcal{A}$:

$$\left| 2 \cdot \Pr \left[\begin{array}{r} \mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ (\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{aux}) \\ \mathsf{cm} \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho); b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, y), m_b) \\ \mathsf{ct} := \bot \text{ if} \\ C(\mathsf{s}) = y,\ C \notin \mathcal{C} \text{ or } |m_0| \neq |m_1| \end{array} : \mathcal{A}(\mathsf{st}, \mathsf{ct}) = b \right] - 1 \right| \leq \mu(\lambda).$$

Later, to show the construction of ECW from cWE, we need a stronger notion of semantic security where the adversary additionally gets to see ciphertexts of the challenge message under true statements with unknown to $\mathcal{A}$ witnesses. This property is formalized in [8], where it is shown that weak semantic security together with hiding of the commitment imply strong semantic security.

E.2 VRF-based Lottery

This section introduces a specific lottery mechanism that is specially interesting as it is based on VRFs and can thus be instantiated from our A-KE-VRF. The description is taken from [8] in almost verbatim form. The backbone of the lottery is a VRF scheme VRF as described in [16]. This VRF has the properties of simulatability and unpredictability under malicious key generation which will become useful when arguing about security of the AfP. The VRF scheme is a tuple $(\mathsf{VRF.Gen}, \mathsf{VRF.Prove}, \mathsf{VRF.Verify})$ where $\mathsf{VRF.Gen}(1^\kappa)$ outputs a pair of keys $(\mathsf{VRF.pk}, \mathsf{VRF.sk})$. The $\mathsf{VRF.Prove}$ takes as input a value x and outputs a pair $(y, \pi) \leftarrow \mathsf{VRF.Prove}_{\mathsf{VRF.sk}}(x)$ which is the output value y and the correctness certificate π. The verification is then done by evaluating $\mathsf{VRF.Verify}_{\mathsf{VRF.pk}}(x, y, \pi)$ which outputs 1 iff π attests to the correctness of y as the output of the VRF evaluated on x with key $\mathsf{VRF.sk}$. We recall the blockchain setup described in Appendix D.1 where each party P_i is represented by a pair $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$ associated with public data $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$. Let aux_i contain a VRF public key $\mathsf{VRF.pk}_i$ as described above and let the lottery secret key be $\mathsf{sk}_{L,i} = (\mathsf{Sig.pk}_i, \mathsf{VRF.sk}_i)$. Finally, we introduce a function $\mathsf{param}(\mathbf{B}, \mathsf{sl})$. This function outputs a tuple $(\{\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i\}_{i \in [n]}, \eta, \phi)$ associated with the specific blockchain $\mathbf{B}$ and slot sl. Beyond obtaining the public information $(\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i)$ the function also returns a nonce, η, as well as a public function $\phi(\cdot)$ which on input stake_i computes the threshold for winning the lottery. The lottery predicate based on the VRF is described in Algorithm 5.

E.3 ECW from cWE

In this section we show the construction ECW from cWE as presented in [8] in almost verbatim form. We define our scheme with respect to a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ executing a blockchain protocol Γ as described in Appendix D,

Algorithm 5 $\mathsf{lottery}_{\mathsf{VRF}}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,j})$

$(\{\mathsf{Sig.pk}_i, \mathsf{VRF}_{L,i}, \mathsf{stake}_i\}_{i\in[n]}, \eta, \phi) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$
$(\mathsf{Sig.pk}_j, \mathsf{VRF.sk}_j) \leftarrow \mathsf{sk}_{L,j}$
$(y, \pi) \leftarrow \mathsf{VRF.Prove}_{\mathsf{sk}_{L,j}}(\mathsf{sl}||\mathsf{P}||\eta)$
if $y < \phi(\mathsf{stake}_j)$ **then**
  if $\mathsf{VRF.Verify}_{\mathsf{vk}_{L,j}}(\mathsf{sl}||\mathsf{P}||\eta, y, \pi) =$ **then**
    return 1
return 0

i.e. each party P_i has access to the blockchain ledger and is associated to a tuple $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{st}_i)$ registered in the genesis block for which it has corresponding secret keys $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$. Our construction uses as a main building block a witness encryption scheme over commitments $\Pi_{cWE} = (\mathsf{Enc}_{cWE}, \mathsf{Dec}_{cWE})$; we assume the commitments to be extractable. The class of circuits $\mathcal{C}$ of Π_{cWE} includes the lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$. We let each party publish an initial commitment of its witness. This way we can do without any interaction for encryption/decryption through a one-time setup where parties publish the commitments over which all following encryptions are done. We construct our ECW scheme Π_{ECW} as follows:

System Parameters: We assume that a commitment key $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ is contained in the genesis block B_0 of the underlying blockchain.

Setup Phase: All parties $P_i \in \mathcal{P}$ proceed as follows:

1. Compute a commitment $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$ to $\mathsf{sk}_{L,i}$ using randomness ρ_i. We abuse the notation and define P_i's secret key as $\mathsf{sk}_{L,i}||\rho_i$.
2. Compute a signature $\sigma_i \leftarrow \mathsf{Sig}_{\mathsf{Sig.sk}_i}(\mathsf{cm}_i)$.
3. Publish $(\mathsf{cm}_i, \sigma_i)$ on the blockchain by executing $\mathsf{Broadcast}(1^\lambda, (\mathsf{cm}_i, \sigma_i))$.

Encryption $\mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, m)$**:** Construct a circuit C that encodes the predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i})$, where $\mathbf{B}$, sl and P are hardcoded and $\mathsf{sk}_{L,i}$ is the witness. Let $\mathcal{P}_{Setup}$ be the set of parties with non-zero relative stake and a valid setup message $(\mathsf{cm}_i, \sigma_i)$ published in the common prefix $\mathbf{B}^{\lceil\kappa}$ (if P_i has published more than one valid $(\mathsf{cm}_i, \sigma_i)$, only the latest one is considered). For every $P_i \in \mathcal{P}_{Setup}$, compute $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{cWE}(\mathsf{ck}, \mathsf{x}_i = (\mathsf{cm}_i, C, 1), m)$. Output $\mathsf{ct} = \big(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \{\mathsf{ct}_i\}_{P_i\in\mathcal{P}_{Setup}}\big)$.

Decryption $\mathsf{Dec}(\mathbf{B}, \mathsf{ct}, \mathsf{sk})$**:** Given $\mathsf{sk} := \mathsf{sk}_{L,i}||\rho_i$ such that $\mathsf{cm}_i = \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$ and $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$ for parameters $\mathbf{B}, \mathsf{sl}, \mathsf{P}$ from ct, output $m \leftarrow \mathsf{Dec}_{cWE}(\mathsf{ck}, \mathsf{ct}_i, (\mathsf{sk}_{L,i}, \rho_i))$. Otherwise, output $\perp$.

E.4 Constructing AfP

We present two constructions of AfP: a general one for any lottery predicate and a specific one for lottery predicates based on VRFs. Both constructions were introduced in [8], whose description we present in almost verbatim form.

General AfP. In general we can add authentication to a message as shown in [8]. Recall that P_i wins P if $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$. Here, $\mathcal{R}(\mathsf{x} = (\mathbf{B}, \mathsf{sl}, \mathsf{P}),$

$\pi^{\mathsf{cw}}) = \mathsf{lottery}(\mathtt{x}, \pi^{\mathsf{cw}})$ is an NP relation where all parties know $\mathtt{x}$ but only the winner knows a witness π^{cw} such that $\mathcal{R}(\mathtt{x}, \pi^{\mathsf{cw}}) = 1$. We can therefore use a signature of knowledge (SoK) [12] to sign m under the knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{P}, \mathsf{sk}_{L,i}) = 1$. This will attest that the message m was sent by a winner of the lottery for P.

VRF-based AfP. Using the VRF-based lottery $\mathsf{lottery}_{\mathsf{VRF}}$, we can construct a more efficient VRF-based AfP as shown in [8]. We first note that our general approach of applying a SoK for the knowledge of a secret key still applies. However, using the structure of the lottery, and in particular the VRF, allows for a much more efficient AfP which has applications in most PoS settings as well.
The AfP scheme uses a $\mathsf{NIZKPoK}$ which has a setup executed as a part of the blockchain setup such that the CRS is in the genesis block. The algorithms for the scheme are $\pi \leftarrow \mathsf{NIZKPoK.Prove}(\mathsf{crs}, \mathtt{x}, \pi^{\mathsf{cw}})$ and $\{0,1\} \leftarrow \mathsf{NIZKPoK.Verify}(\mathsf{crs}, \mathtt{x}, \pi)$. Notice that this construction satisfies both the basic definition of AfP (Definition 12) and AfP privacy (Definition 13).

Protocol Π_{AfP} The VRF-based AfP protocol Π_{AfP} is described below.

Authenticate. $\sigma \leftarrow \Pi_{\mathsf{AfP}}.\mathsf{Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_{L,j}, m)$ To authenticate a message, m, a party first checks that $\mathsf{lottery}_{\mathsf{VRF}}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_{L,j}) = 1$. It then obtains the output and certificate $(y, \pi_{\mathsf{VRF}}) \leftarrow \mathsf{VRFrf.Prove}_{\mathsf{VRF.sk}_{L,j}}(\mathsf{sl}||\mathsf{P}||\eta)$. Finally, it produces $\pi_{\mathsf{NIZKPoK}} \leftarrow \mathsf{NIZKPoK.Prove}\{\sigma_{\mathsf{SIG}} \mid \mathsf{Sig.Verify}_{\mathsf{Sig.pk}_j}(\sigma_{\mathsf{SIG}}, m) = 1\}$ which is a NIZK-PoK of a signature produced under $\mathsf{Sig.sk}_j$.
It then outputs a tuple $\sigma_{\mathsf{AfP}} \leftarrow (\mathsf{Sig.pk}_j, y, \pi_{\mathsf{VRF}}, \pi_{\mathsf{NIZKPoK}})$

Verify. $\{0,1\} \leftarrow \Pi_{\mathsf{AfP}}.\mathsf{Verify}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m)$ To verify an AfP tag the verifier obtains parameters from the blockchain $(\{\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i\}_{i\in[n]}, \eta, \phi) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$. It then parses the tag as $\sigma_{\mathsf{AfP}} \leftarrow (\mathsf{Sig.pk}_j, y, \pi_{\mathsf{VRF}}, \pi_{\mathsf{NIZKPoK}})$ and gets the VRF verification key $\mathsf{VRF.pk}_j$ for the party that the AfP points to. It then checks the following

1. Makes sure that $\mathsf{VRF.Verify}_{\mathsf{VRF.pk}_j}(\mathsf{sl}||\mathsf{P}||\eta, y, \pi_{\mathsf{VRF}}) = 1$ *i.e.* the VRF output was correctly generated under lottery key of party P_j.
2. Checks that $\mathsf{NIZKPoK.Verify}(\pi_{\mathsf{NIZKPoK}}, (\mathsf{Sig.pk}_j, m)) = 1$ which verifies the proof of signature knowledge.
3. And $y < \phi(\mathsf{stake}_j)$ which makes sure that the lottery was conducted correctly with the stake of P_j.

If all checks go through, the algorithm outputs 1. Otherwise, it outputs 0.