StakeNote: A Proof-of-Stake Protocol for CryptoNote Payments

Bernardo David
Common Prefix and IT University of Copenhagen (ITU)
bernardo@bmdavid.com

Dimitris Karakostas
Common Prefix
dimitris@commonprefix.com

Abstract

This work proposes StakeNote, a distributed ledger protocol that combines Proof-of-Stake (PoS) with privacy and anonymity preserving payments. The protocol combines Ouroboros Praos, a provably secure PoS protocol, with CryptoNote, a privacy-preserving payment system based on ring signatures which has been widely used in practice. We prove that StakeNote inherits the security guarantees of Ouroboros Praos and the privacy guarantees of CryptoNote and we demonstrate its practicality via a proof of concept implementation, where block creation requires less than 25 ms and eligibility proofs are approx. 3 KB for anonymity sets of size 16. Finally, we discuss heuristic enhancements that potentially increase privacy and enable dynamic participation.

1 Introduction

Bitcoin [Nak08] introduced a new paradigm for distributed computing and financial applications. On the technical side, Proof-of-Work (PoW) sybil resilience enabled an open and permissionless database, where participants join and leave the protocol arbitrarily. On the application side, the Bitcoin cryptocurrency established Decentralized Finance (DeFi) [WPG+22], which was later enhanced with programmability capabilities via smart contracts. Still, Bitcoin left a lot to be desired.

The first concern is Bitcoin's sustainability. Bitcoin mining consumes such amounts of energy [fAF25, Dig25] that call for regulatory or policy interventions [CGM23]. This concern drove research to more sustainable approaches, the most promising being Proof-of-Stake (PoS) [KRDO17, DGKR18]. PoS replaces physical assets, namely energy in PoW, with digital assets, namely the assets ("stake") traded on the ledger itself.

A second concern stemmed from Bitcoin's public nature. By default all information on Bitcoin, often of sensitive financial nature, is public and readable by everyone. To avoid this, various protocols aim to guarantee user anonymity and privacy, with the most popular being Zerocash [BCG+14] and CryptoNote [VS13], used by the Zcash and Monero platforms respectively.

Naturally, these concerns have led to the exploration of combined solutions, i.e., privacy-preserving PoS protocols. Unfortunately, although the literature does offer such protocols [KKKZ19, GOT19], none has been deployed in practice. As academic research has mostly focused on Zerocash, little progress has been made on practical privacy-preserving PoS protocols using CryptoNote. This lack of research can be attributed to multiple reasons. First, Zerocash is suitable for academic research, since it is provably secure and offers well defined strong privacy guarantees. On the contrary, CryptoNote has not been formally defined nor proven secure yet. In

This work was partially funded by the Zano project and Common Prefix.

fact, a there is a line of works on tracing CryptoNote transactions [KFTS17, MSH+18, YAY+19, HH19, WLS+19, YAV19] under different conditions and different degrees of success, showing that the exact anonymity guarantees are unclear. While there exist provably secure cryptocurrency schemes [MM18, FMMO19] based on the CryptoNote approach, those schemes assume an underlying distributed ledger and are not directly compatible with existing PoS consensus protocols. As a result, none of the systems that use Cryptonote is pure PoS: Monero[1] uses PoW, Zano[2] uses an ad hoc hybrid of PoW and PoS, whereas Sentz[3] uses a custom quorum-based mechanism inspired by Stellar [Maz15].

Our work fills this gap by proposing a *ring-friendly, secure, and practical* PoS distributed ledger protocol for CryptoNote payments. Our protocol is: (i) secure, meaning it guarantees safety and liveness; (ii) as private as CryptoNote; (iii) practical, i.e., does not rely on heavy and inefficient cryptographic primitives and is as efficient as deployed PoS systems. Essentially, our protocol is a realistic option for privacy preserving blockchain systems, in order to move away from PoW towards a more environmentally sustainable alternative.

Related Work

Two design philosophies dominate the literature on private payments. The first, introduced by Zerocash [BCG+14], uses succinct non-interactive zero-knowledge proofs to hide each transaction within the entire set of unspent outputs. The second, introduced by CryptoNote [VS13], uses ring signatures to hide each transaction within an explicit anonymity set of decoys, chosen at signing time. The two approaches expose a fundamental tradeoff between the strength of the anonymity guarantee and the strength of the setup assumption. Zerocash-style protocols offer anonymity within the full UTxO set, but rely on a structured common reference string (CRS) whose generation requires a trusted setup ceremony and, in classical instantiations, must be repeated for each relation.[4] CryptoNote-style protocols require no setup beyond the random oracle, but their anonymity is bounded by the anonymity set used in each transaction and is sensitive to decoy selection.

The same tradeoff carries over to privacy-preserving PoS protocols. Ouroboros Crypsinous [KKKZ19] embeds a Zerocash-style payment scheme within a PoS leader election, while the privacy-aware blockchain of [GOT19] assumes a list of commitments to each party's stake (which RingCT [NM+16] already provides) and builds the eligibility proofs from generic non-interactive zero-knowledge proofs. Both inherit the structured CRS of the underlying proof system. A modular formalization of anonymous PoS leader election [BMSZ20], instantiated with Algorand's selection function, similarly relies on non-interactive zero-knowledge proofs when parties hold arbitrary stake amounts. To our knowledge, no prior PoS protocol explores the CryptoNote side of the tradeoff. StakeNote fills this gap by offering the same (locally bounded) anonymity as CryptoNote and requiring no structured setup.

We note that an orthogonal line of work studies the inherent privacy limitations of anonymous PoS, both information-theoretically [KMNS21] and under differential privacy [WPNM23]. These results apply to any anonymous PoS protocol, including ours, and characterize the anonymity that any such construction can hope to achieve.

2 Model and Preliminaries

We now outline our security model and assumptions, along with the cryptographic primitives and protocols on which our protocol relies.

[1] `https://www.getmonero.org`
[2] `https://zano.org`
[3] `https://www.sentz.com`
[4] Recent transparent or universal-updatable proof systems reduce or remove this dependence, but to our knowledge no privacy-preserving PoS protocol has been built atop them.

2.1 Security Model and Assumptions

Delayed Adaptive Adversaries Instead of security fully adaptive adversaries as achieved in Ouroboros Praos, we consider a weaker form of delayed adaptive adversary. Such adversaries may request to corrupt any party at any point in time but the corruption takes places only after a preset delay. As in the original Ouroboros protocol, we set this delay to be the duration of an epoch, such that any corruption request made in a certain epoch only gives control over the corrupted party to the adversary in the next epoch. This ensures that an adaptive adversary cannot create arbitrary conflicting versions of a block by immediately corrupting a slot leader after it broadcast a block.

Network Model As in Ouroboros Praos, we consider a semi-synchronous network, where the adversary $\mathcal{A}$ can selectively delay any messages sent by honest parties for up to $\Delta \in \mathsf{poly}(\lambda)$ slots where λ is the security parameter. This is formalized in the diffuse functionality discussed in Section 2.2.

GKL security properties As in Ouroboros Praos, we adopt the GKL model [GKL15] to capture the security properties, namely, common prefix, chain quality and chain growth. We briefly recall these properties informally. Common prefix states that for the blockchain in the view of any honest party, removing a suffix of k blocks results in a prefix that is in the blockchain in the view of all other honest parties. Chain quality states that at least one block generated by an honest party appears in a portion of s blocks of the blockchain in an honest party's view. Chain growth states that, every s slots, at least $\tau \cdot s > 0$ blocks are produced.

Execution Model As In Ouroboros Praos, we adopt the GKL model [GKL15] to capture the execution of our protocol. In this model, an *environment* orchestrates the execution by instructing parties to create new transactions, diffusing transactions among parties (i.e., emulating the inclusion of transactions in a mempool via a peer-to-peer network) and instructing the adversary to corrupt parties. The execution is augmented by a *functionality*, which models among other things semi-synchronous network communication among parties and setup assumptions (i.e., the initial stake distribution and key generation for parties present in the beginning of the execution). For any environment and adversary, the GKL security properties must hold for the blockchains in the view of all honest parties, except with negligible probability. In order to achieve these properties, we must restrict the environment to activate all honest parties often enough and not to allow the adversary to corrupt parties that jointly control 50% or more of the stake.

ℓ-DDH Hardness Assumption It was shown in [NR04] that if the standard DDH-problem is hard, then the ℓ-DDH problem is hard for $\ell(\lambda) = poly(\lambda)$ and the distinguishing advantage is the same. Note that the 1-DDH hardness assumption is the usual DDH hardness assumption.

Definition 1 (ℓ-DDH hardness assumption)**.** *Let $\ell := \ell(\lambda)$ be an integer depending on the security parameter λ. Let D_0, D_1 be the probability distributions outputting tuples $\mathbf{x} := (G, \alpha \cdot G, \{\beta_i \cdot G\}_{i=0}^{\ell-1}, \{\gamma_i \cdot G\}_{i=0}^{\ell-1})$ where G is a uniformly random generator of a group $\mathbb{G}$ of order q, and $\alpha, \{\beta_i\}_{i=0}^{\ell-1}$ are independently sampled uniformly random values in $\digamma_q$. In the case of D_0, $\gamma_i = \alpha \cdot \beta_i$ for all $i \in [0, \ell-1]$. In the case of D_1, $\{\gamma_i\}_{i=0}^{\ell-1}$ are sampled uniformly at random from $\digamma_q$ and independently of each other and of everything else. Given a PPT adversary $\mathcal{A}$, we define its distinguishing advantage as the probability $Adv(\mathcal{A}) = \left| Pr[b = b' \mid b \xleftarrow{\$} \{0,1\}, \mathbf{x} \xleftarrow{\$} D_b, b' \xleftarrow{\$} \mathcal{A}(\mathbf{x})] - 1/2 \right|$. We say that the ℓ-DDH problem is hard if for every PPT adversary $\mathcal{A}$, $Adv(\mathcal{A})$ is negligible in λ.*

Random Oracle Model Throughout this paper we model hash functions as random oracles with the appropriate output space. This is crucial for the VRF construction and block signature (i.e., modified CLSAG) we adopt.

2.2 The Ouroboros Family of PoS Protocols

Ouroboros Praos is a Proof-of-Stake (PoS) distributed ledger protocol which is secure against adaptive corruptions under a semi-synchronous network [DGKR18].

In Ouroboros Praos, time is divided into discrete units called *slots*. In the context of a blockchain-based distributed ledger, each time slot is associated with at most one *block*. Parties are equipped with synchronized clocks that indicate the current slot, such that clock drift is subsumed in the slot's length. In general, each slot sl_r is indexed by an integer $r \in \{1, 2, \ldots\}$, where the real time window that corresponds to each slot has two properties. The network is semi-synchronous, i.e., the adversary $\mathcal{A}$ can selectively delay any messages sent by honest parties for up to $\Delta \in \mathsf{poly}(\lambda)$ slots where λ is the security parameter.

In Ouroboros Praos, each party U_i is identified by: (i) an amount of stake $v_i \in \mathbb{N}$; (ii) a Key-Evolving Signature (KES) public key $\mathsf{vk}_i^{\text{kes}}$; (iii) a Verifiable Random Function (VRF) public key $\mathsf{vk}_i^{\text{vrf}}$; (iv) a regular $\mathsf{EUF\text{-}CMA}$ secure digital signature public key $\mathsf{vk}_i^{\text{dsig}}$.

The party's stake v_i defines the party's power (weight) within the consensus protocol. Intuitively, the probability of a party becoming eligible for participation is a (almost) linear function of its stake. For each slot, every party may be elected as slot leader, which allows them to create a block that corresponds to the slot.

The leader election is based on local sortition and implemented via a Verifiable Random Function (VRF) [MRV99]. Specifically, each party computes locally the output of the VRF, using their VRF private key and the slot index as input. If the VRF output is below a threshold, which is a (slightly concave) function of their stake, then they are an eligible slot leader.

After being elected as slot leader, a party constructs a block by collecting unpublished transactions and relevant metadata (e.g., timestamps). The party also includes the VRF eligibility proof, which can be verified by anyone using the party's public VRF key. Finally, the party signs the block using their private KES key, evolve it, and immediately delete the used key. In this manner, even if the adversary corrupts the party immediately afterwards, they cannot retroactively create blocks for past slots for which the party was elected as leader, since the corresponding block signing key has been evolved and deleted.

Finally, the party may create a transaction that spends their stake using their regular key $\mathsf{vk}_i^{\text{dsig}}$. To support dynamic stake distributions, where parties can spend their assets and stake distribution shifts, Ouroboros Praos is organized in *epochs*. Before each epoch, the stake distribution is captured in a snapshot and protocol participation within the epoch follows the snapshot. Additionally, a randomness beacon, which is simulatable using a hash function, defines a pseudorandom nonce per epoch, used to randomize the leader election process.

Leader election A core requirement of our protocol is *sybil resilience and collusion proofness*. A stakeholder should be independently selected, with probability p_i which depends only on its relative stake. In Ouroboros Praos, the probability is determined by the stakeholder's relative stake and a parameter f, called the *active slots coefficient*.[5] Specifically, $p_i = \phi_f(\alpha_i) \triangleq 1 - (1-f)^{\alpha_i}$, where α_i is the relative stake held by stakeholder U_i ($\alpha_i = \frac{v_i}{\sum_j v_j}$).

This function has the desirable property of "aggregation independence", meaning that a party controlling relative stake $\frac{v}{v_{\mathsf{total}}}$ is elected with probability $\phi_f(\frac{v}{v_{\mathsf{total}}})$ no matter how they spread their stake among many UTxOs (or a few).

Note that, since the events "U_i is a leader for sl_j" are independent, this mechanism may generate multiple or no leaders for a given slot.

Diffuse functionality The diffuse functionality $\mathsf{DDiffuse}_\Delta$ models the communication network. It is parameterized by $\Delta \in \mathsf{poly}(\lambda)$ and interacts with (i) the environment $\mathcal{Z}$, (ii) parties (stakeholders) $U_1, \ldots, U_n$, (iii) and the adversary $\mathcal{A}_{\mathsf{net}}$. On each round, every activated party can fetch incoming messages, which have been diffused via $\mathsf{DDiffuse}_\Delta$ by some other party, and diffuse their own messages (once per round). Once activated, $\mathcal{A}_{\mathsf{net}}$ can read all inboxes and diffuse requests and deliver messages to parties in any order it prefers. Additionally, $\mathcal{A}_{\mathsf{net}}$ can

[5] A party controlling all stake has probability f to generate a block: $\phi_f(1) = f$.

delay the deliver of any message to any party up to Δ slots (at which point the functionality delivers the delayed message to the respective parties' inboxes).

Ouroboros Genesis Fork Choice Rule For the fork choice rule, we turn to Ouroboros Genesis [BGK+18], the followup to Ouroboros Praos. One drawback of Ouroboros Praos was the usage of "moving checkpoints" for long-range attack prevention. Briefly, if a new chain forks the party's chain by more than k blocks, then the new chain is discarded. Due to the checkpoints, newly-joining parties needed to acquire a relatively recent finalized block from a trusted source in order to properly "bootstrap". Ouroboros Genesis avoids this requirement via a fork choice rule that enables a newly-joining party to correctly bootstrap while having access only to the protocol's genesis block. Specifically, under Genesis's fork choice rule, given two chains that fork by more than k blocks, the party chooses the one that grows at a faster rate immediately after the forking point. In the context of our work, since our protocol is oblivious to the chain selection rule, we opt for the Ouroboros Genesis rule which gives the best guarantees.

2.3 CryptoNote and Confidential Transactions

CryptoNote is a distributed ledger protocol that aims to offer transaction untraceability [VS13]. The original CryptoNote whitepaper proposed various advancements on Bitcoin's blockchain-based payment system, such as an ASIC-resistant Proof-of-Work (PoW) function, as well as an alternative coin generation function and scripting language. Nonetheless, its main contribution was a transaction mechanism, which is based on traceable ring signatures [FS07].

At a high level, the CryptoNote protocol inherits the UTxO model of Bitcoin. Specifically, each unspent output holds an amount of assets, which (in the original version of CryptoNote) is public. In order to spend a UTxO the party picks a subset of all outputs, whose amount is equal to the UTxO. Following, the UTxO's owner creates a ring signature, which proves that they own (the private key of) one of the outputs, among the chosen subset plus the correct UTxO.

In order to avoid double-spending, the signature is *one-time* via the usage of *key images*. Specifically, each UTxO is controlled by a one-time private key. To consume the UTxO, the party creates a public key using the (UTxO's) private key, called a "key image". The key image is not linked with the public key which is used within the ring signature of the transaction, but is deterministically produced using the same private key. Therefore, when receiving a transaction, participants cross-reference its key images with the set of all past key images; if the key image has been observed before, i.e., if the private key has been used in a past transaction, then the new transaction is rejected.

Finally, CryptoNote enables a party to publish a single address (i.e., public payment key) and receive arbitrary unlinkable payments. To do so, the sender of a transaction creates a one-time public key using a random mask. Following, the recipient observes all newly-created outputs. If a transaction is aimed at them, the recipient can both identify it and recover the corresponding private key.

Confidential Transactions Confidential transactions (CT) is a method for hiding a UTxO's value [PBF+19]. The main idea is using homomorphic commitments, such as Pedersen commitments [Ped92]. Specifically, each UTxO is linked with a commitment of its asset amount. Following, a transaction includes a *range proof*, which proves that (i) the sum of all (committed) input values is greater than the sum of all (committed) output values; (ii) each output value is positive. In other words, the range proof shows that the transaction does not generate new assets.

Ring Confidential Transactions A notable building block of our work is Ring Confidential Transactions (RingCT) [NM+16]. RingCT combines the ring signatures of CryptoNote with the CT scheme, resulting in a payment protocol with both transaction unlinkability and amount hiding.

Concise Linkable Spontaneous Anonymous Group (CLSAG) proofs Finally, our construction relies heavily on CLSAG proofs and signatures [GNB19]. Briefly, CLSAGs are ring signatures inspired by Monero's Multilayered Linkable Spontaneous Anonymous Group (MLSAG) Signatures [NM+16], but with performance enhancements that are particularly useful in our case.

2.4 Building Blocks

Verifiable Random Functions (VRFs) Our protocol uses Verifiable Random Functions (VRF) [MRV99] with unpredictability under malicious key generation [DGKR18, GS24]. VRFs with unpredictability under malicious key generation consist of three algorithms: $\mathsf{VRF.KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: on input a security parameter λ outputs a secret key sk and a public key pk; $\mathsf{VRF.Eval}(\mathsf{sk}, x) \to (y, \pi_y)$: on input a secret key sk and a value x, outputs a pseudorandom value $y \in \{0,1\}^{l_{\mathsf{VRF}}}$ and a proof π_r; $\mathsf{VRF.Verify}(\mathsf{pk}, x, y, \pi_y) \to 0/1$: on input a public key pk, a value x, a value y, and a proof π_y, outputs 1 if and only if the proof verifies.

Range proofs In order to prove eligibility, StakeNote also uses a range proof for a specially crafted commitment. Various protocols can be used to implement range proofs, with varying performance and trust assumptions. One such protocol was described in the "confidential assets" paper [PBF+19] and does not require a trusted setup. Another zero-knowledge proof protocol, which has been widely used in a similar context of confidential payments, is Bulletproofs [BBB+18]. For a systematic review of the space of zero-knowledge range proofs we refer to [CBC+24].

3 StakeNote Protocol

At a high level, the protocol follows the same steps as Ouroboros Praos [DGKR18], with some changes in the public setup and leader eligibility proofs. In our setting, each output consists of a public key and a commitment to the output's stake value. To compute each epoch's snapshot, parties should know when and if an output has been spent. Instead, we require the block generator to prove that they had an unspent output at the time of the snapshot when they create a block.

On each slot, every party takes the following steps to create a block:[6]

1. $\mathcal{P}$ creates a VRF output, using the output's VRF key, in the same manner as Praos, i.e., passing as input the epoch's nonce and slot number.
2. If the VRF output is below some threshold, which is a function of the output's stake value, then $\mathcal{P}$ is the slot leader and proceeds as follows:
   (a) $\mathcal{P}$ collects a subset of the snapshot's UTxOs including the chosen output.
   (b) $\mathcal{P}$ creates a block with the following proofs:
      i. A proof that $\mathcal{P}$ knows the spending key of an output in the chosen subset and that it was unspent at the time of the snapshot.
      ii. A proof that the private VRF key, which created the VRF output from Step 1, is deterministically derived from the private payment key of the same output from Step 2(b)i.
      iii. A proof that the stake value of the same output from Step 2(b)i was used to compute the threshold of Step 2.
   (c) $\mathcal{P}$ signs the block by turning the proofs from Step 2b to a signature of knowledge, proving that the private signing key is the same as the spending key of the output from Step 2(b)i.

[6] If $\mathcal{P}$ controls multiple UTxOs, then the steps are executed for every output.

3.1 Construction

At its core, our construction relies on a proof which shows that the block is produced by some eligible unspent UTxO, among a subset of UTxOs in the snapshot, without revealing which specific UTxO is being used.

The eligibility criterion is the same as Ouroboros Praos, i.e. the output of a VRF evaluated on a random nonce under the VRF secret key corresponding to the VRF public key in the UTxO should be below a certain threshold determined by the leader election probability function. Each UTxO models a single party, so it is associated with a VRF key. Notably, a UTxO's private VRF key is chosen to be the same scalar as the UTxO's private payment key, i.e., the key that is used to consume the output.

Briefly, $\mathcal{P}$ generates the VRF key pair in a way that the secret key is the same as the payment key, computes the VRF output and checks whether he is elected as slot leader, in the same manner as Ouroboros Praos. If so, $\mathcal{P}$ creates the block, proves that the VRF secret key is the same as the UTxO's payment key and that the UTxO's (committed) stake value is above a certain value such that the VRF output is below the threshold determined by the leader election probability function evaluated on that value, and finally creates a ring signature on the block with the UTxO's payment key and a set of decoys. Crucially, all of the above is done in a way that *does not reveal* the exact UTxO that is being used, but only reveals a set of UTxOs which includes the correct one.

Snapshot As in Ouroboros Praos, the stake distribution used to elect slot leaders at a certain epoch must be "frozen" before the random nonce for that epoch is generated. This means that for leader election in a given epoch we must take into consideration only UTxOs that appear in blocks far enough in the past of the chain, *before* the protocol for generating the random nonce is initiated. Moreover, we must ensure that all honest transactions in this snapshot are seen by all honest parties. In practice, this means that a snapshot used for leader election in a given epoch contains all of the UTxOs that appear in blocks generated 2 epochs ago, as the random nonce generation protocol is executed in the previous epoch.

Differently from Ouroboros Praos, as we consider CryptoNote transactions, our snapshot specifically contains both the set $\mathbb{O}_e$ of UTxOs that appeared in the blocks generated up to 2 epochs before epoch e (*until epoch* $e-2$) and the set $\mathbb{I}_e$ of key images that have appeared in the blocks generated up epoch $e-2$. Notice that by proving that it controls an UTxO in $\mathbb{O}_e$ whose key image is *not* in $\mathbb{I}_e$ a party proves that this UTxO was unspent at the time of the snapshot, and thus it is eligible for the leader election procedure.

Looking forward, another particularity of the use of snapshots in our case is that when using a UTxO to generate a modified CLSAG as part of a leader election proof, a party must only *choose decoy UTxOs that are also in the snapshot.* In particular, this means that once a UTxO has been used in such a modified CLSAG, it must always be used with the same set of decoys. Otherwise, the anonymity set protecting that UTxO diminishes, as it will be known that the UTxO used in all modified CLSAGs is the same (because the key image is revealed in the newly generated blocks) and that it must be in the intersection of all the decoy sets used in the newly generated blocks.

Objects and notation

- Global protocol parameters:
  - G_1, G_2, G_{PAY}, G_{VRF}: group generators of finite cyclic groups over a finite field F_p for some prime p.
  - n: ring size of CryptoNote payment transactions.
  - random oracle H_{clsag}.
  - δ: snapshot height delta, i.e., the slot distance between the snapshot's taking and the beginning of the epoch.
  - v_{total}: the total amount of stake in the system.

- For every UTxO (output) o, we have the following:
  - private elements: (i) sk_o: private payment key; (ii) v_o: stake value; (iii) r_o: randomness of value's Pedersen commitment.
  - public elements: (i) $\mathsf{vk}_o = \mathsf{sk}_o \cdot G_{\mathsf{PAY}}$: public payment key; (ii) $C_o = v_o \cdot G_1 + r_o \cdot G_2$: Pedersen commitment to output's stake value.
- For every spent UTxO (output) o, we have the output's key image $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$ that is *not linkable* to o.
- For every UTxO (output) o used to generate a block, we have the following public elements that are *not linkable* to o: (i) the output's key image $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$; (ii) the VRF public key $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$.
- Output sets: (i) $\mathbb{O}$: all outputs that have ever appeared; (ii) $\mathbb{I}$: all key images that have ever appeared.
- Epoch e: (i) sl_e: epoch start (slot); (ii) η_e: epoch random nonce.
- Snapshot sets for epoch e: (i) $\mathbb{O}_e \subset \mathbb{O}$: outputs in the snapshot (UTxOs that appeared in blocks produced up to epoch $e-2$); (ii) $\mathbb{I}_e \subset \mathbb{I}$: key images in the snapshot (UTxOs that appeared in blocks produced up to epoch $e-2$).

Block creation We present the protocol for block creation in Figure 1. The protocol is run in the context of an epoch e for which a random nonce η_e has been generated in the previous epoch (i.e., in epoch $e-1$) following the same procedure as in Ouroboros Praos. Moreover, the epoch defines a snapshot consisting of a set $\mathbb{O}_e$ of outputs and a set $\mathbb{I}_e$ of key images that appear in blocks created before the previous epoch (i.e., block created up to epoch $e-2$).

In our protocol, each UTxO o is associated with private values sk_o, v_o, r_o that implicitly define a VRF key pair $(\mathsf{sk}_o, \mathsf{vk}_{\mathrm{VRF},o})$ where $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$. A party $\mathcal{P}$ is elected as a slot leader (i.e., allowed to create a block) if, for any of their unspent UTxOs in $\mathbb{O}_e$, they can obtain a small enough VRF output $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{v_o}{v_{\mathsf{total}}})$ by evaluating the VRF on the epoch's nonce η_e using the UTxO's sk_o as secret key. Intuitively, this allows the party to prove they were elected by publishing $(\mathsf{vk}_{\mathrm{VRF},o}, y_o, \pi_{y_o})$ in their block, but it leaves the challenge of proving that $\mathsf{vk}_{\mathrm{VRF},o}$ and y_o are valid without revealing o or v_o.

In order to preserve privacy for v_o, we instead reveal a value $T < v_o$ such that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{T}{v_{\mathsf{total}}})$. However, we need to prove that $T < v_o$ without revealing y_o, which we do by revealing a commitment $\mathcal{C}$ to the UTxO value minus T and using a range proof π_{RANGE} to show it contains a positive value. In order to preserve privacy for o and sk_o, we follow the CryptoNote approach by revealing a set of UTxOs $\mathbb{O}_{\mathsf{block}} \subset \mathbb{O}_e$ such that $o \in \mathbb{O}_{\mathsf{block}}$ (instead of revealing o itself), while proving that in $\mathbb{O}_{\mathsf{block}}$ there is an unspent UTxO with sk_o, v_o, r_o such that $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$ and that $\mathcal{C}$ is correctly computed from v_o, r_o, T.

Putting the previous ideas together, we want to create a block by generating a signature on its payload B and proving correctness of revealed values $(\mathsf{vk}_{\mathrm{VRF},o}, y_o, \pi_{y_o}), \mathcal{C}, T, \pi_{\mathsf{RANGE}}$, which allow us to check that the slot leader $\mathcal{P}$ indeed was elected to create a new block. Moreover, we need to show that the UTxO $o \in \mathbb{O}_e$ was unspent when the snapshot was taken. When creating a block, $\mathcal{P}$ publishes a random set of UTxOs $\mathbb{O}_{\mathsf{block}} \subset \mathbb{O}_e$ containing o (plus decoys) together with a proof of knowledge of private values sk_o, v_o, r_o for o such that the following statements hold *without revealing the private values* sk_o, v_o, r_o *or* o:

1. $o \in \mathbb{O}_{\mathsf{block}}$
2. o can be spent using sk_o
3. the VRF public key is $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$
4. the block is signed using sk_o.

Protocol StakeNote

This protocol is run by every party $\mathcal{P}$ on every epoch e with snapshot sets $\mathbb{O}_e, \mathbb{I}_e$ (UTxOs and key images in blocks up to epoch $e-2$, respectively) and random nonce η_e (assumed to be generated as in Ouroboros Praos). A party controls an output $o \in \mathbb{O}_e$ if it knows the secret key $\mathsf{sk} = (\mathsf{sk}_o, v_o, r_o)$ connected to the output.

- **Setup:** Parties have access to an initialization functionality $\mathcal{F}_{\mathsf{INIT}}$, which holds system parameters p for our modified CLSAG, the (confidential) initial stake distribution and the genesis block containing CLSAG-authenticated RingCT transactions following this stake distribution. In the beginning of the execution, every party $\mathcal{P}$ calls $\mathcal{F}_{\mathsf{INIT}}$ with $(\mathsf{init}, \mathcal{P})$, receiving as response parameters p, the genesis block B_0 and the secret key $\mathsf{sk} = (\mathsf{sk}_{\mathcal{P}}, v_{\mathcal{P}}, r_{\mathcal{P}})$ corresponding to a UTxO $(\mathsf{sk}_{\mathcal{P}} \cdot G_{\mathsf{PAY}}, v_{\mathcal{P}} \cdot G_1 + r_{\mathcal{P}} \cdot G_2)$ contained in B_0.
- **Block Creation:** On a slot sl_r, for every unspent output o in the epoch's snapshot $\mathbb{O}_e$ that party $\mathcal{P}$ controls, it proceeds as follows:
  1. Compute $\mathsf{VRF.Eval}(\mathsf{sk}_o, \eta_e || \mathsf{sl}_r) \to (y_o, \pi_{y_o})$.
  2. If $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{v_o}{v_{\mathsf{total}}})$, o is eligible for creating a block on slot sl_r, so $\mathcal{P}$ proceeds. Otherwise, $\mathcal{P}$ ignores the next steps, advancing to the next $o \in \mathbb{O}_e$.
  3. Set the block payload B (transactions and metadata, e.g., timestamps, etc).
  4. Determine the minimum T such that $v_o \geq T$ and $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{T}{v_{\mathsf{total}}})$.
  5. Sample $r' \xleftarrow{\$} \mathbb{Z}_q$ and compute commitment $\mathcal{C} = (v_o - T) \cdot G_1 + r' \cdot G_2$.
  6. Generate a range proof[a] π_{RANGE} that commitment $\mathcal{C}$ contains $(v_o - T) \geq 0$.
  7. Set block contents $\mathcal{M} = \langle B, e, \mathsf{sl}_r, (\mathsf{vk}_{\mathrm{VRF},o}, y_o, \pi_{y_o}), T, \mathcal{C}, \pi_{\mathsf{RANGE}}, \Im_o \rangle$, where $\Im_o$ is the key image $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$ for o and $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$.
  8. Sample a random subset of outputs $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$ such that $o \in \mathbb{O}_{\mathsf{block}}$ and generate a modified CLSAG [GNB19] (Section 3.2) $\mathsf{Sign}(L_{\mathsf{SN}}, (\mathsf{sk}_o, v_o, r_o, r'), (T, \mathcal{C}, \mathsf{vk}_{\mathrm{VRF},o}, \Im_o, \mathbb{O}_{\mathsf{block}}), \mathcal{M}) \to \sigma_{\mathsf{CLSAG}}$.
  9. Diffuse the final block $\langle \mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma_{\mathsf{CLSAG}} \rangle$ in the network.
- **Block Verification:** For every block $\langle \mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma_{\mathsf{CLSAG}} \rangle$ received in epoch e, a full node proceeds as follows:
  1. Parse $\mathcal{M} = \langle B, e, \mathsf{sl}_r, (\mathsf{vk}_{\mathrm{VRF},o}, y_o, \pi_{y_o}), T, \mathcal{C}, \pi_{\mathsf{RANGE}}, \Im_o \rangle$ and check that it extends the local chain according to the Ouroboros Genesis chain rule.
  2. Check that $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$ and that $\Im_o \notin \mathbb{I}_e$ (i.e., $\mathbb{O}_{\mathsf{block}}$ is in snapshot $\mathbb{O}_e$ and the UTxO used to create the block was unspent according to $\Im_o$ and $\mathbb{I}_e$).
  3. Verify π_{RANGE} using the range proof verifier (i.e., check that $T < v_o$).
  4. Check that $\mathsf{VRF.Verify}(\mathsf{vk}_{\mathrm{VRF},o}, \eta_e || \mathsf{sl}_r, y_o, \pi_{y_o}) \to 1$ and that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(T)$ (i.e., v_o is high enough for the leader election to be valid).
  5. Check that $\mathsf{Verify}(L_{\mathsf{SN}}, (T, \mathcal{C}, \mathsf{vk}_{\mathrm{VRF},o}, \Im_o, \mathbb{O}_{\mathsf{block}}), \mathcal{M}, \sigma_{\mathsf{CLSAG}}) \to 1$ (i.e., $\mathsf{vk}_{\mathrm{VRF},o}$, $\mathcal{C}$ and $\Im_o$ are valid w.r.t. a witness $(\mathsf{sk}_o, v_o, r_o, r')$ corresponding to an UTxO $o \in \mathbb{O}_{\mathsf{block}}$ and T, and B is signed under this witness).
  6. Accept the block into the local chain if all checks pass.

[a] For example, this can be done using a Bulletproof [BBB+18]

Figure 1: Protocol StakeNote

5. The VRF output y_o, π_{y_o} is valid with respect to public key $\mathsf{vk}_{\mathrm{VRF},o}$ and input $\eta_e || \mathsf{sl}_r$ (i.e., $\mathsf{VRF.Verify}(\mathsf{vk}_{\mathrm{VRF},o}, \eta_e || \mathsf{sl}_r, y_o, \pi_{y_o}) \to 1$)
6. the stake value of o is $v_o \geq T$, such that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{T}{v_{\mathsf{total}}})$
7. o was unspent in the snapshot(or rather that the key image for o is not in $\mathbb{I}_e$)

Revealing only $\mathbb{O}_{\mathsf{block}}$ but not o maintains the same anonymity guarantees as the original

CryptoNote protocol. Proving the statements above ensures that the distribution of slot leaders (parties who create blocks) remains the same as in Ouroboros Praos, which allows us in principle to inherit the consensus security guarantees from that PoS protocol.

Statements 1, 2, 3 and 4 are proven via our modified CLSAG [GNB19] (Section 3.2). Informally, σ_{CLSAG} contains a new commitment $\mathcal{C}$ and proves knowledge of private values sk_o, v_o, r_o for an output $o \in \mathbb{O}_{\mathsf{block}}$ such that the following statements hold without revealing o or sk_o, v_o, r_o:

1. the VRF public key is $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$;
2. $\mathcal{C}$ is a commitment to the output's stake v_o minus the value T;
3. the output's key image is $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$.

Statement 5 is trivially proven by checking that $\mathsf{Verify}(\mathsf{vk}_{\mathrm{VRF},o}, x, y_o, \pi_{y_o}) \to 1$ given that $\mathsf{vk}_{\mathrm{VRF},o}$ is proven to be valid by σ_{CLSAG}, that y_o, π_{y_o} are revealed in the block and that $\eta_e || \mathsf{sl}_r$ is publicly known.

Statement 6 is proven leveraging the commitment $\mathcal{C}$ in σ_{CLSAG} and the range proof π_{RANGE}, which proves that $\mathcal{C}$ is a commitment to a positive value $v_o - T > 0$. This helps us establish that the value T revealed with the block is not larger than the actual value of o, meaning that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{T}{v_{\mathsf{total}}})$ can be trivially checked in public given that y_o is valid as established before.

Finally, statement 7 is proven by revealing the key image $\Im_o$ for o, allowing for publicly checking that $\Im_o \notin \mathbb{I}_e$. A caveat of this approach is that it is possible to link different blocks generated using the same o.

Block verification Block verification follows easily by checking all of the statements proven as part of block creation. This requires ensuring that $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$ and that $\Im_o \notin \mathbb{I}_e$, verifying the modified CLSAG σ_{CLSAG}, verifying the range proof π_{RANGE}, verifying the VRF output y_o, π_{y_o} and ensuring that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(T)$.

3.2 Block's CLSAG signature

We now describe our construction's main signature scheme, which is built using the ideas of CLSAG [GNB19]. This primitive allows us to prove that a slot leader has been elected due to controlling an UTxO o (in the snapshot), with value v_o, without revealing the exact UTxO. Essentially, the leader proves that the UTxO is unspent and that the value is "high enough", while keeping it private.

We will present this primitive as a *signature of knowledge* [CL06], an approach commonly used to construct ring signatures. Instead of proving that a message was authenticated by the owner of a given secret key as in a regular digital signature, a signature of knowledge allows for proving that a message is authenticated by a party who *knows* a witness w for a statement $w \in L$ for *any NP language* L. A signature of knowledge is composed of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$, which outputs parameters p implicitly taken as input by the other algorithms;
- $\mathsf{Sign}(L, w, x, m)$, which outputs a signature σ that is valid if w is a valid witness for a statement $x \in L$ for a NP language L.
- $\mathsf{Verify}(L, x, m, \sigma) \to \{0, 1\}$, which outputs 1 if and only if σ was generated using a valid witness w showing that $x \in L$.

Intuitively, the statement we are trying to prove can be imagined as a matrix of n rows and 4 columns, comprising an OR statement across rows and an AND statement across columns. Each row corresponds to one of the outputs in $\mathbb{O}_{\mathsf{block}}$, and each column (also often called a "layer") corresponds to a value of interest of which the party proves knowledge: (i) column 1 proves knowledge of sk_o corresponding to the payment key sk_o; (ii) column 2 proves that the VRF public key $\mathsf{vk}_{\mathrm{VRF},o}$ was computed from the same sk_o; (iii) column 3 proves that the key image $\Im_o$ was computed from the same sk_o; (iv) column 4 proves knowledge of the difference between the output's value and the revealed threshold and of the randomness used to compute a commitment $\mathcal{C}$ to this difference.

Formally, we are interested in statements in the language L_{SN} defined below.

Definition 2 (Language L_{SN})**.** *Let* $(w, x) \in L_{\mathsf{SN}}$*. It holds that* $w = (\mathsf{sk}_o, v_o, r_o, r')$ *and* $x = (T, \mathcal{C}, \mathsf{vk}_{VRF,o}, \Im_o, \mathbb{O}_{\mathsf{block}})$*, where* $\mathbb{O}_{\mathsf{block}} = \langle(\mathsf{vk}_0, C_0), \ldots, (\mathsf{vk}_{n-1}, C_{n-1})\rangle$*, such that:*

$$\bigvee_{i=0,\ldots,n-1} \Big(\mathsf{vk}_i = \mathsf{sk}_o \cdot G_{\mathsf{PAY}} \wedge \mathsf{vk}_{VRF,o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}} \wedge \Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_i) \wedge (r_o - r') \cdot G_2 = C_i - T \cdot G_1 - \mathcal{C}\Big).$$

Modified CLSAG Scheme

Our signature scheme uses a random oracle H_{clsag} with codomain F_q and consists of the tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{p}$: sample a cyclic group $\mathbb{G}$ of order q, generators $G_1, G_2, G_{\mathsf{PAY}}, G_{\mathsf{VRF}} \xleftarrow{\$} \mathbb{G}$, outputting $\mathsf{p} = (q, G_1, G_2, G_{\mathsf{PAY}}, G_{\mathsf{VRF}})$, which is implicitly taken as input by all algorithms.
- $\mathsf{Sign}(L_{\mathsf{SN}}, w, x, \mathcal{M}) \to \sigma$:
  1. Parse w as $(\mathsf{sk}_o, v_o, r_o, r')$, x as $(T, \mathcal{C}, \mathsf{vk}_{\mathrm{VRF},o}, \Im_o, \mathbb{O}_{\mathsf{block}})$ and $\mathbb{O}_{\mathsf{block}}$ as $\langle(\mathsf{vk}_0, C_0), \ldots, (\mathsf{vk}_{n-1}, C_{n-1})\rangle$.
  2. Find the index k, such that $\mathsf{vk}_k = (\mathsf{sk}_o \cdot G_{\mathsf{PAY}}, v_o \cdot G_1 + r_o \cdot G_2)$.
  3. Sample $(\alpha_x, \alpha_r) \xleftarrow{\$} (\mathsf{F}_p)^2$ and compute:
     - $L_k = \alpha_x \cdot G_{\mathsf{PAY}}$, $V_k = \alpha_x \cdot G_{\mathsf{VRF}}$, $I_k = \alpha_x \cdot H(\mathsf{vk}_o)$
     - $A_k = \alpha_r \cdot G_2$
     - $c_{(k+1) \mod n} = H_{\mathsf{clsag}}(\mathcal{M}||k||L_k||V_k||I_k||A_k)$
  4. Set $i = (k+1) \mod n$ and proceed as follows until $i = k$:
     (a) Sample $(s_i^{(x)}, s_i^{(r)}) \xleftarrow{\$} (\mathsf{F}_p)^2$ and compute:
     - $L_i = s_i^{(x)} \cdot G_{\mathsf{PAY}} + c_i \cdot \mathsf{vk}_i$, $V_i = s_i^{(x)} \cdot G_{\mathsf{VRF}} + c_i \cdot \mathsf{vk}_{\mathrm{VRF},o}$, $I_i = s_i^{(x)} \cdot H(\mathsf{vk}_i) + c_i \cdot \Im_o$
     - $A_i = s_i^{(r)} \cdot G_2 + c_i \cdot D_i$, where $D_i = C_i - T \cdot G_1 - \mathcal{C}$.[a]
     - $c_{i+1 \mod n} = H_{\mathsf{clsag}}(\mathcal{M}||i||L_i||V_i||I_i||A_i)$

     (b) Set $i = (i+1) \mod n$.
  5. Compute $s_k^{(x)} = \alpha_x - c_k \cdot \mathsf{sk}_o$ and $s_k^{(r)} = \alpha_r - c_k \cdot (r_o - r')$.
  6. Output $\sigma_{\mathsf{CLSAG}} = \langle c_0, (s_0^{(x)}, s_0^{(r)}), \ldots, (s_{n-1}^{(x)}, s_{n-1}^{(r)})\rangle$.
- $\mathsf{Verify}(L_{\mathsf{SN}}, x, \mathcal{M}, \sigma) \to \{0, 1\}$:
  1. Parse x as $(T, \mathcal{C}, \mathsf{vk}_{\mathrm{VRF},o}, \Im_o, \mathbb{O}_{\mathsf{block}})$, $\mathbb{O}_{\mathsf{block}}$ as $\langle(\mathsf{vk}_0, C_0), \ldots, (\mathsf{vk}_{n-1}, C_{n-1})\rangle$ and σ_{CLSAG} as $\langle c_0, (s_0^{(x)}, s_0^{(r)}), \ldots, (s_{n-1}^{(x)}, s_{n-1}^{(r)})\rangle$.
  2. Set $c'_0 = c_0$ and, for every $i \in [0, \ldots, n-1]$, compute:
     - $L_i = s_i^{(x)} \cdot G_{\mathsf{PAY}} + c'_i \cdot \mathsf{vk}_i$, $V_i = s_i^{(x)} \cdot G_{\mathsf{VRF}} + c'_i \cdot \mathsf{vk}_{\mathrm{VRF},o}$, $I_i = s_i^{(x)} \cdot H(\mathsf{vk}_i) + c'_i \cdot \Im_o$
     - $A_i = s_i^{(r)} \cdot G_2 + c'_i \cdot D_i$, where $D_i = C_i - T \cdot G_1 - \mathcal{C}$
     - $c'_{(i+1)} = H_{\mathsf{clsag}}(\mathcal{M}||i||L_i||V_i||I_i||A_i)$
  3. If $c_0 = c'_n$, output 1. Otherwise, output 0.

[a] Observe that: $D_k = C_o - T \cdot G_1 - \mathcal{C} = (v_o - T - v_o + T) \cdot G_1 + (r_o - r') \cdot G_2 = (r_o - r') \cdot G_2$.

Figure 2: Modified CLSAG Scheme

Notice that a signature of knowledge of a witness for a statement in L_{SN} is a ring signature for the ring $\langle(\mathsf{vk}_0, C_0), \ldots, (\mathsf{vk}_{n-1}, C_{n-1})\rangle$, as it does not reveal to which (vk_i, C_i) the witness $(\mathsf{sk}_o, v_o, r_o, r')$ corresponds.

Our Modified CLSAG scheme is presented in Figure 2. It is constructed by using standard Schnorr-style sigma protocols for equality and knowledge of discrete logarithms. The proof is then turned into a signature of knowledge via the Fiat-Shamir transformation [FS87] along with the standard OR-composition technique [CDS94] for proving disjunctions among several statements.

3.3 Security Analysis

We now analyze the security of StakeNote, arguing that it achieves the same security guarantees as Ouroboros Praos, although against a weaker delayed adaptive adversary. First, Lemma 1 establishes that the security of the VRF and modified CLSAG schemes is maintained although we reuse the same secret key sk_o to compute the public payment key $\mathsf{vk}_o = \mathsf{sk}_o \cdot G_{\mathsf{PAY}}$, the VRF public key $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$ and the key image $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$.

Lemma 1. *Let $G_{\mathsf{PAY}}, G_{\mathsf{VRF}}, H(\mathsf{vk}_o)$ be generators of a cyclic group $\mathbb{G}$ of order q. The tuples $\mathbf{T_0} = (G_{\mathsf{PAY}}, G_{\mathsf{VRF}}, H(\mathsf{vk}_o), \mathsf{vk}_o = \mathsf{sk}_o \cdot G_{\mathsf{PAY}}, \mathsf{vk}_{VRF,o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}, \Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o))$ where $, \mathsf{sk}_o \xleftarrow{\$} \mathbb{Z}_q$ and $\mathbf{T_1} = (G_{\mathsf{PAY}}, G_{\mathsf{VRF}}, H(\mathsf{vk}_o), \mathsf{vk}_o = \mathsf{sk}_o \cdot G_{\mathsf{PAY}}, \mathsf{vk}_{VRF,o} = \mathsf{sk}_{\mathsf{VRF}} \cdot G_{\mathsf{VRF}}, \Im_o = \mathsf{sk}_{\Im} \cdot H(\mathsf{vk}_o))$ where $\mathsf{sk}_o, \mathsf{sk}_{\mathsf{VRF}}, \mathsf{sk}_{\Im} \xleftarrow{\$} \mathbb{Z}_q$ are computationally indistinguishable.*

Proof. The proof directly follows from the ℓ-DDH Hardness Assumption 1. Given black-box access to an adversary $\mathcal{A}$ that distinguishes $\mathbf{T_0}, \mathbf{T_1}$, we can construct an adversary $\mathcal{A}_{\ell-DDH}$ that solves the ℓ-DDH problem with the same probability.

First, notice that $\mathbf{T_0}$ is distributed as a tuple from D_0 in the ℓ-DDH problem:

$$\mathbf{x_0} := (G_{\mathsf{PAY}}, \mathsf{sk}_o \cdot G_{\mathsf{PAY}}, G_{\mathsf{VRF}} = \beta_0 \cdot G_{\mathsf{PAY}}, H(\mathsf{vk}_o) = \beta_1 \cdot G_{\mathsf{PAY}}, (\mathsf{sk}_o \cdot \beta_0) \cdot G_{\mathsf{PAY}}, (\mathsf{sk}_o \cdot \beta_1) \cdot G_{\mathsf{PAY}})$$

Also, $\mathbf{T_1}$ is distributed as a tuple from D_1 in the ℓ-DDH problem:

$$\mathbf{x_1} := (G_{\mathsf{PAY}}, \mathsf{sk}_o \cdot G_{\mathsf{PAY}}, G_{\mathsf{VRF}} = \beta_0 \cdot G_{\mathsf{PAY}}, H(\mathsf{vk}_o) = \beta_1 \cdot G_{\mathsf{PAY}}, \mathsf{sk}_{\mathsf{VRF}} \cdot G_{\mathsf{PAY}}, \mathsf{sk}_{\Im} \cdot G_{\mathsf{PAY}})$$

$\mathcal{A}_{\ell-DDH}$, uses this correspondence to ℓ-DDH tuples to construct $\mathbf{T_0}, \mathbf{T_1}$ tuples that it inputs to $\mathcal{A}$. Upon receiving a challenge $\mathbf{x} := (G, \alpha \cdot G, \beta_0 \cdot G, \beta_1 \cdot G, \gamma_0 \cdot G, \gamma_1 \cdot G)$, $\mathcal{A}_{\ell-DDH}$ constructs $\mathcal{A}$'s input as follows:

$$(G_{\mathsf{PAY}} = G, G_{\mathsf{VRF}} = \beta_0 \cdot G, H(\mathsf{vk}_o) = \beta_1 \cdot G, \mathsf{vk}_o = \alpha \cdot G, \mathsf{vk}_{\mathrm{VRF},o} = \gamma_0 \cdot G, \Im_o = \gamma_1 \cdot G)$$

$\mathcal{A}_{\ell-DDH}$ outputs whatever $\mathcal{A}$ outputs. Since the input given to $\mathcal{A}$ is distributed exactly as $\mathbf{T_0}$ when the challenge comes from D_0 and exactly as $\mathbf{T_1}$ when the challenge comes from D_1, $\mathcal{A}_{\ell-DDH}$ has the same advantage as $\mathcal{A}$. □

We now show Lemma 2, establishing that parties are elected to generate a block in StakeNote with the same probability as in Ouroboros Praos. This is crucial when later arguing security of StakeNote.

Lemma 2. *In Protocol StakeNote, for every slot sl_r of an epoch e, a party that controls stake v distributed over multiple UTxOs $o_1, \ldots, o_\ell$ with respective amounts $v_{o_1}, \ldots, v_{o_n}$ in the snapshot $\mathbb{O}_e$ for epoch e such that $v = v_{o_1} + \cdots + v_{o_n}$ is considered eligible to generate a block (i.e., passes the check in Step 2 of StakeNote's block generation phase) with the same probability as in Ouroboros Praos.*

Proof. First, notice that the epoch random nonce η_e is generated exactly as in Ouroboros Praos and that the eligibility test checking that $y_o < 2^{l_{\mathsf{VRF}}} \phi_f(\frac{v_o}{v_{\mathsf{total}}})$ for $\mathsf{VRF.Eval}(\mathsf{sk}_o, \eta_e || \mathsf{sl}_r) \to (y_o, \pi_{y_o})$ is exactly the same eligibility test as in Ouroboros Praos. Moreover, notice that the epoch snapshot $\mathbb{O}_e$ for epoch e considers the same set of UTxOs as Ouroboros Praos for epoch e. The only difference in the setup for eligibility check, since the VRF public key $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_o \cdot G_{\mathsf{VRF}}$ is such that it depends on $\mathsf{vk}_o = \mathsf{sk}_o \cdot G_{\mathsf{PAY}}$ and $\Im_o = \mathsf{sk}_o \cdot H(\mathsf{vk}_o)$. However, as established in Lemma 1, this does not introduce any difference in relation to the Ouroboros Praos setup where $\mathsf{vk}_{\mathrm{VRF},o} = \mathsf{sk}_{\mathsf{VRF}} \cdot G_{\mathsf{VRF}}$ is generated from an independently sampled uniformly random secret key $\mathsf{sk}_{\mathsf{VRF}}$.

The crucial detail establishing that the probability of being eligible for generating a block in slot sl_r of epoch e in StakeNote is the same as in Ouroboros Praos is the *aggregation independence* property of function $\phi_f(\frac{v_o}{v_{\mathsf{total}}})$. This property is central in the original security proof of Ouroboros Praos, and guarantees that a party controlling total stake v according to the snapshot $\mathbb{O}_e$ is elected to generate a block in every slot sl_r with the same probability regardless of how v is distributed over multiple UTxOs $o_1, \ldots, o_\ell$ with respective amounts $v_{o_1}, \ldots, v_{o_n}$ such that

$v = v_{o_1} + \cdots + v_{o_n}$. Hence, since the eligibility check is performed exactly as in Ouroboros Praos considering epoch random nonces generated exactly in the same way and exactly the same set of UTxOs in the snapshot, the party is considered eligible to generate a block for every slot with exactly the same probability regardless of how this party's stake is distributed over multiple UTxOs. □

Next, Lemma 3 analyzes the security of the modified CLSAG construction (Figure 2). Although our construction is similar to the CLSAG of [GNB19], we state it as a signature of knowledge, since the SimExt-Security [CL06] guarantees of such schemes will make it easier to analyze our protocol's security in the GKL model [GKL15]. However, our construction also retains the privacy guarantees offered by the original CLSAG. In particular, the simulation soundness implied by SimExt-Security implies that no information about the witness is leaked, hiding the exact UTxO used to generate the signature within the ring.

Lemma 3. *The modified CLSAG construction of Figure 2 is a signature of knowledge for language L_{SN} (Definition 2) with SimExt-Security [CL06] under the ℓ-DDH assumption in the random oracle model.*

Proof. (Sketch) In order to show that the construction Figure 2 is a signature of knowledge with SimExt-Security as defined in [CL06], we must show that it is correct, construct a simulator that generates valid signatures without knowing a witness and construct an extractor that obtains a witness from a valid signature. Correctness follows by inspection, so we will focus on establishing that it is possible to construct a simulator and an extractor for our construction.

First, we observe that our construction is essentially an OR-composition following the techniques of [CDS94] of the Chaum-Pedersen sigma protocol for discrete logarithm equality [CP93] (used to prove that sk_o is used to correctly compute the VRF public key and key image) and of the Schnorr proof of knowledge of discrete logarithms (used to prove knowledge of $r_o - r'$). It is clear that both of these sigma protocols have the completeness, honest verifier zero knowledge and special soundness properties [Dam02], meaning that they can be used to obtain a zero knowledge proof system in the random oracle model via the Fiat-Shamir transform. At the core of this construction is a Fiat-Shamir transform where the message to be signed is given as input to a random oracle when computing the challenge for the underlying sigma protocols.

In order to construct a simulator and an extractor for our scheme, we can leverage the techniques from [FKMV12]. As shown in [FKMV12], zero knowledge proof systems obtained via the Fiat-Shamir transform from sigma protocols are both simulation sound and weakly simulation extractable. The generic construction of a simulator in the simulation soundness proof for the Fiat-Shamir transform presented in [FKMV12] can be directly applied to our construction. The caveat in their proof of the simulation extractability property is that the extractor must rewind a black-box copy of the adversary, which is not allowed in the strong version of simulation extrability. However, in this case, we can rewind the adversary when arguing security of our scheme, so we can use the generic extractor construction of [FKMV12] in order to obtain an extractor for our modified CLSAG. □

To analyze the security of StakeNote (Figure 1), we show that the leader election process determining the parties who generate blocks is computationally indistinguishable from Ouroboros Praos's. Hence, the block distribution generated by corrupted or honest parties in our protocol is essentially the same as Ouroboros Praos. Establishing this fact will allow us to inherit the security properties of Ouroboros Praos under the same assumptions (Theorem 1).

Theorem 1. *Let* VRF.KeyGen, VRF.Eval, VRF.Verify *be the Verifiable Random Functions with unpredictability under malicious key generation from Ouroboros Praos [DGKR18]. Let* (Setup, Sign, Verify) *be a signature of knowledge for the language L_{SN}. StakeNote (cf. Figure 1) satisfies the common prefix, chain growth and chain quality properties (resp. liveness and persistence guarantees) with the same parameters as established in [DGKR18, Theorems 7, 8, 9] (resp. [DGKR18, Theorem 11]) under the ℓ-DDH assumption in the random oracle model with security against a delayed adaptive adversary with corruption delay equal to epoch length over a semi-synchronous network.*
.

Proof. (Sketch) The core argument is that StakeNote elects leaders to generate blocks with the same distribution as in Ouroboros Praos. In the terminology of Ouroboros Praos, we are interested in the characteristic strings denoting empty slots, slots with blocks generated by honest parties and slots with blocks generated by corrupted parties. Each execution is characterized by such a characteristic string according to the blocks that were produced, and the security of Ouroboros Praos is established by proving statements about characteristic strings. Hence, we want to show that the characteristic strings arising from executions of StakeNote are distributed exactly as characteristic strings arising from executions of Ouroboros Praos, except with negligible probability.

First, we observe that the randomness for leader election in StakeNote is generated exactly as in Ouroboros Praos, i.e., by having slot leaders of an epoch execute the VRF-based leaky resettable beacon protocol of Ouroboros Praos in order to generate the randomnes for the next epoch. Moreover, the oblivious leader election scheme adopted by StakeNote is very similar to the one adopted by Ouroboros Praos. A party determines if it is a slot leader by evaluating the VRF on the epoch's randomness concatened with the slot number, then checking if this output is smaller than a function ϕ of the relative stake represented by the amount of each of their UTxOs, where ϕ is defined exactly as in Ouroboros Praos. There are two main differences in StakeNote: 1. a party's total stake (and thus their total relative stake) is unknown and distributed over many different UTxOs, rather than publicly known as in Ouroboros Praos; 2. a slot leader does not reveal the exact UTxO used in leader election (nor its exact value) when proving that they were elected a leader, instead publishing a signature of knowledge showing that a VRF public key (to be used in verifying the proof of leader election) has been correctly computed from a UTxO within a set of UTxOs. We must show that these differences do not affect the distribution of characteristic strings.

As formally established in Lemma 2, the first difference does not affect the distribution of characteristic strings due to the "independent aggregation" property of the ϕ function from Ouroboros Praos. This property ensures that a party gets elected with the same probability regardless of whether their stake is aggregated into a single UTxO or arbitrarily distributed across multiple UTxOs. Hence, even though the leader election check is executed considering the individual amount of each UTxO controlled by a party, that party is guaranteed to be elected with the same probability as if its entire stake was aggregated into a single UTxO.

The second difference between StakeNote and Ouroboros Praos is trickier to analyse, since parties cannot simply verify that a slot leader has been correctly elected as in Praos, as that would require learning the relative stake controlled by the slot leader as well as their VRF public key. Instead, in the case of StakeNote, parties learn a VRF public key attached to the block and a stake *threshold* T that must be guaranteed to respect $T \leq v_o$, where v_o is the stake of the UTxO used to execute the leader election check. Notice that this verification will work as expected (and produce the same results as Ouroboros Praos) as long as the VRF public key and threshold T contained in the block are correctly generated from the secret ket sk_o connected to the UTxO while respecting the condition that $T \leq v_o$. The correctness of the VRF public key is guaranteed by the simulation extractability property of the modified CLSAG construction established in Lemma 3. The fact that $T \leq v_o$ is also guaranteed by the simulation extractability of the modified CLSAG construction, which ensures that the commitment $\mathcal{C}$ is indeed a commitment to $v_o - T$, and by the soundness of the range proof, which guarantees that the value of $\mathcal{C}$ is above zero (i.e., $v_o - T > 0$ implies that $T \leq v_o$).

The last difference between StakeNote and Ouroboros Praos left to analyze is the lack of forward security in the modified CLSAG signature, whereas Praos utilizes a key evolving signature scheme. The lack of forward security means that StakeNote does not achieve security against fully adaptive adversaries. However, if the adversary is only delayed adaptive with corruption delay equal to epoch duration (as we consider), an adversary corrupting a party in one epoch only gains control in the next epoch. Hence, if the adversary requests to corrupt a slot leader who just generated a block, it only gains control of that party in the next epoch, and does not succeed in generating a conflicting version of that block.

We have thus established that, under the assumptions stated in the theorem, the characteristic strings arising from executions of StakeNote are distributed exactly as characteristic strings of executions of Ouroboros Praos, except with negligible probability. Hence, StakeNote achieves

the same properties as Ouroboros Praos with the same parameters. □

3.4 Discussion

CLSAG Our CLSAG proof σ_{CLSAG} satisfies two different purposes. First, as is the original purpose of CLSAG, the block producer proves that they control one of the outputs in the group. Note that, in our case, linkability is not needed since we allow the same private key to produce multiple messages, that is multiple blocks per epoch. Second, it offers a discrete logarithm equality proof, between vk_o and $\mathsf{vk}_{\mathrm{VRF},o}$. This is achieved by CLSAG's elegant idea of reusing the same response $s_i^{(x)}$ for the two columns, that correspond to vk_o and $\mathsf{vk}_{\mathrm{VRF},o}$, effectively showing that the secret of both columns is the same.

Key Image Reuse With our mechanism, a UTxO's key image is published multiple times: (i) every time a block is created using the corresponding UTxO; (ii) when the UTxO is spent. Each of these actions should not reveal the exact UTxO which corresponds to the revealed key image. Therefore, for each of these actions, the party should use an anonymity set of decoys. It is easy to see that, if these sets are different, then the correct UTxO is in their intersection. Therefore, to retain the highest possible level of anonymity, the user should use the same anonymity set at all times. Essentially, the user picks the anonymity set of decoys once, the first time that it publishes the key image, and reuses it for every future action. Note that this does not reduce the anonymity guarantees of the original CryptoNote protocol, since the broader set of outputs, from which the decoys are picked, still consists of all historical outputs.

Block Linkability In our construction, each block is associated with the key image of the output used for its creation. Since this key image is deterministic for each output, all blocks that are created using the same output are publicly linkable. In other words, everyone can deduce that the same output was used for the creation of all blocks that share the same key image, albeit the precise output is hidden within the anonymity set.

4 Implementation

We evaluate StakeNote's with a proof of concept implementation in Rust.[7] Our implementation uses the ristretto255 group on the Curve25519 elliptic curve. Specifically, the VRF implementation uses the `vrf_r255` library,[8] which implements the ECVRF-RISTRETTO255-SHA512 ciphersuite.[9] The CLSAG implementation uses the ristretto255 implementation of the `curve25519_dalek` library.[10] The range proof is constructed with the `bulletproofs` library,[11] which implements Bulletproofs [BBB+18] using ristretto255.

Our evaluation focuses around two main questions:

1. What is the time and size performance of StakeNote?
2. What is the output's amount leakage, due to revealing the threshold T?

4.1 Performance

We evaluated the performance of our mechanism by running 10,000 block creation instances, each simulating the leader election process. First, we created an output with amount of stake between $5 - 45\%$ of the total stake, as well as a set of outputs to be used as decoys. Second, we created a VRF key pair, which shares the same secret key as the chosen output. Third, we picked a random value as the epoch's nonce η_e and iterated over a counter i, which simulated the slot number until the VRF output is below the eligibility threshold.

[7] `https://github.com/commonprefix/stakenote`
[8] `https://docs.rs/vrf-r255/latest/vrf_r255`
[9] `https://github.com/C2SP/C2SP/blob/main/vrf-r255.md`
[10] `https://docs.rs/curve25519-dalek/latest/curve25519_dalek/`
[11] `https://docs.rs/bulletproofs/latest/bulletproofs`

Component	Creation			Verification		
	Min (ms)	**Max (ms)**	**Avg (ms)**	**Min (ms)**	**Max (ms)**	**Avg (ms)**
VRF output	0.369	2.224	0.428	0.22	1.64	0.27
T estimation	0.001	0.066	0.004	-	-	-
Range proof	14.40	61.68	16.53	3.26	19.69	3.95
CLSAG proof	7.04	36.32	8.10	7.42	35.28	8.48
Total	21.44	98.07	24.63	10.68	54.97	12.43

Table 1: Timing evaluation of StakeNote.

Time Performance All timing measurements are shown in Table 1.

During block creation, we first generate of the VRF output and proof, which requires between $0.369 - 2.224$ ms, at an average time of 0.428 ms. Second, we perform a binary search to find the smallest threshold T (with 180 precision decimal points); this is essentially instantaneous (less than 0.1 ms). Third, we compute the range proof; this is the slowest computation, between $14.40 - 61.68$ ms, with average proof creation time 16.53 ms. Fourth, we compute the CLSAG proof, which lasts between $7.04 - 36.32$ ms with an average time of 8.10 ms. In total, block creation lasts between $21.44 - 98.07$ ms (average 24.63 ms).

Block verification is significantly faster. VRF verification lasts between $0.22 - 1.64$ ms, with the average being 0.27 ms. Verification of the range proof lasts between $3.26 - 19.69$ ms, at an average time of 3.95 ms. CLSAG verification is slower, requiring between $7.42 - 35.28$ ms with an average time being 8.48 ms.

As the average values suggest, block creation typically lasts less than 25 ms and verification less than 15 ms. The evaluations took place on home equipment, so timings occasionally increase e.g., during increased load to the CPU, although never exceeding 100 ms for creation and 55 ms for verification. We also note that our code is not optimized beyond using Rust's standard "release" compiling profile.

Component		Size (bytes)
VRF	Public key	32
	Output	128
	Proof	80
Range proof	Threshold value T	8
	Bulletproof	672
CLSAG	Key Image	32
	$(v_o - T)$ commitment	32
	c_0	32
	$s_i^{(x)}$ *(per output in the ring)*	32
	$s_i^{(r)}$ *(per output in the ring)*	32
	public key *(per output in the ring)*	32
	amount commitment *(per output in the ring)*	32
Total		$1{,}064 + n \cdot 128$

Table 2: Proof size evaluation of StakeNote with ring size n.

Size Performance The size measurements shown in Table 2.

Assuming the CLSAG ring size is n, the block consists of the following: (i) $2 + 2 \cdot n$ elliptic curve points; (ii) $2 + 2 \cdot n$ scalars; (iii) one 256-bit number; (iv) VRF output and proof; (v) range proof. Naturally, these depend on the primitives' implementation, so the following are reference values.

Regarding the VRF elements, the public key is 32 bytes, the output is 128 bytes, and the proof is 80 bytes. The range proof is again the heaviest element, requiring 672 bytes. The value T is 8 bytes, the key image is 32 bytes, and the commitment to the difference between the output's amount and T is 32 bytes. Regarding the CSLAG proof, the value c_0 is 32 bytes, whereas each

pair $(s_i^{(x)}, s_i^{(r)})$ is $(32, 32)$ bytes. Additionally, each output amounts to 32 bytes for the public key plus 32 bytes for the amount's commitment. In total, the StakeNote proofs require 1,064 bytes plus 128 bytes for each output in the ring.

4.2 Output Amount Leakage

Our construction reveals a lower bound T of the used output's stake amount. Specifically, the construction reveals that *one of* the outputs in the anonymity set has *some amount* which is higher than T. Therefore, a relevant question is how close T is to the real stake amount of the correct output, i.e., how much does T reveal regarding the correct output's amount.

Since the VRF output is chosen from a uniform distribution, the value T should also be uniform, albeit with the extra restriction of being smaller than the output's real amount. We experimentally verify this intuition as follows. We ran 10,000 block creation instances, each simulating the leader election process, for various stake amounts (between $5 - 45\%$ of the total stake). In each instance, we used a binary search to find the smallest (180 decimal point precision) threshold, for which the leader eligibility inequality holds. Next, we measured this threshold as a percentage of the output's real amount. These percentages form the distribution of observed values for the threshold T.

To evaluate how close this distribution is to the uniform distribution, we performed a series of tests: (i) χ^2 test: $1{,}052 - 1{,}141$; (ii) $p-$value: $0.00115 - 0.0973$; (iii) total variation: $0.128 - 0.134$; (iv) max deviation: $0.0011 - 0.0013$. Under all tests, the observed distribution of T values is indeed very close to uniform.

5 Practical Enhancements

Although the protocol of Section 3 is sound, it presents two potential practical drawbacks. First, blocks created with the same output are linkable via the key image. Second, the protocol's security hinges on half of *all* stake participating honestly, thus not allowing honest regular users to abstain from consensus. In this section we discuss practical enhancements that address these concerns.

5.1 Block Unlinkability

In Ouroboros Praos, the stake distribution snapshot is taken before the epoch starts, so only the VRF keys that belong in the snapshot are eligible for leader election. In CryptoNote this is not possible, since outputs are not verifiably spent. A CryptoNote transaction defines a set of "decoy" outputs (the anonymity set) and the user proves that the output o is eligible to be spent, while hiding o within the anonymity set, such that no party can identify whether o is unspent.

This creates a problem: when using a UTxO o to create a block, how can one prove that o was part of the snapshot? In StakeNote of Section 3, we utilize *key images*. However, since key images are deterministic, they link all blocks created by the same output. To avoid block linkability, we need to avoid using the output's key image multiple times, or not use it in the block's proof altogether.

The first practical option is to minimize the times when a key image is used. Essentially, this is achievable by using each UTxO only once for block production and consuming it immediately after the first use. In that case the UTxO's key image is used exactly twice: once for producing a block and once for spending the output. In practice, a privacy-oriented participant can split their staking amount across many outputs such that, on expectation, each output produces at most one block per two epochs.

An alternative approach is to explicitly specify which outputs are eligible for consensus participation, ensuring that these output are identifiably spent. This is achievable by adding an extra bit to every output, which signifies whether it is a regular payment output or a "staking" output. To consume a staking output, the party does not use any decoys, so the exact point when each staking output is spent is publicly identifiable. Additionally, staking outputs cannot be used as decoys for regular payment transactions, while also payment outputs cannot be used

as decoys for block creation. Essentially, the system's stake is temporarily split between payment and staking outputs, so each epoch's snapshot consists of unspent staking outputs.

5.2 Dynamic Participation via Estimated Stake

Enabling dynamic participation is more challenging, because it requires changes on the protocol level. Here we sketch a mechanism which allows to infer participating stake by observing the block production rate of every epoch.

The protocol of Figure 1 makes use of the global variable v_{total}, i.e., the total stake in the system. Now, we replace v_{total} with a per-epoch total-stake value $v_{(e)}$, assuming an initial value $v_{(0)}$ for the first two epochs. Each epoch lasts R slots. At the beginning of epoch e, the protocol counts the number of blocks $B_{(e-2)}$ that were added to the canonical chain during epoch $e-2$ and estimates the stake used by the leader election in e as:

$$v_{(e)} = v_{(e-2)} \cdot \frac{B_{(e-2)}}{f \cdot R}$$

where $f \cdot R$ is the expected number of blocks per epoch when both (i) $v_{(e)}$ matches the true participating stake and (ii) all participating stake is honest. Therefore, in the per-output eligibility threshold of Figure 1, v_{total} is replaced with $v_{(e)}$.

Discussion The base protocol of Section 3 is secure under the standard Praos assumption that the honest parties control more than half of the total stake. Give the above stake estimation mechanism, this assumption changes such that, in every epoch, the honest parties control more than half of the *participating* stake.

Intuitively, the new variant's security holds as follows. For ease of readability and without loss of generality, we assume that the honest and adversarial stake amounts remain fixed throughout the epochs discussed next.

Assume that the protocol's execution is secure up to epoch e. On e, the inferred participating stake is $v_{(e)}$. Let v_H and $v_{\mathcal{A}}$ denote the honest and adversarial participating stake respectively on epoch e, with $v_H > v_{\mathcal{A}}$ by assumption.

If all participating stake $v_H + v_{\mathcal{A}}$ was honest, then block production would be ideal and the estimated stake for epoch $e+2$ would be: $v_{(e+2)} = v_{(e)} \cdot \frac{B_{(e-2)}}{f \cdot R} = v_{(e)}$.

However, the adversary can choose to withhold blocks arbitrarily, in order to artificially change the stake estimation. Remember that each output is elected per slot with probability proportional to its stake divided by $v_{(e)}$ and, due to aggregation independence, the election probability of each party depends only on its stake amount, not on how it is spread across multiple UTxOs. Therefore, the honest parties create on expectation $f \cdot R \cdot \frac{v_H}{v_{(e)}}$ blocks during epoch e, whereas the adversary creates $f \cdot R \cdot \frac{v_{\mathcal{A}}}{v_{(e)}}$ blocks.

By withholding an arbitrary amount of blocks, the adversary $\mathcal{A}$ can bias the estimated stake participation for epoch $e+2$. Specifically, $\mathcal{A}$ can choose for $B_{(e)}$ any value in the range: $[f \cdot R \cdot \frac{v_H}{v_{(e)}}, f \cdot R]$. Consequently, the estimated stake for epoch $e+2$ can take any value in the range: $[v_{(e)} \cdot \frac{v_H}{v_{(e)}}, v_{(e)}]$. Notice that, by assumption, it holds that $v_H > v_{\mathcal{A}}$. Therefore, the final range of estimated stake for epoch $e+2$ is: $[\frac{v_{(e)}}{2}, v_{(e)}]$.

For any value v in this interval, the per-output threshold rescales honest and adversarial outputs identically. Therefore, the ratio of expected honest to adversarial block production remains $\frac{v_H}{v_{\mathcal{A}}}$. Since the security argument of Ouroboros Praos depends on this ratio rather than on absolute block rates, its security should intuitively carry over, with one caveat: by lowering the threshold for block eligibility, more blocks are created on expectation and, consequently, the probability of forks increases. Therefore, the frequency of slots with multiple honest leaders also increases, hence chain growth is slightly hurt. Nonetheless, the active slots coefficient f can be set small enough such that, even under the artificially lowered stake estimation, the probability of multiple honest leaders per slot (which result in honest forks) does not affect security. A full formal analysis of this mechanism, including the interaction with network delay and chain-quality losses due to honest forks, is left for future work.

Acknowledgements

We thank Andrey Sabelnikov and Sowle for their helpful discussions, questions, and feedback. The project was partially funded by Zano and Common Prefix.

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018.

[BMSZ20] Foteini Baldimtsi, Varun Madathil, Alessandra Scafuro, and Linfeng Zhou. Anonymous lottery in the proof-of-stake setting. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 318–333. IEEE Computer Society Press, 2020.

[CBC+24] Miranda Christ, Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Deepak Maram, Arnab Roy, and Joy Wang. Sok: Zero-knowledge range proofs. In Rainer Böhme and Lucianna Kiffer, editors, *6th Conference on Advances in Financial Technologies, AFT 2024, September 23-25, 2024, Vienna, Austria*, volume 316 of *LIPIcs*, pages 14:1–14:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Berlin, Heidelberg, August 1994.

[CGM23] Sanaz Chamanara, S Arman Ghaffarizadeh, and Kaveh Madani. The environmental footprint of bitcoin mining across the globe: Call for urgent action. *Earth's Future*, 11(10):e2023EF003871, 2023.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Berlin, Heidelberg, August 2006.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Berlin, Heidelberg, August 1993.

[Dam02] Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 84, 2002.

[DGKR18] 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, Cham, April / May 2018.

[Dig25] Digiconomist. Bitcoin energy consumption index, 2025.

[fAF25] Cambridge Centre for Alternative Finance. Cambridge bitcoin electricity consumption index, 2025.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Cham, December 2019.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 181–200. Springer, Berlin, Heidelberg, April 2007.

[GKL15] 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, Berlin, Heidelberg, April 2015.

[GM19] Ian Goldberg and Tyler Moore, editors. *FC 2019*, volume 11598 of *LNCS*. Springer, Cham, February 2019.

[GNB19] Brandon Goodell, Sarang Noether, and Arthur Blue. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Paper 2019/654, 2019.

[GOT19] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 690–719. Springer, Cham, May 2019.

[GS24] Emanuele Giunta and Alistair Stewart. Unbiasable verifiable random functions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 142–167. Springer, Cham, May 2024.

[HH19] Abraham Hinteregger and Bernhard Haslhofer. Short paper: An empirical analysis of monero cross-chain traceability. In Goldberg and Moore [GM19], pages 150–157.

[KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 153–173. Springer, Cham, September 2017.

[KKKZ19] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy*, pages 157–174. IEEE Computer Society Press, May 2019.

[KMNS21] Markulf Kohlweiss, Varun Madathil, Kartik Nayak, and Alessandra Scafuro. On the anonymity guarantees of anonymous proof-of-stake protocols. In *2021 IEEE Symposium on Security and Privacy*, pages 1818–1833. IEEE Computer Society Press, May 2021.

[KRDO17] 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, Cham, August 2017.

[Maz15] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 32(4):1–45, 2015.

[MM18] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *PoPETs*, 2018(2):105–121, April 2018.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, July 2018.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[NM+16] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *ledger*, 1:1–18, 2016.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.

[PBF+19] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 43–63. Springer, Berlin, Heidelberg, March 2019.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992.

[VS13] Nicolas Van Saberhagen. Cryptonote v 2.0. 2013.

[WLS+19] Dimaz Ankaa Wijaya, Joseph K. Liu, Ron Steinfeld, Dongxi Liu, and Jiangshan Yu. On the unforkability of monero. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *ASIACCS 19*, pages 621–632. ACM Press, July 2019.

[WPG+22] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2022.

[WPNM23] Chenghong Wang, David Pujol, Kartik Nayak, and Ashwin Machanavajjhala. Private proof-of-stake blockchains using differentially-private stake distortion. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 1577–1594. USENIX Association, August 2023.

[YAV19] Jiangshan Yu, Man Ho Allen Au, and Paulo Jorge Esteves Veríssimo. Re-thinking untraceability in the CryptoNote-style blockchain. In Stephanie Delaune and Limin Jia, editors, *CSF 2019 Computer Security Foundations Symposium*, pages 94–107. IEEE Computer Society Press, 2019.

[YAY+19] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. New empirical traceability analysis of CryptoNote-style blockchains. In Goldberg and Moore [GM19], pages 133–149.