Single Secret Leader Election

Dan Boneh[1], Saba Eskandarian[1], Lucjan Hanzlik[1,2], and Nicola Greco[3]

[1] Stanford University
[2] CISPA Helmholtz Center for Information Security
[3] Protocol Labs

Abstract. In a *Single Secret Leader Election (SSLE)*, a group of participants aim to randomly choose exactly one leader from the group with the restriction that the identity of the leader will be known to the chosen leader and nobody else. At a later time, the elected leader should be able to publicly reveal her identity and prove that she has won the election. The election process itself should work properly even if many registered users are passive and do not send any messages. Among the many applications of SSLEs, their potential for enabling more efficient proof-of-stake based cryptocurrencies have recently received increased attention.

This paper formally defines SSLE schemes and presents three constructions that provide varying security and performance properties. First, as an existence argument, we show how to realize an ideal SSLE using indistinguishability obfuscation. Next, we show how to build SSLE from low-depth threshold fully homomorphic encryption (TFHE) via a construction which can be instantiated with a circuit of multiplicative depth as low as 10, for realistically-sized secret leader elections. Finally, we show a practical scheme relying on DDH that achieves a slightly relaxed notion of security but which boasts extremely lightweight computational requirements.

1 Introduction

Leader election is a question of fundamental importance in the distributed consensus literature and has for decades been the subject of academic study. The meteoric rise of blockchains in both academic and industry settings [41], however, has motivated a host of new research questions and motivated renewed enthusiasm for combining privacy with consensus applications. For example, a number of recent works have studied *secret* leader election in the context of Proof of Stake (PoS) blockchains [7, 9, 43], where the identity of a randomly chosen leader remains secret until she reveals herself as the leader [3, 26, 29, 35]. The added secrecy guarantee defends against several attacks that could otherwise compromise liveness of the blockchain. For example, once a leader is selected, an attacker could mount a Denial of Service (DoS) attack on the chosen leader and prevent her for publishing a block. The system would then need to select an alternate leader, who might also get attacked before publishing a block, and so on, thereby halting the system. Secret leader election solves this issue by ensuring that the identity of the leader remains hidden until the leader publishes a new block.

Existing proposals for secret leader election work by electing a few potential leaders *in expectation* and describing a simple run-off procedure such that one of the potential leaders can be recognized as the absolute winner of the election after all potential leaders have revealed themselves. The possibility of several potential leaders, however, can lead to wasted effort and potentially even forks in the blockchain in case of attacks on the run-off procedure.

This situation has led to a desire for a new approach to secret leader election that guarantees that one, and only one, leader obtains a valid proof that it won the election [38]. In response, this paper formally defines and constructs a *Single Secret Leader Election* (SSLE). In a SSLE scheme, a group of users register to participate in a series of elections. Each election chooses exactly one leader; the leader knows that she was selected, but all other users only learn her identity once she reveals herself as the leader, along with a proof that she was indeed selected by the protocol. A variant of the basic scheme may ask for an ordered list of leaders, say 10 chosen leaders, to learn their position in the chosen list along with a proof of that position, but learn nothing else about the list. Practical deployments additionally require restrictions on computation, communication, and (most importantly) storage costs of such a protocol.

1.1 Our Contributions

This paper formally defines and constructs SSLE schemes. We begin by describing both the practical and theoretical requirements of an SSLE scheme before developing a syntax and a set of security definitions that formally capture these requirements, the paper's first core contribution. Along the way, we describe an important "straw man" solution that does not satisfy our full security definitions, but may suffice for some applications.

It is not difficult to see that SSLE can be constructed from general multiparty computation. However, an MPC protocol where all parties must send one or more messages can easily be disrupted by an attacker who can take a single participant offline. For protection against denial of service as well as to minimize communication and storage costs, elections must take place even if a large subset of users send *no messages* for each election. We wish to construct schemes where users only send a single message to register as participants for many consecutive elections. As we shall see, two of our schemes require mostly zero messages per-election, once a user has registered.

Our second core contribution consists of SSLE constructions from three different classes of cryptographic assumptions and an exploration of the security and performance tradeoffs associated with each approach. We begin by showing feasibility of constructing an ideal SSLE scheme through a construction relying on indistinguishability obfuscation [5, 27]. Next, we show how to build an SSLE scheme from LWE [44] using threshold FHE [11] with a very low-depth leader election circuit. Finally, we give a construction relying on the Decision Diffie-Hellman (DDH) assumption and random shuffles [33, 34] whose security and performance properties may suffice for practical use-cases. The latter two constructions are proven secure in the random oracle model [6, 25]. In addition to proving the security of each construction, we discuss practical considerations associated with deploying them and cover a number of variations to tailor them to applications with a range of requirements and constraints. We briefly summarize each approach below.

SSLE from indistinguishability obfuscation. Our first and simplest solution from indistinguishability obfuscation [5, 27] serves to show the feasibility of constructing an SSLE scheme as we define it and gives an example of a scheme that demonstrates all the qualitative properties one could want from an SSLE scheme. The construction involves obfuscating a program that takes as input all the participants' public keys and outputs a commitment to each user indicating whether that user is the leader as well as a ciphertext encrypted to each user that holds the randomness used for that user's commitment. The winner is chosen by evaluating a puncturable PRF [15, 17, 36] on public randomness, with the PRF key hidden inside the obfuscated program.

SSLE from threshold FHE. Next, we construct an SSLE scheme based on threshold FHE [11]. The core idea is for each user to post an encryption of a secret s_i when they register to participate and use computation under the FHE with the public randomness as input to select one string s_i from the registered set. Since only the user who generated s_i knows her secret, only she learns that she is the leader. As long as a threshold number of users are available to publish a partial decryption, the election will succeed even if some users are offline due to an active DoS attack. Given this high-level approach, the main technical challenge lies in choosing s_i with a circuit that has low multiplicative depth in order to save on computational costs and ciphertext size. We show how to achieve depth of as little at 10 AND gates by combining low-depth block ciphers with a technique for efficiently expanding $\log N$ bits of randomness to a length N vector with zeros in every position except for a single 1.

SSLE from DDH and shuffles. Our final and most lightweight construction assumes only the hardness of DDH in some group. Instead of encrypting each user's string s_i as we do with the FHE-based solution, we hide the link between each user and his or her s_i by shuffling s_i into the database of secrets at registration time. A Naïve approach requires shuffling a set of size N, the number of participants, whenever a new user joins and also posting proofs that the shuffle was carried out correctly. We show how to eliminate the need for a proof and reduce the shuffle to a set of size $\sqrt{N}$ at the cost of some degradation of the resulting security property. The resulting balance of performance and security offers a tradeoff well-suited to real-world applications.

1.2 Related Work

An RFP published by Protocol Labs [38] informally describes SSLE and gives a sketch of a solution from functional encryption [13, 42] that roughly satisfies their requirements. Unfortunately, this scheme requires a new trusted setup phase each time the set of participants in an election changes. Although we are the first to formally consider *single* secret leader election, secret leader election in the case without the strict requirement of electing a single leader has been studied extensively in prior work, especially in the context of Proof of Stake [7, 9, 43] blockchain applications. These approaches potentially elect multiple leaders and then suggest ways to pick one leader from among the set once the set has been made public.

Ganesh et al. [26] and Ouroboros Crypsinous [35] extend previous proof of stake systems in the Ouroboros family [4,22,37] to consider privacy-preserving proof of stake. Algorand [29] and Fantomette [3] both introduce secret leader election protocols as part of their overall proof of stake systems as well. Their approaches center around evaluating a VRF and checking if the output for each user falls near or below a target threshold. This mechanism filters out most potential leaders. Then the few remaining potential leaders reveal themselves and choose the final leader with a simple tie-breaker, e.g. lowest VRF output. The downside of this approach is that the leader does not know that she was selected, until everyone else reveals their values. Moreover, if the final leader's messages do not reach all nodes in the network, those nodes may incorrectly conclude that a different leader was elected, causing the chain to fork. This cannot happen in an election scheme that guarantees electing exactly one leader. In Section 3 we will present a similar scheme that does not rely on VRFs on our way to formalizing security requirements for SSLE.

Finally, Zether [18] proposes a privacy-preserving proof of stake that hides both the winner(s) of an election and each user's stake as an application of their techniques.

2 Preliminaries

Notation. Let $x \leftarrow F(y)$ denote the assignment of the output of $F(y)$ to x, and let $x \xleftarrow{R} S$ denote assignment to x of an element sampled uniformly random from set S. We use λ to refer to a security parameter and sometimes omit it if its presence is implicit. The notation $[k]$ represents the set of integers $1, 2, ..., k$, and $\emptyset$ denotes the empty set. We use $\mathcal{A}^H$ to denote that $\mathcal{A}$ has oracle access to some function H. A function $\mathsf{negl}(x)$ is *negligible* if for all $c > 0$, there is a x_0 such that for all $x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We omit x if the parameter is implicit. PPT stands for probabilistic polynomial time. Finally, we allow algorithms to output $\perp$ to indicate failure. When referring to a function with some input fixed, we use $\cdot$ in the place of other parameters, e.g. $f(x, \cdot)$.

Standard Primitives. We use a number of standard cryptographic tools throughout the paper, including PRFs, weak PRFs, CPA-secure PKE, commitment schemes, and the DDH assumption. Definitions of these tools appear in Appendix A.

Randomness Beacons. Each election in all our constructions uses a fresh public randomness R generated for that election. A number of works study how to generate such randomness, with approaches ranging from harnessing randomness from financial data (e.g. markets, cryptocurrencies) [8, 16, 21] to cryptographic delay functions [10, 39], and a number of systems have been built to provide reliable public randomness [20, 31, 46]. The chosen source of public randomness for a particular instantiation of our schemes is orthogonal to our work, so we do not specify a particular means to generate the random input R used in our elections.

Indistinguishability obfuscation. Our first construction, intended to show the feasibility of satisfying our definitions of SSLE, makes use of indistinguishability obfuscation [5, 27], defined as follows.

Definition 1 (Indistinguishability obfuscator ($i\mathcal{O}$)). *A uniform PPT machine $i\mathcal{O}$ is called an* indistinguishability obfuscator *for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:*

- *For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that $Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$.*

– *For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: for all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then $|Pr[D(i\mathcal{O}(\lambda, C_0)) = 1] - Pr[D(i\mathcal{O}(\lambda, C_1)) = 1]| \leq \alpha(\lambda)$.*

Puncturable PRFs. Our obfuscation-based construction in Section 4 also relies on puncturable PRFs [15, 17, 36], defined below. Puncturable PRFs behave as regular PRFs, except their keys can be *punctured* such that a punctured key cannot be evaluated at one point in the PRF's domain.

Definition 2. *A* puncturable *family of PRFs F is given by a triple of algorithms (Key_F, $\mathsf{Puncture}_F$, Eval_F), and a pair of computable functions $n(\cdot)$ and $m(\cdot)$ satisfying the following conditions:*

– *Functionality preserved under puncturing. For every PPT adversary $\mathcal{A}$ such that $\mathcal{A}(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$, for all $x \in \{0,1\}^{n(\lambda)}$ where $x \notin S$, we have that*

$$Pr[\mathsf{Eval}_F(K, x) = \mathsf{Eval}_F(K_S, x) : K \leftarrow \mathsf{Key}_F(1^\lambda), K_S = \mathsf{Puncture}_F(K, S)] = 1.$$

– *Pseudorandom at punctured points. For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$ and state σ, consider an experiment where $K \leftarrow \mathsf{Key}_F(1^\lambda)$ and $K_S \leftarrow \mathsf{Puncture}_F(K, S)$. Then we have*

$$|Pr[\mathcal{A}_2(\sigma, K_S, S, \mathsf{Eval}_F(K, S)) = 1] - Pr[\mathcal{A}_2(\sigma, K_S, S, U_{m(\lambda)\cdot|S|}) = 1]| = \mathsf{negl}(\lambda)$$

where $\mathsf{Eval}_F(K, S)$ denotes the concatenation of $\mathsf{Eval}_F(K, x_1), ..., \mathsf{Eval}_F(K, x_k)$ where $S = \{x_1, ..., x_k\}$ is the enumeration of the elements of S in lexicographic order and U_l denotes the uniform distribution over l bits.

For ease of notation, we write $F(K, x)$ to represent $\mathsf{Eval}_F(K, x)$. We also represent the punctured key $\mathsf{Puncture}_F(K, S)$ by $K(S)$.

Threshold FHE. In Section 5 we present an SSLE construction relying on a low-depth threshold FHE (TFHE) [11]. A threshold FHE allows computation on encrypted data as well as threshold decryption of ciphertexts, where a threshold number of key holders must come together to decrypt any ciphertext. We modify the standard syntax and security definitions of TFHE to allow the encryption algorithm to additionally output a proof of knowledge π of the encrypted plaintext. This can be done by combining a traditional TFHE scheme with standard NIZK techniques (e.g. in the random oracle model).

Definition 3 (Threshold fully homomorphic encryption (TFHE) [11]). *Let $P = \{P_1, ..., P_N\}$ be a set of parties and let $\mathcal{S}$ be a class of efficient access structure on P. A threshold fully homomorphic encryption scheme for $\mathcal{S}$ is a tuple of PPT algorithms $\mathsf{TFHE} = (\mathsf{TFHE.Setup}, \mathsf{TFHE.Encrypt}, \mathsf{TFHE.Eval}, \mathsf{TFHE.PartDec}, \mathsf{TFHE.FinDEec})$ with the following properties:*

– $\mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathcal{A}) \rightarrow (\mathsf{pk}, \mathsf{sk}_1, ..., \mathsf{sk}_N)$*: On input the security parameter λ, a depth bound d, and an access structure $\mathcal{A}$, output a public key pk, and a set of secret key shares $\mathsf{sk}_1, ..., \mathsf{sk}_N$.*

– $\mathsf{TFHE.Encrypt}(\mathsf{pk}, \mu) \rightarrow (\mathsf{ct}, \pi)$*: On input a public key pk, and a plaintext $\mu \in \mathbb{F}_2^n$ for $n = \mathsf{poly}(\lambda)$, the encryption algorithm outputs a ciphertext ct and a proof π. $\mathsf{Encrypt}$ can optionally take a third parameter r, the randomness to be used for encryption.*

– $\mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, ..., \mathsf{ct}_k) \rightarrow \hat{\mathsf{ct}}$*: On input a public key pk, circuit $C : \mathbb{F}_2^{n \times k} \rightarrow \mathbb{F}_2^n$ of depth at most d, and a set of ciphertexts $\mathsf{ct}_1, ..., \mathsf{ct}_k$, the evaluation algorithm outputs a ciphertext $\hat{ct}$.*

– $\mathsf{TFHE.PartDec}(\mathsf{pk}, \mathsf{ct}, \mathsf{sk}_i) \rightarrow \mathsf{p}_i$*: On input a public key pk, a ciphertext ct, and a secret key share sk_i, the partial decryption algorithm outputs a partial decryption p_i related to the party P_i.*

– $\mathsf{TFHE.FinDec}(\mathsf{pk}, \mathsf{ct}, B) \rightarrow \hat{\mu}$*: On input a public key pk, ciphertext ct, and a set $B = \{\mathsf{p}_i\}_{i \in S}$ for some $S \subseteq \{P_1, ..., P_N\}$ the final decryption algorithm outputs a plaintext $\hat{\mu} \in \mathbb{F}_2^n \cup \perp$.*

– $\mathsf{TFHE.Verify}(\mathsf{pk}, \mathsf{ct}, \pi) \rightarrow 1/0$*: On input a public key pk, ciphertext ct, and proof π this algorithm accepts or rejects the proof π for the given ciphertext.*

– $\mathsf{TFHE.VerifyDec}(\mathsf{pk}, \mathsf{p_i}, \mathsf{ct}) \to 1/0$: *On input a public key* pk*, a partial decryption* $\mathsf{p_i}$*, and a ciphertext* ct*, this algorithm accepts or rejects the partial decryption.*

In order for a TFHE to be considered secure for our purposes, it must satisfy the compactness, correctness, robustness, semantic security, plaintext extractability, and simulation security definitions below.

Definition 4 (Compactness [11]). *We say that a* TFHE *scheme is compact if there exists polynomials* $\mathsf{poly}_1(\cdot)$ *and* $\mathsf{poly}_2(\cdot)$ *such that for all* λ*, depth bound* d*, circuit* $C : \mathbb{F}_2^{n\times k} \to \mathbb{F}_2^n$ *of depth at most* d*, access structure* $\mathbb{A}$*, and* $\mu \in \mathbb{F}_2^n$*, the following holds. For* $(\mathsf{pk}, \mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$*,* $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Encrypt}(\mathsf{pk},\mu_i)$ *for* $i \in [k]$*,* $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$*,* $p_j \leftarrow \mathsf{TFHE.PartDec}(\mathsf{pk},\mathsf{ct},\mathsf{sk}_j)$ *for* $j \in [N]$*, we have that* $|\hat{ct}| \le \mathsf{poly}(\lambda, d)$ *and* $|p_j| \le \mathsf{poly}(\lambda, d, N)$*.*

Definition 5 (Correctness [11]). *We say that a* TFHE *scheme satisfies evaluation correctness if for all* λ*, depth bound* d*, access structure* $\mathbb{A}$*, circuit* $C : \mathbb{F}_2^{n\times k} \to \mathbb{F}_2^n$ *of depth at most* d*,* $S \in \mathbb{A}$*, and* $\mu_i \in \mathbb{F}_2^n$ *for* $i \in [k]$*, the following condition holds. For* $(\mathsf{pk}, \mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$*,* $(\mathsf{ct}_i, \pi_i) \leftarrow \mathsf{TFHE.Encrypt}(\mathsf{pk},\mu_i)$ *and* $\mathsf{TFHE.Verify}(\mathsf{pk}, \mathsf{ct}_i, \pi_i)=1$ *for* $i \in [k]$*,* $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1,...,\mathsf{ct}_k)$*,*

$$\Pr\Big[\mathsf{TFHE.FinDec}(\mathsf{pk}, \hat{ct}, \{\mathsf{TFHE.PartDec}(\mathsf{pk},\mathsf{ct},\mathsf{sk}_i)\}_{i\in S}) = C(\mu_1, ..., \mu_k)\Big] \ge 1 - \mathsf{negl}(\lambda).$$

Moreover, we additionally require that

$$\Pr\Big[D \leftarrow \{\mathsf{TFHE.PartDec}(\mathsf{pk},\mathsf{ct},\mathsf{sk}_i)\}_{i\in S} : \{\mathsf{TFHE.VerifyDec}(\mathsf{pk}, D_i, \mathsf{ct}) = 1\}_{i\in S}\Big] = 1.$$

Definition 6 (Robustness [11]). *We say that a* TFHE *scheme satisfies robustness if for all* λ*, and depth bound* d*, the following holds. For all PPT adversaries* $\mathcal{A}$*, the following experiment* $\mathsf{Expt}_{\mathcal{A},\mathsf{TFHE.rob}}(1^\lambda, 1^d)$ *outputs* 1 *with negligible probability:*

$\mathsf{Expt}_{\mathcal{A},\mathsf{TFHE.rob}}(1^\lambda, 1^d)$*:*

1. *On input the security parameter* 1^λ *and a circuit depth* 1^d*, the adversary* $\mathcal{A}$ *outputs messages* $\mu_1, \ldots, \mu_k$*, a circuit* $C : \mathbb{F}_2^{n\times k} \to \mathbb{F}_2^n$ *of depth at most* d *and an access structure* $\mathbb{A}$*.*
2. *The challenger runs* $(\mathsf{pk},\mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ *and provides* $(\mathsf{pk},\mathsf{sk}_1,...,\mathsf{sk}_N)$ *and ciphertext* $\mathsf{ct} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk},C,\mathsf{ct}_1,...,\mathsf{ct}_k)$ *to* $\mathcal{A}$*, where* $(\mathsf{ct}_i,\pi_i) \leftarrow \mathsf{TFHE.Encrypt}(\mathsf{pk}, \mu_i)$ *for each* $i \in [k]$*.*
3. $\mathcal{A}$ *outputs a two sets* $S_1 = \{\mathsf{p}_1, ..., \mathsf{p}_t\}$ *and* $S_2 = \{\mathsf{p}'_1, ..., \mathsf{p}'_t\}$ *of partial decryptions.*
4. *The experiment outputs* 1 *iff:*
   – $\mathsf{TFHE.VerifyDec}(\mathsf{pk}, \mathsf{p}, \mathsf{ct}) = 1$ *for all* $\mathsf{p} \in \{S_1, S_2\}$*, and*
   – $\mathsf{TFHE.FinDec}(\mathsf{pk}, \mathsf{ct}, S_1) \neq \mathsf{TFHE.FinDec}(\mathsf{pk}, \mathsf{ct}, S_2)$*.*

Definition 7 (Semantic security [11]). *We say that a* TFHE *scheme satisfies semantic security if for all* λ*, and depth bound* d*, the following holds. For any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathsf{Expt}_{\mathcal{A},\mathsf{TFHE.sem}}(1^\lambda, 1^d)$ *outputs* 1 *with negligible probability:*

$\mathsf{Expt}_{\mathcal{A},\mathsf{TFHE.sem}}(1^\lambda, 1^d)$*:*

1. *On input the security parameter* 1^λ *and a circuit depth* 1^d*, the adversary* $\mathcal{A}$ *outputs* $\mathbb{A} \in S$*.*
2. *The challenger runs* $(\mathsf{pk},\mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ *and provides* pk *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *outputs a set* $S \subseteq \{P_1, ..., P_N\}$ *such that* $S \notin \mathbb{A}$ *as well as messages* $m_0, m_1 \in \mathbb{F}_2^n$*.*
4. *The challenger provides* $\{\mathsf{sk}_i\}_{i\in S}$ *along with* $\mathsf{TFHE.Encrypt}(\mathsf{pk},m)$ *for* $m \xleftarrow{R} \{m_0, m_1\}$ *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *outputs a guess* m'*. The experiment outputs* 1 *if* $m = m'$*.*

Definition 8 (Plaintext Extractability). *We say that a* TFHE *scheme is plaintext extractable if for all* λ*, depth bound* d*, and access structure* $\mathbb{A}$ *the following holds. There exists a PPT extraction algorithm* $\mathcal{E}$ *such that for all PPT adversaries* $\mathcal{A}$*, algorithm* $\mathcal{E}$ *interacts with* $\mathcal{A}$ *(e.g., emulating its random oracle) so that:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})\ ;\ (\mathsf{ct},\pi) \leftarrow \mathcal{A}(\mathsf{pk})\ ;\ (\mu, r) \leftarrow \mathcal{E}(\mathsf{pk}, \mathsf{ct}, \pi)\ :$$
$$\mathsf{TFHE.Encrypt}(\mathsf{pk}, \mu; r) \neq \mathsf{ct}, \text{ and}$$
$$\mathsf{TFHE.Verify}(\mathsf{pk}, \mathsf{ct}, \pi)=1\ )\Big] \le \mathsf{negl}(\lambda).$$

Definition 9 (Simulation security [11]). *We say that a TFHE scheme satisfies simulation security if for all λ, depth bound d, and access structure $\mathbb{A}$, the following holds. There exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all PPT adversaries $\mathcal{A}$, the following experiments $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(1^\lambda, 1^d)$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda, 1^d)$ are indistinguishable:*

$\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(1^\lambda, 1^d)$:

1. *On input the security parameter 1^λ and a circuit depth 1^d, the adversary $\mathcal{A}$ outputs $\mathbb{A} \in \mathbb{S}$.*
2. *The challenger runs $(\mathsf{pk},\mathsf{sk}_1,...,\mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ and provides pk to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a maximal invalid party set $S^* \subseteq \{P_1, ..., P_N\}$, messages $\mu_1, ..., \mu_k \in \mathbb{F}_2^n$ and randomness $r_1, ..., r_k$.*
4. *The challenger provides the keys $\{\mathsf{sk}_i\}_{i\in S^*}$ and $\{\mathsf{TFHE.Encrypt}(\mathsf{pk},\mu_i;r_i)\}_{i\in[k]}$ to $\mathcal{A}$.*
5. *$\mathcal{A}$ issues a polynomial number of adaptive queries of the form $(S \subseteq \{P_1, ..., P_N\}, C)$ for circuits $C : \mathbb{F}_2^{n\times k} \to \mathbb{F}_2^n$ of depth at most d. For each query, the challenger computes $\hat{\mathsf{ct}} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk},C,\mathsf{ct}_1,...,\mathsf{ct}_k)$ and provides the set $\{\mathsf{TFHE.PartDec}(\mathsf{pk},\hat{\mathsf{ct}},\mathsf{sk}_i)\}_{i\in S}$ to $\mathcal{A}$.*
6. *At the end of the experiment, $\mathcal{A}$ outputs the set of sets it received from the challenger.*

$\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(1^\lambda, 1^d)$:

1. *On input the security parameter 1^λ and a circuit depth 1^d, the adversary $\mathcal{A}$ outputs $\mathbb{A} \in \mathbb{S}$.*
2. *The challenger runs $(\mathsf{pk},\mathsf{sk}_1,...,\mathsf{sk}_N,\mathsf{st}) \leftarrow \mathcal{S}_1(1^\lambda, 1^d, \mathbb{A})$ and provides pk to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a maximal invalid party set $S^* \subseteq \{P_1, ..., P_N\}$, messages $\mu_1, ..., \mu_k \in \mathbb{F}_2^n$ and randomness $r_1, ..., r_k$.*
4. *The challenger provides the keys $\{\mathsf{sk}_i\}_{i\in S^*}$ and $\{\mathsf{TFHE.Encrypt}(\mathsf{pk},\mu_i;r_i)\}_{i\in[k]}$ to $\mathcal{A}$.*
5. *$\mathcal{A}$ issues a polynomial number of adaptive queries of the form $(S \subseteq \{P_1, ..., P_N\}, C)$ for circuits $C : \mathbb{F}_2^{n\times k} \to \mathbb{F}_2^n$ of depth at most d. For each query, the challenger runs the simulator $\{\mathsf{p}_i\}_{i\in S} \leftarrow \mathcal{S}_2(C, \{\mathsf{ct}_1, ..., \mathsf{ct}_k\}, C(\mu_1, ..., \mu_k), S, \mathsf{st})$ and sends $\{\mathsf{p}_i\}_{i\in S}$ to $\mathcal{A}$.*
6. *At the end of the experiment, $\mathcal{A}$ outputs the set of sets it received from the challenger.*

3 Defining Single Secret Leader Election

This section defines single secret leader election (SSLE) and its security properties as well as a number of practical restrictions on performance and resilience that an SSLE scheme should satisfy.

SSLE requirements. Informally, an SSLE scheme involves N users $\mathcal{U}_1, \ldots, \mathcal{U}_N$ with access to each other's public keys, a shared public ledger, and an unbiased randomness beacon. This group of users needs to repeatedly select exactly one leader U_{i^*} such that only U_{i^*} knows who she is and other users remain oblivious to the leader's identity, until she reveals herself. That is, each participant learns whether she is the leader and nothing else. The selected leader can provide a proof that she was selected.

The scheme must satisfy a number of properties: (1) *uniqueness* means that exactly one leader is chosen in each election; (2) *fairness* means that each user has a $\frac{1}{N}$ probability of becoming the leader, and as long as there is at least one honest user, a set of malicious users cannot influence the result of an election; and (3) *unpredictability* means that an adversary who does not control the leader cannot learn which user has been elected.

Ultimately, an SSLE protocol should satisfy a robustness property requiring that a denial of service attack against an α fraction of users succeeds in disrupting the election with probability at most α. This is the best-possible security because an attack against a random α fraction of the network hits the randomly-chosen leader with probability α, and the election fails if the chosen leader is unable to perform its leadership role. Robustness is in fact implied by the combination of uniqueness, fairness, and unpredictability because so long as there is exactly one leader chosen uniformly at random from the set of participants such that no adversary can guess who the leader will be, an attacker can do no better than guess who to attack.

We would like to minimize the amount of data posted to the public ledger and the computational cost for users in each election. Since blockchain applications involve running elections on a continuing basis, we can allow for communication costs to be amortized over many elections, with participants registering once to participate in all elections until they decide to exit.

3.1 A "straw man" non-example

Before presenting formal definitions and constructions for SSLE schemes, we describe a simple scheme that *does not* satisfy the requirements for an SSLE. This construction bears a resemblance to prior solutions that rely on VRFs (e.g. [3, 29]) but uses only commitments and no global secret keys. It serves to illustrate the kind of approach used in prior work and highlight the new requirements that motivate SSLE. Since some use cases do not strictly require that exactly one leader be elected, it is possible that this scheme suffices for some use-cases.

Our straw man scheme proceeds as follows. Users can register to participate in elections by posting a commitment $\mathsf{com}(v_{\mathsf{id}})$ to the ledger, where id is a user's public identity and v_{id} is a random element in $\mathbb{F}_p$, where p is a λ-bit prime that is a public parameter shared by all participants. Each election begins when a public randomness beacon publishes a random value $R \xleftarrow{R} \mathbb{F}_p$. The winner of the election is the participant who has the minimal value of $|R - v_{\mathsf{id}}|$. Unfortunately, no user can determine alone if she is the winner, so any user who has a good chance of being the winner reveals herself as a potential winner. That is, all users for whom $|R - v_{\mathsf{id}}| < 10 \cdot \frac{2^\lambda}{N}$ open their commitments to v_{id}, so that the ultimate winner becomes apparent as the one whose choice of v_{id} results in the smallest value of $|R - v_{\mathsf{id}}|$ among those who post.

It is clear that the above protocol will elect one leader with high probability and that the leader will be chosen uniformly at random from among the list of participants. The leader's identity is totally unpredictable until the small group of candidate leaders is revealed, but once that list is revealed the leader's identity is known to everyone. This means that if the leader is to privately do some expensive task before revealing herself, all potential leaders must do this task before revealing that they *might* be leaders, resulting in a great deal of duplication of work. Another problem with this scheme happens if a participant realizes that she was selected as the leader, but chooses not to reveal herself, allowing another participant with a higher value of $|R - v_{\mathsf{id}}|$ to claim leadership. Later, the true leader who remained secret can make her claim to winning the election public and throw into doubt the result of any work done in the intervening time, e.g. any blocks published in a proof of stake blockchain system. Yet another problem happens in case the final leader's post of v_{id} does not reach all the users (nodes) in the system. In this case, users will have different views of who was elected, causing a fork. This cannot happen if the election protocol ensures that only a single user can prove that it won the election, no matter what the other users do.

In general, solutions based on selecting a small group of potential leaders by flipping a biased coin will not meet the requirement that the unique leader must learn she was elected before her identity becomes public. Ensuring that there is a canonical leader whose identity remains hidden until she chooses to reveal herself – and never sooner – is the problem that an SSLE must solve.

3.2 Formalizing SSLE definitions

We now formally define the syntax and security properties of SSLE. In order to accommodate our diverse approaches to solving this problem, the syntax includes parameters and outputs which may be left empty if not required by a given scheme. For blockchain applications, we use the state st to record data that will be stored on the blockchain by the elected leader in each election, and by changes made after a user registers for elections.

Definition 10 (Single secret leader election (SSLE)). *A single secret leader election scheme is a tuple of PPT algorithms* $\mathsf{SSLE} = (\mathsf{SSLE.Setup}, \mathsf{SSLE.Register}, \mathsf{SSLE.RegisterVerify}, \mathsf{SSLE.Elect}_1, \mathsf{SSLE.Elect}_2, \mathsf{SSLE.Verify})$ *with the following behavior:*

- $\mathsf{SSLE.Setup}(1^\lambda, \ell, N) \to \mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N, \mathsf{st}_0$*: The setup process generates public parameters pp, a number of secrets to be used later, and an initial state* st_0*. Here* N *is an upper bound on the number of participants supported by the scheme, and* ℓ *an optional lower bound on the number of required users per election.* $\mathsf{SSLE.Setup}$ *is a one-time setup process intended to be run a single time before initiating a series of elections.*
- $\mathsf{SSLE.Register}(i, \mathsf{pp}, \mathsf{st}) \to k_i, \mathsf{rt}_i, \mathsf{st}'$*: Each user registers with a unique public identity* $i \in [N]$*, the public parameters pp, and the current state st. Registration outputs a secret* k_i*, gives a user a registration token*

rt_i, and modifies the state to st'. SSLE.Register is run by each participant when that participant wants to begin taking part in elections. The participant registers once and stays registered unless she decides to leave. Some schemes will require an elected leader to re-register after having been elected.

- *SSLE.RegisterVerify(i, k_i, rt_i, pp, st) $\to 0/1$: SSLE.RegisterVerify is run by previously registered users after a new user registers to verify that the registration was carried out correctly. Verification can use the verifying user's secret k_i, registration token rt_i, the public parameters pp and current state st.*
- *SSLE.Elect$_1$(pp, st, R, i, sk_i) $\to p_i, l_i$: Leader election begins by taking public parameters, current state, a random $R \in \mathcal{R}$ (generated by a randomness beacon), and user $\mathcal{U}_i$'s secret key sk_i, and outputting intermediate values p_i and l_i.*
- *SSLE.Elect$_2$(pp, st, l_1,...,l_m, i, k_i, sk_i, rt_i, p_i) $\to 1/0, \pi/\perp$: Leader election concludes by taking outputs of Elect$_1$ as well as $\mathcal{U}_i$'s secrets k_i, sk_i, and rt_i and outputting whether user $\mathcal{U}_i$ has been chosen as the leader, and potentially a proof of leadership. For schemes that do not require an intermediate output from an election, we use the shorthand notation Elect(pp, st, R, i, k_i, sk_i, rt_i) $\to 1/0, \pi/\perp$ to combine Elect$_1$ and Elect$_2$, omitting unused inputs. The algorithms making up SSLE.Elect define the actual protocol to be executed between participants in an election each time they wish to elect a leader.*
- *SSLE.Verify(i, pp, st, R, π_i; p_i) $\to$ 1/0: Given an index i, the state st, the election randomness $R \in \mathcal{R}$, a proof π_i claiming that a particular user was elected leader, and optionally an intermediate value p_i from the election, the verification algorithm accepts or rejects the proof that user $\mathcal{U}_i$ has been elected leader. SSLE.Verify is used to check the authenticity of a participant who claims to be the leader when it is time for the leader to reveal herself.*

We could also include a Revoke algorithm for users to indicate that they no longer wish to participate. We refrain from formalizing this algorithm as it does not significantly impact the security properties we wish to achieve, but our schemes can be modified to include a Revoke algorithm.

We now formalize our security definitions. All our elections account for the repeated nature of elections in real deployments of SSLE, allowing the adversary to choose which users register for each election and how many elections occur. While our definitions have all previously registered users run SSLE.RegisterVerify after each registration, our Obfuscation and TFHE-based SSLE schemes (see Sections 4 and 5) will retain all their security properties so long as any single honest user runs SSLE.RegisterVerify.

Uniqueness requires that exactly one participant in an election can prove that she is the elected leader. Our definition allows an adversary to corrupt as many users as it wants and still requires that at most one leader be elected in any given election. We do allow for zero leaders to be elected because if a corrupted participant is elected leader, it may choose not to announce that it is the leader. We also allow the adversary to produce proofs of leadership after seeing honest parties' messages to account for an attacker who will use this information to produce fake proofs.

Definition 11 (Uniqueness). *We denote the uniqueness experiment with security parameter λ using UNIQUE[$\mathcal{A}, \lambda, \ell, N$]. The experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:*

Setup Phase. *Adversary $\mathcal{A}$ picks a number $c < N$ as well as a set of indexes $M \subset [N]$, $|M| = c$ of users to corrupt. The challenger $\mathcal{C}$ runs pp, sk_1,...,sk_N, $st_0 \leftarrow$ SSLE.Setup($1^\lambda, \ell, N$) and gives $\mathcal{A}$ the parameters pp, state st_0, and secrets sk_i for $i \in M$.*

Elections Phase. *Adversary $\mathcal{A}$ can choose any set of users to register for elections and for any number of elections to occur, where $\mathcal{A}$ plays the role of users $\mathcal{U}_i$ for $i \in M$ and $\mathcal{C}$ plays the role of the rest of the users. The challenger $\mathcal{C}$ also generates the election randomness $R \in \mathcal{R}$.*
To register an uncorrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$, and $\mathcal{C}$ runs $k_i, \mathsf{rt}_i, \mathsf{st}' \leftarrow \mathsf{SSLE.Register}(i, \mathsf{pp}, \mathsf{st})$.
To register a corrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$ along with an updated state st'. In either case, $\mathcal{C}$ then runs $\mathsf{SSLE.RegisterVerify}(j, k_j, \mathsf{rt}_j, \mathsf{pp}, \mathsf{st})$ for any previously regisered user $\mathcal{U}_j$ where $j \in [N] \setminus M$. If any call to SSLE.RegisterVerify returns 0, the game immediately ends with output 0. Otherwise the state is updated to st'.
Each election begins with $\mathcal{C}$ generating $p_i, l_i \leftarrow \mathsf{SSLE.Elect}_1(\mathsf{pp}, \mathsf{st}, R, i, \mathsf{sk}_i)$ on behalf of each uncorrupted registered user and $\mathcal{A}$ sending values l_i for any subset of corrupted registered users. Let $l_1, ..., l_t$ be the

set of intermediate values l_i generated in this step. Then, for all uncorrupted users, $\mathcal{C}$ sets $(b_j, \pi_j) \leftarrow \mathsf{SSLE.Elect}_2(\mathsf{pp}, l_1, ..., l_t, j, k_j, \mathsf{sk}_j, \mathsf{rk}_j)$ if user j has registered for that election or $(0, \perp)$ otherwise. $\mathcal{C}$ sends (b_j, π_j) for each uncorrupted user to $\mathcal{A}$.

Output Phase. *For each election in the elections phase, $\mathcal{A}$ outputs values (b_i, π_i) for each $i \in M$.*

The experiment outputs 0 if for each election with randomness $R \in \mathcal{R}$ and state st, there is at most one user $\mathcal{U}_i^*$ (either corrupted or uncorrupted) who outputs $b_{i^*} = 1$ and π_{i^*} such that $\mathsf{Verify}(i^*, \mathsf{pp}, \mathsf{st}, R, \pi_{i^*})) = 1$. Otherwise the experiment outputs 1.

We say an SSLE scheme is unique *if no PPT adversary $\mathcal{A}$ can win the uniqueness game except with negligible probability. That is, for all PPT $\mathcal{A}$ and for any $\ell < N$, the quantity*

$$\Pr\Big[\mathit{UNIQUE}[\mathcal{A}, \lambda, \ell, N] = 1\Big] \leq \mathsf{negl}(\lambda).$$

If uniqueness only holds so long as there are at least t uncorrupted *users participating in each election, we say that $\mathcal{S}$ is t-threshold unique. We say user $\mathcal{U}_{i^*}$* wins *an election if it outputs a tuple $(1, \pi_{i^*})$ such that $\mathsf{Verify}(i^*, \mathsf{pp}, \mathsf{st}, R, \pi_{i^*}) = 1$.*

We define unpredictability with a security game where an adversary can control any number of participants in an election and, after participating in several elections, must guess which honest user won a challenge election. Our game captures the intuition that if an adversary does not control the winner, it can do no better than guess which of the honest users won the election.

Definition 12 (Unpredictability). *We denote the unpredictability experiment with security parameter λ by $\mathit{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$. The experiment is played between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$ as follows:*

Setup Phase. *Adversary $\mathcal{A}$ picks a set of indexes $M \subset [N]$, $|M| = c$ of users to corrupt. The challenger $\mathcal{C}$ runs pp, sk_1,...,sk_N, $\mathsf{st}_0 \leftarrow \mathsf{SSLE.Setup}(1^\lambda, \ell, N)$ and gives $\mathcal{A}$ the parameters pp, state st_0, and secrets sk_i for $i \in M$.*

Elections Phase. *Adversary $\mathcal{A}$ can choose any set of users to register for elections and for any number of elections to occur, where $\mathcal{A}$ plays the role of users $\mathcal{U}_i$ for $i \in M$ and $\mathcal{C}$ plays the role of the rest of the users. The challenger $\mathcal{C}$ also generates the election randomness $R \in \mathcal{R}$.*

To register an uncorrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$, and $\mathcal{C}$ runs $k_i, \mathsf{rt}_i, \mathsf{st}' \leftarrow \mathsf{SSLE.Register}(i, \mathsf{pp}, \mathsf{st})$. To register a corrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$ along with an updated state st'. In either case, $\mathcal{C}$ then runs $\mathsf{SSLE.RegisterVerify}(j, k_j, \mathsf{rt}_j, \mathsf{pp}, \mathsf{st})$ for any previously regisered user $\mathcal{U}_j$ where $j \in [N] \setminus M$. If any call to $\mathsf{SSLE.RegisterVerify}$ returns 0, the game immediately ends with output 0. Otherwise the state is updated to st'.

Each election begins with $\mathcal{C}$ generating $p_i, l_i \leftarrow \mathsf{SSLE.Elect}_1(\mathsf{pp}, \mathsf{st}, R, i, \mathsf{sk}_i)$ on behalf of each uncorrupted registered user and $\mathcal{A}$ sending values l_i for any subset of corrupted registered users. Let $l_1, ..., l_t$ be the set of intermediate values l_i generated in this step. Then, for all uncorrupted users, $\mathcal{C}$ sets $(b_j, \pi_j) \leftarrow \mathsf{SSLE.Elect}_2(\mathsf{pp}, l_1, ..., l_t, j, k_j, \mathsf{sk}_j, \mathsf{rk}_j)$ if user j has registered for that election or $(0, \perp)$ otherwise. Finally, $\mathcal{C}$ sends (b_j, π_j) for each uncorrupted user to $\mathcal{A}$.

Challenge Phase. *At some point after all users $\mathcal{U}_j$ for $j \in [n]$ have registered, $\mathcal{A}$ indicates that it wishes to receive a challenge, and one more election occurs. In this election, $\mathcal{C}$* does not *send (b_j, π_j) for each uncorrupted user to $\mathcal{A}$. Let $\mathcal{U}_i$ be the winner of this election. The game ends with $\mathcal{A}$ outputting an index $i' \in [N]$. If, for $\mathcal{U}_i$ elected in the challenge phase, $i \in M$, then the output of $\mathit{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$ is set to 0. Otherwise, $\mathit{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$ outputs 1 iff $i = i'$.*

We say that an SSLE scheme $\mathcal{S}$ is unpredictable *if no PPT adversary $\mathcal{A}$ can win the unpredictability game with greater than negligible advantage when the winner of the election is uncorrupted. That is, if for all PPT $\mathcal{A}$, for any $c \leq n - 2$, $n \leq N$, and for any $\ell < N$ the quantity*

$$\Pr\Big[\mathit{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c] = 1 \mid i \in [N] \setminus M\Big] \leq \frac{1}{n - c} + \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ wins with advantage $\alpha + \mathsf{negl}(\lambda)$ for $\alpha > \frac{1}{n-c}$, with α potentially depending on c, n, or N, we say that $\mathcal{S}$ is α-unpredictable. If the value of α depends on N, then we require that $n = N$. If unpredictability

only holds for $c < t$ for some $t > 0$, we say that $\mathcal{S}$ is t-threshold unpredictable. We can also define a selectively *secure version of the unpredictability game where the adversary registers all the users who wish to participate in the challenge phase during the setup and sends the list to the challenger before the challenger runs* SSLE.Setup.

Note that a trivial election scheme that never elects a leader does satisfy this notion of unpredictability because it will never be the case that $i \in [N] \setminus M$. However, this is fine because such a trivial scheme does not satisfy fairness (defined below), and a viable SSLE scheme must sastisfy all our definitions.

It might seem that any unpredictable scheme must also be fair or else the unpredictability adversary could gain a non-negligible advantage by guessing the index of a user more likely to win an election. However, observe that the definition of unpredictability only considers the case where the adversary does not control the winner of the challenge election. If the adversary can manipulate an SSLE protocol so that it always controls the winner, our unpredictability definition becomes vacuous.

We require fairness to protect against such an adversary. The key idea behind our definition of fairness is that a scheme is fair if the best the adversary can do to be elected is to actually win the election honestly, i.e. with probability equal to the fraction of adversary-controlled participants. Moreover, fairness also requires that an honest user wins the election with probability equal to the fraction of honest users.

Definition 13 (Fairness). *We denote the fairness experiment with security parameter λ using $\mathsf{FAIR}[\mathcal{A}, \lambda, \ell, N, c, n]$. The experiment is played between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$ as follows:*

Setup Phase. *Adversary $\mathcal{A}$ picks a set of indexes $M \subset [N]$, $|M| = c$, of users to corrupt. The challenger $\mathcal{C}$ runs $\mathsf{pp}, \mathsf{sk}_1, ..., \mathsf{sk}_N, \mathsf{st}_0 \leftarrow \mathsf{SSLE.Setup}(1^\lambda, \ell, N)$ and gives $\mathcal{A}$ the parameters pp, state st_0, and secrets sk_i for $i \in M$.*

Elections Phase. *Adversary $\mathcal{A}$ can choose any set of users to register for elections and for any number of elections to occur, where $\mathcal{A}$ plays the role of users $\mathcal{U}_i$ for $i \in M$ and $\mathcal{C}$ plays the role of the rest of the users. The challenger $\mathcal{C}$ also generates the election randomness $R \in \mathcal{R}$.*
To register an uncorrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$, and $\mathcal{C}$ runs $k_i, \mathsf{rt}_i, \mathsf{st}' \leftarrow \mathsf{SSLE.Register}(i, \mathsf{pp}, \mathsf{st})$.
To register a corrupted user, $\mathcal{A}$ sends the index i of the user to $\mathcal{C}$ along with an updated state st'. In either case, $\mathcal{C}$ then runs $\mathsf{SSLE.RegisterVerify}(j, k_j, \mathsf{rt}_j, \mathsf{pp}, \mathsf{st})$ for any previously regisered user $\mathcal{U}_j$ where $j \in [N] \setminus M$. If any call to $\mathsf{SSLE.RegisterVerify}$ returns 0, the game immediately ends with output 0. Otherwise the state is updated to st'.
Each election begins with $\mathcal{C}$ generating $p_i, l_i \leftarrow \mathsf{SSLE.Elect}_1(\mathsf{pp}, \mathsf{st}, R, i, \mathsf{sk}_i)$ on behalf of each uncorrupted registered user and $\mathcal{A}$ sending values l_i for any subset of corrupted registered users. Let $l_1, ..., l_t$ be the set of intermediate values l_i generated in this step. Then, for all uncorrupted users, $\mathcal{C}$ sets $(b_j, \pi_j) \leftarrow \mathsf{SSLE.Elect}_2(\mathsf{pp}, l_1, ..., l_t, j, k_j, \mathsf{sk}_j, \mathsf{rk}_j)$ if user j has registered for that election or $(0, \perp)$ otherwise. $\mathcal{C}$ sends (b_j, π_j) for each uncorrupted user to $\mathcal{A}$.

Challenge Phase. *At some point after all users $\mathcal{U}_i$ for $i \in [n]$ have registered, $\mathcal{A}$ indicates it wishes to receive a challenge, and one more election occurs. $\mathsf{FAIR}[\mathcal{A}, \lambda, \ell, N, c, n]$ outputs 1 if there is no $i \in [n] \setminus M$ for which $\mathsf{Verify}(i, \mathsf{pp}, \mathsf{st}, \pi_i) = 1$ in the challenge election.*

We say that an SSLE scheme $\mathcal{S}$ is fair *if no PPT adversary $\mathcal{A}$ can win the fairness game with greater than negligible advantage. That is, if for all PPT $\mathcal{A}$, $n \leq N$, $c < n$, and for any $\ell < N$,*

$$\left| \Pr\left[\mathsf{FAIR}[\mathcal{A}, \lambda, \ell, N, c, n] = 1 \right] - c/n \right| \leq \mathsf{negl}(\lambda).$$

If fairness only holds for $c < t$ for some $t > 0$, we say $\mathcal{S}$ is t-threshold fair. We can define a selectively *secure version of the fairness game where the adversary registers all users who wish to participate in the challenge phase during setup and sends the list to the challenger before it runs* SSLE.Setup.

Requirements and variations for real-world use cases. Our goal in constructing SSLE schemes is to build protocols that satisfy the above security definitions while achieving performance characteristics acceptable for use in applications of SSLE. Restrictions imposed by applications include limitations on ledger

growth as a result of each election, limitations on the computation required of each party, and scalability requirements to large numbers of users.

Some use cases of SSLE [38] require that users' chances of winning an election be weighted, e.g. according to their stake in a proof of stake system as recorded in a public *power table*. Each of our constructions will be accompanied by a discussion of how to adapt the scheme to handle this requirement.

Another common use case requires picking multiple leaders in an ordered list, such that each leader learns her own position in the list (and a proof for that fact), but nothing about the other leaders. Any SSLE scheme can trivially achieve a similar property by repeating the protocol several times, once for each leader, before having any leader reveal herself. Note that it might happen that the same leader appears several times on the ordered list, which may or may not be acceptable. However, it may also be possible for a scheme to natively support such a functionality without incurring the cost of running several elections in parallel. Our obfuscation and DDH-based solutions (in Sections 4 and 6, respectively) will natively support efficient selection of multiple leaders.

4 SSLE from Obfuscation

The first question to answer after stating the requirements of a SSLE is whether a protocol with all the requisite security properties can be achieved. In this section, we answer the question affirmitively by presenting an SSLE protocol built from indistinguishability obfuscation ($i\mathcal{O}$) [5,27] that satisfies all the requirements set forth in Section 3, albeit with selective unpredictability and fairness. Note that the goal of this construction is not to present a candidate that can be realized in practice, but to showcase the behavior we would expect of an ideal SSLE scheme as a first step toward practical constructions. Subsequent sections will describe practical protocols targeted at real-world use cases.

In this solution, a one-time distributed setup protocol will choose a puncturable PRF [15, 17, 36] key k and embed it in an obfuscated program. The program, given a list of public keys, an index, and some public randomness, uses the PRF to choose one key from the list of public keys to be the winner and outputs a commitment to 0 or 1. If the index matches the winning public key, it outputs a commitment to 1. Otherwise, it outputs a commitment to 0. Moreover, the randomness used in the commitment is encrypted to the public key of the input index as a second output.

In order to register to participate in this scheme, a user just needs to generate a key pair and publish the public key. In each round, users run the obfuscated program with the list of participating public keys, their own indexes, and randomness from a public randomness beacon. After decrypting their respective commitment randomnesses, the leader will find a commitment to 1 whereas all other users will receive a zero. To prove leadership, the leader publishes the randomness used to commit to the output it received.

More precisely, the scheme would begin with a trusted setup phase in which, first, a random key k is chosen from $\mathcal{K}$, the distribution of keys for a puncturable PRF. Let F be a puncturable PRF, ($\mathsf{COM.com}, \mathsf{COM.verify}$) a commitment scheme, and ($\mathsf{PKE.Encrypt}$,$\mathsf{PKE.Decrypt}$) a CPA-secure public-key encryption scheme. The output of the setup phase is an obfuscated circuit $\widetilde{P} \leftarrow \mathcal{O}(P)$ that is posted to the ledger, where $\mathcal{O}$ is an indistinguishability obfuscator and P implements the following function.

$P((\mathsf{pk}_0,...,\mathsf{pk}_{n-1}), i, n, R)$:

1. $s \leftarrow R, \mathsf{pk}_0, ..., \mathsf{pk}_{n-1}$
2. $(w, r, r') \leftarrow F(k, s)$
3. $b \leftarrow 1$ if $i = w \bmod n$, $b \leftarrow 0$ otherwise
4. $\mathsf{c} \leftarrow \mathsf{COM.com}(b; r)$
5. $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_i, r; r')$
6. Output c, ct.

For each election with n participants, user $\mathcal{U}_i$ sets $(\mathsf{c}, \mathsf{ct}) \leftarrow \widetilde{P}((\mathsf{pk}_1, ..., \mathsf{pk}_n), i, n, R)$ and runs $\mathsf{COM.verify}(\mathsf{c}, 1, \mathsf{PKE.Dec}(\mathsf{sk}_i, \mathsf{ct}))$ with the output R of the randomness beacon to recover a bit 1 or 0. By construction, all users will receive a 0 except the leader, who receives a 1. Moreover, the choice of leader is determined by the

output of the PRF on the list of participant keys and the public randomness, ensuring fairness. Uniqueness comes from the binding property of the commitment scheme. Unpredictability comes from the security of the punctured PRF, commitment scheme, encryption scheme, and obfuscator, which together ensure that users can only read their own output from $\widetilde{P}$. We formalize the protocol below.

Construction 14 (Obfuscation-based SSLE)**.** *Our Obfuscation-based SSLE scheme* OSSLE *= (OSSLE.Setup, OSSLE.Register, OSSLE.RegisterVerify, OSSLE.Elect, SSSLE.Verify) with security parameter* λ *uses a puncturable PRF* F*, a commitment scheme* $\mathsf{COM} = (\mathsf{COM.com}, \mathsf{COM.verify})$*, a public-key encryption scheme* PKE*=(PKE.Setup, PKE.Encrypt, PKE.Decrypt), and an indistinguishability obfuscator* $\mathcal{O}$*.*

- *OSSLE.Setup(*$1^\lambda, \ell, N$*): Choose* $k \xleftarrow{R} \{0,1\}^\lambda$ *and use it to create the leader picking circuit* P *described above. Then Let* $\widetilde{P} \leftarrow \mathcal{O}(P)$*. Output* pp*=(*$\lambda, \widetilde{P}$*) and* st*={}. Inputs* ℓ *and* N *are unused.*
- *OSSLE.Register(i, pp, st): Recover* λ *from* pp*. Let (*pk_i*,*sk_i*)* $\leftarrow$ *PKE.Setup(*1^λ*). Append* pk_i *to* st *and output (*pk_i*,*sk_i*) as* rt_i*. Output* k_i *is left empty.*
- *OSSLE.RegisterVerify(i,* k_i*,* rt_i*, pp, st): Output 1 iff* $\mathsf{pk}_i \in \mathsf{st}$ *and there are no duplicate public keys in* st*. Input* k_i *is unused.*
- *OSSLE.Elect(pp, st,* R*,* i*,* rt_i*): Interpret* st *as* pk_1*,...,*pk_n*,* rt_i *as* sk_i*, and recover* $\widetilde{P}$ *from* pp*. Then run* c*,* $\mathsf{ct} \leftarrow \widetilde{P}(\mathsf{pk}_1,...,\mathsf{pk}_n, i, n, R)$*. If* $\mathsf{COM.verify}(\mathsf{c}, 1, \mathsf{PKE.Dec}(\mathsf{sk}_i, \mathsf{ct})) = 0$*, output* 0*. Otherwise, output* 1 *and* $\pi \leftarrow (i,$ *PKE.Decrypt(*sk_i*,* ct*)).*
- *OSSLE.Verify(i, pp, st,* R*,* π_i*): Interpret* st *as* pk_1*,...,*pk_n*,* π_i *as* (i, r)*, and recover* $\widetilde{P}$ *from* pp*. Then run* c*,* $\mathsf{ct} \leftarrow \widetilde{P}(\mathsf{pk}_1,...,\mathsf{pk}_n, i, n, R)$*. Output* $\mathsf{COM.verify}(\mathsf{c}, 1, r)$*.*

Extensions. The obfuscation-based approach outlined above can easily accommodate a power table that determines each user's probability of election by taking an additional input T representing the power table and giving each user $\mathcal{U}_i$ a range of values of $w \bmod n$ for which they would be elected whose size corresponds to their stake. The scheme could also be extended to output multiple encryptions instead of only one in order to elect more than one leader in each election.

We prove the following security theorem in Appendix B.

Theorem 15. *Assuming that* F *is a puncturable PRF, that* COM *is a correct, binding, and hiding commitment scheme, that* PKE *is a correct and CPA-secure public-key encryption scheme, and that* $\mathcal{O}$ *is an indistinguishability obfuscator, then* OSSLE *is a unique, selectively unpredictable, and selectively fair SSLE scheme.*

5 SSLE from TFHE

This section shows how to build an SSLE scheme based on threshold fully homomorphic encryption (TFHE) [11] for a shallow circuit. This scheme will require t users to post partial decryptions of a ciphertext in each election, for a threshold t chosen as a parameter to the scheme. However, we maintain resistance against disruption by using threshold encryption so that as long as *any* t of the participants remain online, the election will succeed. One caveat of this scheme compared to the previous scheme is a more expensive user registration process.

Setup requires a group of $\ell = t$ users to set up a TFHE scheme and generate a TFHE encryption of a PRF key k. When a user joins, she needs approval from t existing participants who can generate a new threshold decryption key for that user. Additionally, users register by uploading a TFHE ciphertext containing a random secret $k_i \in \{0,1\}^\lambda$, which is appended to a vector of secrets. To elect a leader, users (loosely speaking) generate randomness inside the TFHE and then use it to randomly select a value of k_i from the vector of secrets. The user whose secret is chosen knows she has been elected, but nobody else knows that the revealed secret is hers. To participate in future elections, she re-registers with a new secret k_i'. To realize this scheme, we need to show, first, how to get secret randomness in the TFHE and, second, how to use it to select a leader.

Generating randomness inside the TFHE. One way to easily generate randomness inside the TFHE is to have many users upload encryptions of random bit vectors and then to xor together users' contributions and use the result. This approach requires no FHE multiplications, but requires a great deal of communication. We reduce communication by taking advantage of a randomness beacon which outputs public randomness $R \in \mathbb{F}_2^{\log N}$ each election. We would like to interpret the public randomness R as an FHE encryption of a secret randomness R'. Unfortunately, we know of no dense FHE scheme where any R can be directly interpreted as a ciphertext. We get around this by treating R as an input to a PRF keyed with k (which we have encrypted under the TFHE). The computation to generate randomness for each election consists of running R' through a block cipher under the TFHE to get a TFHE ciphertext of $\mathsf{PRF}(k, R')$. In practice, we could use a low-depth block cipher designed for use in FHE schemes [2, 19, 24, 40] to keep multiplicative depth as low as 5. Moreover, since the output of most block ciphers produces more bits than we will need in each election, the block cipher could be evaluated once every several elections, and elections in between could simply use subsequent chunks of the block cipher output, only xoring them with new randomness beacon outputs instead of running a new block cipher evaluation. We discuss choice of PRF as well as other optimizations and practical considerations in more detail after formalizing our construction.

Selecting a leader. Once we have generated a TFHE ciphertext containing $\log N$ random bits, we can use them to select a leader. First, we will expand the random bits to a vector of length N with only one randomly chosen entry set to 1 and all other entries set to 0. We begin by expanding each random bit b into a vector $(b, 1-b)$, so that each vector has one 0 and one 1. Then we pair off the vectors produced, take the outer products between them, and reinterpret the output matrices as longer vectors, resulting in vectors of length 4 which will still have 1 in exactly one index and zero elsewhere. We repeat this process until we are left with a single vector v of length N, which will be set to 1 at exactly one index and zero everywhere else. This requires only $\log N$ multiplications and has multiplicative depth $\log \log N$, making it extremely efficient (e.g. depth 4 for $N = 2^{16}$ participants). Having computed v under the TFHE, we take the inner product between v and s to get the value s_i which determines the leader. This step has a multiplicative depth of 1, bringing the total depth of the entire leader election circuit to as little as 10 for $N = 2^{16}$ participants.

Defending against duplication and modification attacks. The scheme as described thus far remins vulnerable to two attacks that we call duplication and modification attacks.

A *duplication attack* compromises uniqueness. Two malicious users who choose the *same secret* k_i can both legitimately claim to be the winner of the election if k_i is chosen as the winning key. To avoid such an attack, we must ensure that no two users can share the same k_i. We achieve this by splitting each user's key into private and public components k_{iL} and k_{iR}. The private component plays the same role that k_i has played thus far, and the public component is posted publicly to ensure that duplicate keys are detected at registration time. Using a random oracle, the public and private components of each user's key can be generated from a single master key such that it is hard to find a master key that results in collisions in the private components of the output. In practice, we only require it to be hard to find collisions in the private component of the hash function's output, so we can instantiate such a random oracle for $\lambda = 128$ using one call to the SHA384 hash function, setting the first 256 bits as the private output and the last 128 bits as the public output.

A *modification attack* targets unpredictability. A malicious user $\mathcal{U}_j$ registers by uploading a value of s_j that corresponds to the plaintext of s_i plus one for some honest user $\mathcal{U}_i$ (and uploading a random value for k_{jR}). This ciphertext s_j can easily be obtained because the encryption is homomorphic. Then if $\mathcal{U}_j$ "wins" an election, the value $k_i + 1$ will be revealed. $\mathcal{U}_j$ cannot prove that he has won this election, but note that in the definition of unpredictability, malicious users are not required to prove they have won an election. Later, if $\mathcal{U}_i$ wins an election, the malicious user can recognize that that the decrypted value k_i matches the one copied from $\mathcal{U}_i$ and predict that $\mathcal{U}_i$ is the winner before she reveals herself. We defend against this attack by having each user $\mathcal{U}_i$ upload a proof of knowledge π_s of the plaintext corresponding to s_i at registration time, thus ruling out attackers that register by modifying another user's secrets.

5.1 Construction

We now formalize the construction of our TFHE-Based SSLE. After describing the construction itself, we discuss some practical considerations related to the instantiation of the protocol and its applicability ot real use cases. Our construction makes use of a subroutine $v \leftarrow \mathsf{Expand}(r)$ that takes a random vector $r \in \mathbb{F}_2^{\log N}$ and returns the r^{th} standard basis vector $v \in \mathbb{F}_2^N$, that is zero in all positions except a 1 in the r^{th} position. We show how to instantiate Expand with multiplicative depth $\log \log N$ after presenting the main construction.

Construction 16 (TFHE-based SSLE)**.** *Our TFHE-based SSLE scheme* $\mathsf{TSSLE} = (\mathsf{TSSLE.Setup}, \mathsf{TSSLE.Register}, \mathsf{TSSLE.RegisterVerify}, \mathsf{TSSLE.Elect}_1, \mathsf{TSSLE.Elect}_2, \mathsf{TSSLE.Verify})$ *uses a weak PRF* $f : \mathbb{F}_2^\lambda \times \mathbb{F}_2^n \to \mathbb{F}_2^{\log N}$ *of multiplicative depth* d*, a threshold FHE scheme* $\mathsf{TFHE} = (\mathsf{TFHE.Setup}, \mathsf{TFHE.Encrypt}, \mathsf{TFHE.Eval}, \mathsf{TFHE.PartDec}, \mathsf{TFHE.FinDEec})$*, and a random oracle* H*.*

- $\mathsf{TSSLE.Setup}(1^\lambda, \ell, N)$*: The setup algorithm prepares an access structure* $\mathbb{A}_t$ *for* $t\ (= \ell)$ *out of* N *secret sharing. It then executes the setup algorithm* $(\mathsf{pk}, \mathsf{sk}_1, ..., \mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^{d+\lceil \log \log(N) \rceil + 1}, \mathbb{A}_t)$*. Choose random* $r_1, ..., r_N \in \mathbb{F}_2^\lambda$ *and let* $c_i \leftarrow \mathsf{TFHE.Encrypt}(\mathsf{pk}, r_i)$ *for* $i \in [N]$*. Compute the encrypted PRF key* $\mathsf{rk} \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C_s, c_1, ..., c_N)$ *for circuit* $C_s : \mathbb{F}_2^{\lambda \times N} \to \mathbb{F}_2^\lambda$ *which computes the function* $C_s(r_1, ..., r_N) = \Sigma_{i=1}^N r_i = r$*. Output* $\mathsf{pp}=(\mathsf{pk}, \mathsf{rk})$*,* $\mathsf{sk}_1, ..., \mathsf{sk}_N$*. Note that because* rk *is computed using ciphertexts* $c_1, ..., c_N$*, the generation of the PRF key can be easily distributed.*
- $\mathsf{TSSLE.Register}(i, \mathsf{pp}, \mathsf{st})$*: Interpret* pp *as* $(\mathsf{pk}, \mathsf{rk})$ *and sample* $k_i \xleftarrow{R} \mathbb{F}_2^\lambda$*. Compute* $k_{iL}, k_{iR} \leftarrow H(k_i)$*, set* $s_i, \pi_{si} = \mathsf{TFHE.Encrypt}(\mathsf{pk}, k_{iL})$*, and append* (s_i, k_{iR}, π_{si}) *to* st*. Output* k_i *and* rt_i*, the randomness used to encrypt* k_{iL}*.*
- $\mathsf{TSSLE.RegisterVerify}(i, k_i, \mathsf{rt}_i, \mathsf{pp}, \mathsf{st})$*: Interpret* st *as a list of values* $(s_1, k_{1R}, \pi_{s1}), ..., (s_n, k_{nR}, \pi_{sn})$*. Output 1 if* $\mathsf{TFHE.Verify}(\mathsf{pk}, s_n, \pi_{sn}) = 1$ *and if there are no duplicate values among* $s_1, ..., s_n$ *and among* $k_{1R}, ..., k_{nR}$*.*
- $\mathsf{TSSLE.Elect}_1(\mathsf{pp}, \mathsf{st}, R, i, \mathsf{sk}_i)$*: Begin by interpreting* pp *as* $(\mathsf{pk}, \mathsf{rk})$*,* st *as* $(s_1, k_{1R}, \pi_{s1}), ..., (s_n, k_{nR}, \pi_{sn})$*, and* R *as a vector in* $\mathbb{F}_2^{\log N}$*. Compute* $p_i \leftarrow \mathsf{TFHE.Eval}(\mathsf{pk}, C_e, \mathsf{rk}, s_1, ..., s_n)$ *for circuit* C_e *(described below) with the value of* R *hard-coded inside of it. Output* p_i *and* $l_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{pk}, p_i, \mathsf{sk}_i)$*.*

  $C_e(r, k_{1L}, ..., k_{nL})$*:*

  1. $u \leftarrow f(r, R)$*. Drop all but the first* $\log n$ *entries of* u*.*
  2. $v \leftarrow \mathsf{Expand}(u)$
  3. $p_i = \Sigma_{j=1}^N (v_j \cdot k_{jL})$
  4. *Output* p_i*.*
- $\mathsf{TSSLE.Elect}_2(\mathsf{pp}, \mathsf{st}, l_1, ..., l_m, i, k_{iL}, \mathsf{sk}_i, \mathsf{rt}_i, p_i)$*: Begin by interpreting* pp *as* $(\mathsf{pk}, \mathsf{rk})$ *and* st *as* $(s_1, k_{1R}, \pi_{s1}), ..., (s_n, k_{nR}, \pi_{sn})$*. Produce a new list* $l'_1, ..., l'_t$ *of elements from* $l_1, ..., l_m$ *such that* $\mathsf{TFHE.VerifyDec}(\mathsf{pk}, l_i, p_i) = 1$*. Compute the plaintext* $l \leftarrow \mathsf{TFHE.FinDec}(\mathsf{pk}, (l'_1, ..., l'_t))$*. If* $l \neq k_{iL}$ *output* $(0, \perp)$*. Otherwise, remove* (s_i, k_{iR}) *from* st *and output* $1, \pi = (k_i, \mathsf{rt}_i, l'_1, ..., l'_t)$*.*
- $\mathsf{TSSLE.Verify}(i, \mathsf{pp}, \mathsf{st}, R, \pi; p_i)$*: Begin by interpreting* pp *as* $(\mathsf{pk}, \mathsf{rk})$*,* st *as* $(s_1, k_{1R}, \pi_{s1}), ..., (s_n, k_{nR}, \pi_{sn})$*, and* π *as* $(k_i, \mathsf{rt}_i, l_1, ..., l_t)$*. Next, check* $\mathsf{TFHE.VerifyDec}(\mathsf{pk}, l_i, p_i) = 1$ *for* $i \in [t]$ *(output 0 if any check fails), compute the plaintext* $l' \leftarrow \mathsf{TFHE.FinDec}(\mathsf{pk}, (l_1, ..., l_t))$*, and set* $k'_{iL}, k'_{iR} \leftarrow H(k_i)$*. If* $s_i = \mathsf{TFHE.Encrypt}(\mathsf{pk}, k'_{iL}; \mathsf{rt}_i)$*,* $l' = k'_{iL}$*, and* $k'_{iR} = k_{iR}$*, output 1. Otherwise, output 0.*

We implement the function $\mathsf{Expand}(r)$ as follows. For $u \in \mathbb{F}_2^m$ and $v \in \mathbb{F}_2^n$, we use $w = u \otimes v \in \mathbb{F}_2^{m \times n}$ to represent the outer product between two vectors u and v, defined as $w_{ij} = u_i v_j$.

$\mathsf{Expand}(u)$:

1. Let $N = 2^{|u|}$.
2. Let vector $v_i^0 \leftarrow (u_i, 1 - u_i)$ for each value $u_i \in \mathbb{F}_2$ in u. Let $N' \leftarrow N/2$
3. For i from 1 to $\log \log N$:
   - $v_j^{(i)} \leftarrow v_j^{(i-1)} \otimes v_{j+2^{N'/2}}^{(i-1)}$ for $j \in [N'/2]$

- Reinterpret $v_j^{(i)}$ as a vector in $\mathbb{F}_2^{2^{2^i}}$ formed by concatenating the rows of the matrix.
- $N' \leftarrow N/2^{2^i}$

4. Output $v_0^{(\log \log N)}$.

5.2 Practical Considerations

Adding users after setup. Our scheme, as written, requires the list of all users of the system to be known at the time TSSLE.Setup runs, but it can easily be extended to allow growth in the number of users after initial setup, so long as t users are available at setup time. In this case, the original t users run TSSLE.Setup with a t out of t access structure. Whenever a new user with identity i arrives, some subset of t existing users generate a new key share sk_i for user i. Using the TFHE scheme of [11], this share generation be easily accomplished via a simple protocol among the t existing users.

Maintaining security over time. Consider an attacker that waits for users to become inactive in the leader election protocol. Once they withdraw from the system, the attacker purchases their threshold key shares, thereby gradually accruing t key shares. This lets the attacker break the security of the system by decrypting ciphertexts on its own. This attack is outside of the security model of our system because we only claim security against an attacker who controls fewer parties than the threshold t, but should be considered in practice nonetheless. To defend against this risk, the active parties can periodically refresh the TFHE key using a distributed key generation (DKG) protocol. The DKG protocol generates a new FHE key with new key shares every several elections, thereby making old key shares obsolete. All users must stop using the old TFHE public key every time the TFHE key changes, lest an attacker use an old TFHE secret key to learn the secrets corresponding to users who may be elected in future elections. We note that a protocol analogous to the DKG protocol for public key encryption schemes (e.g., [28]) can also be used to refresh the TFHE key of [11].

Unequal election probabilities. This scheme can easily be extended to accommodate a power table T that allocates different likelihoods for each user being elected. Users who have higher likelihood of being elected are simply allowed to run TSSLE.Register multiple times corresponding to their allotted likelihood of winning an election. The table T is public, so all users know how many times to allow each other to register. Since in practice, differences in election probabilities are typically not extreme, the scheme's efficiency does not degrade significantly by having some users register multiple times. Moreover, extra registrations have no impact on the multiplicative depth of the C_e circuit, meaning they do not cause ciphertexts to get larger except through a limited number of additional ciphertexts added to the state st.

PRF instantiation and optimization. The construction relies on a weak PRF f whose multiplicative depth d forms a significant portion of the $d + \lceil \log \log N \rceil + 1$ depth parameter to the underlying TFHE construction. As such, it is important to choose a PRF with low multiplicative depth. Fortunately, recent years have seen the development of a number of low-depth block ciphers optimized for the MPC and FHE settings, including MiMC [1], LowMC [2], Kreyvium [19], FLIP [40], and Rasta [24]. For one parameter setting, Rasta has a multiplicative depth of 6, with the tradeoff that the block size is 351 bits. The variant of Rasta with more aggressive parameter settings, Agrasta, offers a multiplicative depth of 5 with 127 bit blocks, meaning our construction could be instantiated for $N = 2^{15}$ users with a depth of only 10 multiplications.

Note that although all the block ciphers we have considered thus far aim to act as *strong* PRFs, our scheme only requires a *weak* PRF, so it is possible that a low-depth weak PRF of significantly lower depth exists. The Dark Matter weak PRF [12] is a promising first step in this direction, but, despite its depth 2 circuit, it assumes mod operations can be carried out for free, which is not the case for the TFHE construction we use. Computing those mod operations under an FHE would cause the depth to become worse than some of the block ciphers we consider.

Since the computation of a block cipher under the TFHE constitutes the most computationally costly component of our scheme, we propose the following practical optimization to minimize the number of elections in which the PRF actually needs to be evaluated. Note that each election requires only $\log N$ bits of

randomness inside the TFHE, but the block sizes for the ciphers we use to instantiate the PRF are much larger than typical values of $\log N$. We can take advantage of all the random bits output by each evaluation of the PRF by splitting its output into chunks of size $\log N$ and using the next available chunk for each election, only evaluating the PRF again if the supply of random bits has run out. This could reduce the amortized running time of PRF evaluation in the TFHE by $8\times$ or more for realistic group sizes of 2^{15} or less. In order to ensure that it is impossible to compute the leader for a future election ahead of time, the next chunk of PRF-generated randomness could be xored with the plaintext randomness R for each election, ensuring that the exact value of the randomness for each election remains unknown until the randomness has become available.

Reducing on-chain costs. We can reduce the final storage costs for each election in a blockchain use-case by making a trade-off where we increase communication and computation for each election. In the scheme described above the proof π contains a threshold number of partial decryptions, which are stored in perpetuity so that elections can be verified after the fact. Instead of publicly posting and perpetually storing partial decryptions of the final threshold FHE ciphertext, users can communicate the decryptions to each other off-chain. When the leader reveals herself, she posts a short proof that the partial decryptions communicated during the election successfully decrypted her secret.

This corresponds to a generic transformation of any SSLE scheme with proof π that depends polynomially on the number of participants to an SSLE scheme with a shorter proof. The idea is that, given partial intermediate outputs $l_1, \ldots, l_N$ of users $U_1, \ldots, U_N$ to create π, the leader creates a succinct ZK proof (e.g. using ZK-SNARK [32]) that she knows $l_1, \ldots, l_N$, an index i, secrets k_i, sk_i, and registration token rt_i for which $\mathsf{SSLE.Elect}_2(\mathsf{pp}, l_1,\ldots,l_N, i, k_i, \mathsf{sk}_i, \mathsf{rt}_i)$ outputs 1.

5.3 Security

We now state our security theorem for the TFHE-based SSLE construction. A full proof appears in Appendix C.

Theorem 17. *Assuming that f is a weak PRF and that TFHE is a secure t out of N threshold FHE that satisfies the definitions of correctness, compactness, semantic security, robustness, simulation security, and plaintext extractability, then TSSLE is a t-threshold unique, t-threshold unpredictable, and t-threshold fair SSLE scheme in the random oracle model.*

6 SSLE from DDH and Shuffling

Our final scheme uses only the simplest of cryptographic tools and exhibits costs satisfactory for deployment in practical systems today. In return, it achieves weaker security properties than the preceding constructions. As a step toward our actual scheme, we will consider a simplification that incurs more communication and computation. Then we will show how to drastically reduce communication and computation costs while maintaining much of the security of the simplified scheme.

A high-communication scheme. In this scheme, the setup operation initializes an empty list l on a public ledger, effectively requiring users to do nothing at all. Registration will involve a user choosing a secret value $k_i \in \mathbb{Z}_q$ for some λ-bit prime q, uploading a special commitment to k_i to the list, and shuffling/re-randomizing all elements of l. Moreover, we require each user who shuffles l to post a NIZK proof that they have honestly shuffled l. To elect a leader, the output of a randomness beacon, R, is used to select a row from l. The user to whom the chosen row belongs reveals her commitment as proof of leadership and re-registers with a new secret for future rounds. A user can leave the pool of participants by revealing its row so it can be excluded from future elections.

In order for this scheme to work, we need a commitment scheme for random strings that can be re-randomized such that the new version is unlinkable to any previous version, yet the owner of the secret k_i can identify a re-randomized commitment as her own after it has been shuffled. We will now describe such a scheme. The scheme fixes a group element $g \in \mathbb{G}$ for a group $\mathbb{G}$ of prime order q. The commitment

is computed as $\mathsf{com}(k_i, r) = (g^r, g^{k_i r}) \in \mathbb{G} \times \mathbb{G}$ for $k_i, r \in \mathbb{Z}_q$. To reveal, a user outputs k_i, and the commitment (u, v) can be verified by checking that $v = u^{k_i}$. To rerandomize a commitment, anyone can compute $\mathsf{Rerand}((u, v), r') = (u^{r'}, v^{r'})$.

Reducing communication. As described, the high-communication scheme above requires each user who registers to compute a shuffle over N list items, re-randomize each, and post the new list along with a proof of honest shuffling and re-randomization. We can reduce communication costs by shuffling new entries into only a part of the list l instead of shuffling the entire list each time a new participant joins, resulting in a linear tradeoff between communication costs and security. More specifically, we assign each row l_i in l to one of $\sqrt{N}$ buckets, placing l_i in bucket j if $i = j(\mathsf{mod}\sqrt{N})$. As we shall see, this saves costs both by reducing communication for each registration and by removing the need for NIZK proofs. Unfortunately, the performance savings come at a cost in security. While still satisfying uniqueness and fairness, this scheme only provides $\frac{1}{\sqrt{N}-c}$-unpredictability, where c is the number of corrupted users. This is the case because the adversary must make its guess as to the winner of the election among all the honest users in the chosen bucket, not the total number of users in that bucket. While we use $\sqrt{N}$ buckets as an example, the same idea can easily be instantiated with a larger number of buckets and a correspondingly weaker security guarantee (and vice versa).

Improving the communication/security tradeoff. We can further improve the unpredictability of our scheme by, instead of deterministically allocating each user to a bucket, assigning new users to a bucket at random when they register. This prevents the adversary from corrupting a disproportionate number of users in one bucket, but introduces the possibility of buckets of different sizes. Using the same reasoning as above, this variation has $1/\hat{h}$-unpredictability, where $\hat{h}$ is the minimum number of honest users in a bucket. Using a Chernoff bound on the minimum number of honest users in a bucket, we find that this scheme gives $\frac{N^{1/4}}{N^{3/2}-c\sqrt{N}-\sqrt{2\lambda(N-c)}}$-unpredictability (with security parameter λ). This is a significant improvement because the adversary now needs to corrupt $O(N)$ users to guarantee breaking unpredictability instead of just $O(\sqrt{N})$.

Registration in this improved scheme requires unbiased public randomness to assign users to buckets. This randomness can be generated at the cost of introducing a one-election delay to registration: users publicly declare intent to register, and then the next randomness beacon output to be released determines bucket assignments. If the required amount of randomness gets too large to simultaneously support the election and many registrations, the beacon output can be used as a seed to a PRG instead. The one-election delay after a user declares intent to register is necessary because otherwise a user could wait until the beacon outputs favorable randomness before registering, biasing the bucket assignments.

It may be possible to instantiate our approach with a more involved shuffling procedure to get even stronger security guarantees. For example, consider the square shuffle, analyzed by Hastad [33, 34], where a table is laid out as a square grid and shuffles are applied to an interleaved pattern of the rows and columns. Using a square shuffle instead of a bucketing approach would allow for a user's row to move anywhere in the table within $2\sqrt{N}$ shuffle steps. We leave the analysis of our protocol instantiated with alternative shuffles for future work.

Removing NIZKs. We can also remove the need for NIZK proofs that shuffles were carried out honestly, saving even more space and time. Since each shuffle only operates on $\sqrt{N}$ entries, users whose entries were included in a shuffle can check every shuffled entry, requiring only $\sqrt{N}$ exponentiations, to ensure that their entry still appears after the shuffle. Proving that a shuffle was not honest simply requires posting the secret that does not appear after the shuffle, allowing others to verify that it did appear before the shuffle and did not appear after. This approach requires several users to verify every new registration but does not significantly increase demands on users who already need to check if they have won each election. It is of course also possible have users publish NIZKs to prove that they have performed an honest shuffle at registration time. One or the other approach may be more suitable depending on the application.

Defending against duplication attacks. Similar to the first sketch of the tfhe-based scheme in Section 5, the scheme described thus far remains vulnerable to a duplication attack where two malicious users register with the same secret so that they can both be elected leader at the same time. The solution in this scheme is identical to that of the Section 5.

It is also possible for a malicious user to upload a commitment to a rerandomization of another user's secret without knowing how to open the commitment, in hopes of disrupting fairness by increasing that user's chances of winning the election. We have each user check new registrations to ensure that a new registrant has not uploaded a rerandomized commitment to that user's own secret.

The modification attacks described in the previous section do not apply here because users' secrets are never revealed before a user proves she is the leader.

6.1 Construction

We formalize the variant of our scheme with deterministically allocated buckets. A very similar construction in a model where public randomness is available at registration time would yield the construction with randomly assigned buckets. We will prove the security of both constructions below because the analysis is almost identical.

Construction 18 (Shuffling-based SSLE)**.** *Our shuffling-based SSLE scheme* $\mathsf{SSSLE} = (\mathsf{SSSLE.Setup}, \mathsf{SSSLE.Register}, \mathsf{SSSLE.RegisterVerify}, \mathsf{SSSLE.Elect}, \mathsf{SSSLE.Verify})$ *for up to* N *users with security parameter* λ *uses a group* $\mathbb{G}$ *of prime order* q *where DDH is hard and a random oracle* H*.*

- $\mathsf{SSSLE.Setup}(1^\lambda, \ell, N)$*: Create an empty vector* $l = \{\}$ *and choose* $g \xleftarrow{R} \mathbb{G}$*. Output* $\mathsf{st} = (l, g, N)$*. Input* ℓ *is unused.*
- $\mathsf{SSSLE.Register}(i, \mathsf{pp}, \mathsf{st})$*: Interpret* st *as* $(l, g, N, k_{1R}, ..., k_{nR})$*. Sample* $k_i \xleftarrow{R} \{0,1\}^\lambda$ *and compute* $k_{iL}, k_{iR} \leftarrow H(k_i)$*, append* k_{iR} *to* st*, sample* $r_i \xleftarrow{R} \mathbb{Z}_q$*, and append* $(g^{r_i}, g^{r_i k_{iL}})$ *to* l*. If there is any entry in* l *whose value is* $\perp$*, put* $(g^{r_i}, g^{r_i k_{iL}})$ *in place of* $\perp$ *instead of appending it to* l*. Next, sample a random permutation* Π *on* $\lceil\sqrt{N}\rceil$ *elements and set* $b \leftarrow |l| \bmod \sqrt{N}$*. Finally, update* l *such that each* $l_{j \cdot b} = (u_{j \cdot b}, v_{j \cdot b})$ *is replaced by* $l_{j \cdot b} \leftarrow (u_{\Pi(j) \cdot b}^{r_j}, v_{\Pi(j) \cdot b}^{r_j})$ *for* $r_j \xleftarrow{R} \mathbb{Z}_q$*. Output* k_i*, the new value of* st*, and* p_i*, the index where the user's new entry has been moved by* Π*.*
- $\mathsf{SSSLE.RegisterVerify}(i, k_{iL}, \mathsf{rt}_i, \mathsf{pp}, \mathsf{st})$*: Interpret* rt_i *as* p_i *and* st *as* $(l, g, N, k_{1R}, ..., k_{nR})$*. Then run the following checks:*
  - *If* $p_i (\bmod \sqrt{N}) = |l| \bmod \sqrt{N}$ *(the newly registered user is in the same bucket as* $\mathcal{U}_i$*), check that there is exactly one entry* $l_j = (u_j, v_j)$ *in the bucket (where* j *is a multiple of* $p_i \bmod \sqrt{N}$*) such that* $u_j^{k_{iL}} = v_j$*, and update* $p_i \leftarrow j$ *for that entry.*
  - *If* $p_i \bmod \sqrt{N} \neq |l| \bmod \sqrt{N}$ *(the newly registered user is not in the same bucket as* $\mathcal{U}_i$*), check that there is no entry* $l_j = (u_j, v_j)$ *in the bucket (where* j *is a multiple of* $p_i \bmod \sqrt{N}$*) such that* $u_j^{k_{iL}} = v_j$*.*
  - *Check that there are no duplicates among* $k_{1R}, ..., k_{nR}$*.*

  If the checks above pass, output 1. Otherwise, output 0.
- $\mathsf{SSSLE.Elect}(\mathsf{pp}, \mathsf{st}, R, i, k_i, \mathsf{sk}_i, \mathsf{rt}_i)$*: Interpret* st *as* $(l, g, N, k_{1R}, ..., k_{nR})$ *and* rt_i *as* p_i*. Let* z *be the number of times* $\perp$ *appears in* l*, and let* z' *be the number of times* $\perp$ *appears before the* i^{th} *entry of* l*. If* $p_{i-z'} \neq R \bmod (N - z)$*, output 0. Otherwise, remove entry* l_{p_i} *from* l *(putting* $\perp$ *in its position) and* k_{iR} *from* st*, set* $\pi = (i, p_i, k_i)$*, and output 1. Input* sk_i *is unused.*
- $\mathsf{SSSLE.Verify}(i, \mathsf{pp}, \mathsf{st}, R, \pi)$*: Interpret* st *as* $(l, g, N, k_{1R}, ..., k_{nR})$*,* π *as* (i, p_i, k_i)*, and* l_{p_i} *as* (u, v)*. Compute* $k'_{iL}, k'_{iR} \leftarrow H(k_i)$*. If* $p_i = R \bmod N$*,* $u^{k'_{iL}} = v$*, and* $k'_{iR} = k_{iR}$*, output 1. Otherwise, output 0. If* $\mathsf{SSSLE.Verify}$ *outputs 1, user* $\mathcal{U}_i$ *is no longer registered.*

Extensions. This construction can support unequal election probabilities by allowing users with more power to register multiple times. It can also support election of multiple leaders by picking multiple values of R, one for each leader.

6.2 Security

We analyze the security of our shuffling-based SSLE construction in Appendix D. We prove the first theorem below for the scheme with deterministically allocated buckets before describing how to extend the analysis to the construction which assigns buckets randomly with access to public randomness at registration time.

Theorem 19. *Assuming that $\mathbb{G}$ is a group in which the DDH problem is hard, then for any adversary $\mathcal{A}$, SSSLE is a unique, fair, and $\frac{1}{\sqrt{N}-c}$-unpredictable SSLE scheme in the random oracle model.*

Theorem 20 (Informal). *Assuming that $\mathbb{G}$ is a group in which the DDH problem is hard, then for any adversary $\mathcal{A}$, SSSLE modified to assign buckets randomly at user registration time is a unique, fair, and $\frac{N^{1/4}}{N^{3/2}-c\sqrt{N}-\sqrt{2\lambda(N-c)}}$-unpredictable SSLE scheme in the random oracle model.*

7 Conclusion

Efficient constructions for SSLE are an important tool in the blockchain space. This paper formally defines SSLE and constructs three SSLE schemes. Our protocols based on obfuscation, FHE, and DDH offer a range of tradeoffs between security and performance, with the last construction providing levels of security and performance that may satisfy practical requirements. Although our work explores a range of tradeoffs, it remains an open problem to construct an SSLE scheme that simultaneously provides both optimal security and performance. One promising direction to explore would be to improve the security/performance tradeoff of our DDH-based construction by instantiating it with more complex shuffling schemes. We leave this as a compelling problem for future work to address.

References

1. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, 2016.
2. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. *IACR Cryptology ePrint Archive*, 2016.
3. Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette. *CoRR*, abs/1805.06786, 2018.
4. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, 2018.
5. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.
7. Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography*, 2016.
8. Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *CoRR*, abs/1605.04559, 2016.
9. Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016.
10. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, 2018.
11. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.
12. Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: - new simple PRF candidates and their applications. In *TCC*, 2018.
13. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.
14. Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography (version 0.4)*. 2017. `https://cryptobook.us`.
15. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.
16. Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015.

17. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.
18. Benedikt Buenz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. 2018.
19. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *J. Cryptology*, 31(3):885–916, 2018.
20. Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, 2017.
21. Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In *2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '10, Washington, D.C., USA, August 9-10, 2010*, 2010.
22. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT*, 2018.
23. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.
24. Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In *CRYPTO)*, 2018.
25. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.
26. Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. *IACR Cryptology ePrint Archive*, 2018.
27. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013.
28. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.
29. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, 2017.
30. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 276–288, 1984.
31. David M. Goldschlag and Stuart G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *Financial Cryptography*, 1998.
32. Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.
33. Johan Håstad. The square lattice shuffle. *Random Struct. Algorithms*, 29(4):466–474, 2006.
34. Johan Håstad. The square lattice shuffle, correction. *Random Struct. Algorithms*, 48(1):213, 2016.
35. Thomas Kerber, Markulf Kohlweiss, Aggelos Kiayias, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. *IACR Cryptology ePrint Archive*, 2018.
36. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, 2013.
37. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, 2017.
38. Protocol Labs. Secret single-leader election (ssle). https://github.com/protocol/research-RFPs/blob/master/RFPs/rfp-6-SSLE.md.
39. Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015.
40. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *EUROCRYPT*, 2016.
41. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
42. Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010.
43. QuantumMechanic. Topic: proof of stake instead of proof of work, 2011.
44. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

45. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.
46. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, 2017.

A Standard Cryptographic Primitives

Definition 21 (Pseudorandom Function [30]). *Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable, length-preserving keyed function. We say that F is a pseudorandom function (PRF) if for all probabilistic polynomial time distinguishers D,*

$$|Pr[D^{F_k}(1^n) = 1] - Pr[D^{f_n}(1^n) = 1]|$$

is negligible where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and f_n is chosen uniformly at random from the set of functions mapping n-bit strings to n-bit strings. If D is restricted to only querying its oracle on randomly chosen elements from $\{0,1\}^n$, then we call F a weak *PRF.*

Definition 22 (Public Key Encryption). *A public-key encryption scheme PKE consists of algorithms PKE=($\mathsf{PKE.Setup}$, $\mathsf{PKE.Encrypt}$, $\mathsf{PKE.Decrypt}$) over a message space $\mathcal{M}$, a randomness space $\mathcal{R}$, and a ciphertext space $\mathcal{T}$ with the following properties*

- $\mathsf{PKE.Setup}(1^\lambda) \to (\mathsf{pk},\mathsf{sk})$*: On input the security parameter λ, the setup algorithm generates a public key pk and a secret key sk.*
- $\mathsf{PKE.Encrypt}(\mathsf{pk}, m; r) \to \mathsf{ct}$*: On input a public key pk, a message $m \in \mathcal{M}$, and optional randomness $r \in \mathcal{R}$, the encryption algorithm returns a ciphertext $\mathsf{ct} \in \mathcal{T}$.*
- $\mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$*: On input a secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{T}$, the decryption algorithm outputs a message $m \in \mathcal{M} \cup \{\perp\}$.*

We say that a PKE scheme is correct if for all keys $\mathsf{pk},\mathsf{sk} \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, and for all messages $m \in \mathcal{M}$, we have that $Pr[\mathsf{PKE.Decrypt}(\mathsf{sk}, \mathsf{PKE.Encrypt}(\mathsf{pk}, m)) = m] = 1$.

We will require our public key encryption scheme to satisfy the standard notion of CPA security [14].

Definition 23 (Commitment Scheme). *A commitment scheme COM consists of algorithms COM=($\mathsf{COM.com}$,$\mathsf{COM.verify}$) over a message space $\mathcal{M}$ and randomness space $\mathcal{R}$ with the following properties*

- $\mathsf{COM.com}(m; r) \to c$*: On input a message $m \in \mathcal{M}$ and optionally a commitment randomness $r \in \mathcal{R}$, the algorithm returns a commitment c.*
- $\mathsf{COM.verify}(c, m, r) \to 1/0$*: On input a commitment c, a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, the opening algorithm outputs a bit.*

We say that a COM scheme is correct if for all all messages $m \in \mathcal{M}$ and all randomness $r \in \mathcal{R}$, we have $Pr[\mathsf{COM.verify}(\mathsf{COM.com}(m, r), m, r)] = 1$.

We will require our commitment scheme to satisfy the standard binding and hiding properties [14].

Definition 24 (DDH Assumption [14,23]). *Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, we define two experiments. Experiment b: (for $b = 0, 1$)*

- *The challenger computes*

$$\alpha, \beta, \gamma \xleftarrow{R} \mathbb{Z}_q, u \leftarrow g^\alpha, v \leftarrow g^\beta, w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^\gamma$$

  and gives the triple (u, v, w_b) to the adversary.
- *The adversary outputs a bit $\hat{b} \in \{0,1\}$.*

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s advantage in solving the Decisional Diffie-Hellman problem for $\mathbb{G}$ as

$$\mathsf{DDHadv}[\mathcal{A}, \mathbb{G}] := \Big|\mathsf{Pr}[W_0] - \mathsf{Pr}[W_1]\Big|.$$

We say that the Decisional Diffie-Hellman (DDH) assumption holds for $\mathbb{G}$ if for all efficient adversaries $\mathcal{A}$, the quantity $\mathsf{DDHadv}[\mathcal{A}, \mathbb{G}]$ is negligible.

B Proof of Theorem 15

Uniqueness. Since each user has a copy of $\widetilde{P}$ generated during an honest setup phase, it is not possible for an adversary to tamper with $\widetilde{P}$ in order to change the election result. The circuit P will always pick exactly one index i to receive a commitment to 1 and all others will receive commitments to 0. We will prove the scheme has uniqueness by showing that if an adversary could break uniqueness, it would break the binding property of the commitment scheme COM. Suppose there is an election with a winning proof (i, r_i) and that an adversary can produce another winning proof $(j, r_j) \neq (i, r_i)$. Then since, by construction, there is only one commitment to 1 output by P, one of the two winning proofs is an opening to 1 of a commitment to 0, which breaks the binding property of COM.

Unpredictability. Unpredictability will be proven through a series of hybrids. As is common with constructions based on obfuscation, we only prove selective security for unpredictability and fairness, using the fact that we know the public keys for the challenge election ahead of time to choose the point at which we puncture the PRF. We will use punctured programming [45] to replace the evaluation of the PRF at one point with a random value and then hard-code the corresponding output ciphertexts into the program, replacing the commitment to b with a commitment to 0 and replacing r with a random string.

- $\mathsf{H}_0[x]$: This hybrid corresponds to the real selective unpredictability experiment $\mathsf{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$, except the experiment outputs 0 if the uncorrupted user $\mathcal{U}_x$ does not win the challenge election.
- $\mathsf{H}_1[x]$: This hybrid changes the user which the challenger counts as the "winner" of the election. Instead of the winner being the user that can produce a proof of leadership that will be accepted by Verify, the winner is the user $\mathcal{U}_i$ for which $i = w \bmod n$. This hybrid is indistinguishable from the preceding hybrid because these two definitions of "winner" are identical in the construction.
- $\mathsf{H}_2[x]$: In this hybrid, the challenger picks the randomness R to be used in the challenge election in the setup phase before running Setup. The output of this game is identical to the preceding hybrid because R is chosen uniformly at random in both hybrids.

  For the following hybrids, let s^* be the value of $(R, \mathsf{pk}_0, ..., \mathsf{pk}_{n-1})$ to be used in the challenge election.
- $\mathsf{H}_3[x]$: In this hybrid, we modify step 2 of the circuit P where we previously set $(w, r, r') \leftarrow F(k, s)$. We replace this operation with a conditional branch where if $s = s^*$, then $(w, r, r') \leftarrow (w^*, r^*, r'^*)$ for hard-coded values $(w^*, r^*, r'^*) \leftarrow F(k, s^*)$. Otherwise $(w, r, r') \leftarrow F(k(s^*), s)$, where $k(s^*)$ is the key k punctured at s^*. This hybrid is indistinguishable from $\mathsf{H}_2[x]$ by the security of the indistinguishability obfuscator, as shown in Lemma 25.
- $\mathsf{H}_4[x]$: In this hybrid, instead of setting $(w^*, r^*, r'^*) \leftarrow F(k, s^*)$ when $s = s^*$ in the modified second step of P, we set (w^*, r^*, r'^*) to be uniformly random values. This hybrid is indistinguishable from $\mathsf{H}_3[x]$ by the security of the puncturable PRF F, as shown in Lemma 26.
- $\mathsf{H}_5[x]$: In this hybrid, we modify step 5 of the circuit P so that if $s = s^*$ it outputs a hard-coded value $\mathsf{ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i, r^*; r'^*)$ and outputs $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_i, r; r')$ otherwise. This hybrid is indistinguishable from $\mathsf{H}_4[x]$ by the security of the indistinguishability obfuscator, as shown in Lemma 27.
- $\mathsf{H}_6[x]$: In this hybrid, we further modify step 5 of the circuit P so that if $i = x$, we replace the value of ct^* with an encryption of a random string $r'' \in \mathcal{R}$. This hybrid is indistinguishable from $\mathsf{H}_5[x]$ by the CPA-security of the encryption scheme PKE, as shown in Lemma 28.

- $\mathsf{H}_7[x]$: In this hybrid, we modify step 4 of the circuit P so that if $s = s^*$ it outputs a hard-coded value $\mathsf{c}^* \leftarrow \mathsf{COM.com}(b, r^*)$ and outputs $\mathsf{c} \leftarrow \mathsf{COM.com}(b, r)$ otherwise. The circuit will include hard-coded commitments to $b = 0$ and $b = 1$ and output the appropriate one based on the value of b. This hybrid is indistinguishable from $\mathsf{H}_6[x]$ by the security of the indistinguishability obfuscator, as shown in Lemma 29.
- $\mathsf{H}_8[x]$: In this hybrid, we further modify step 4 of the circuit P so that if $i = x$, we replace the value of c^*, with a commitment to 0. This hybrid is indistinguishable from $\mathsf{H}_7[x]$ by the hiding property of the commitment scheme COM, as shown in Lemma 30.

From $\mathsf{H}_8[x]$ the output of $\widetilde{P}$ when run on the public keys of the participants and the public election randomness R consists only of a commitment to 0 and an encryption of a random string under pk_i, regardless of which uncorrupted user is the winner. Thus the view of the adversary $\mathcal{A}$ is independent of the winner of the challenge election, and it cannot do better than guessing the index of the election winner with probability $\frac{1}{n-c}$ (if the winner is uncorrupted).

Since no efficient adversary $\mathcal{A}$ can distinguish between each pair of hybrids above with more than negligible advantage, we have that

$$\left|\mathsf{Pr}[\mathsf{H}_8[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M]\right| \leq \mathsf{negl}(\lambda).$$

Next, since all our hybrids were parameterized by the condition that uncorrupted user $\mathcal{U}_x$ wins the challenge election, we take the union bound over all N users to get

$$\begin{aligned}
&\mathsf{Pr}[\mathsf{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c] = 1 \mid i \in [N] \setminus M] \\
&= \frac{1}{n-c} + \Sigma_{x=1}^{N}\left|\mathsf{Pr}[\mathsf{H}_8[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M]\right| \\
&\leq \frac{1}{n-c} + \Sigma_{x=1}^{N}\mathsf{negl}(\lambda) \\
&\leq \frac{1}{n-c} + N\mathsf{negl}(\lambda) \\
&\leq \frac{1}{n-c} + \mathsf{negl}(\lambda).
\end{aligned}$$

This completes the proof of selective unpredictability. We now state and prove the remaining Lemmas that establish the indistinguishability of hybrids $\mathsf{H}_0[x]$ through $\mathsf{H}_8[x]$. For a hybrid experiment $\mathsf{H}[x]$ and an adversary $\mathcal{A}$, we use $\mathsf{H}[x](\mathcal{A})$ to denote the random variable that represents the output of experiment $\mathsf{H}[x]$ with adversary $\mathcal{A}$.

Lemma 25. Suppose that $\mathcal{O}$ is an indistinguishability obfuscator (Definition 1). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_3[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_2[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes between $\mathsf{H}_2[x]$ and $\mathsf{H}_3[x]$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of the obfuscator $\mathcal{O}$. Algorithm $\mathcal{B}$ simulates the unpredictability challengers of $\mathsf{H}_2[x]$ and $\mathsf{H}_3[x]$ exactly except for in the invocation of $\mathcal{O}(\cdot)$, where the two challengers differ. For this step, it uses the challenger implied by the obfuscation security definition, and sends the obfuscation challenger the circuits P_2 and P_3 used in $\mathsf{H}_2[x]$ and $\mathsf{H}_3[x]$ respectively. Algorithm $\mathcal{B}$ passes on the output of $\mathcal{A}$ as its own output. If $\mathcal{B}$ receives $\mathcal{O}(P_2)$ from its challenger, it provides a perfect simulation of $\mathsf{H}_2[x]$, and if it receives $\mathcal{O}(P_3)$, it provides a perfect simulation of $\mathsf{H}_3[x]$. Thus $\mathcal{B}$ distinguishes between $\mathcal{O}(P_2)$ and $\mathcal{O}(P_3)$ with the same advantage that $\mathcal{A}$ distinguishes between $\mathsf{H}_2[x]$ and $\mathsf{H}_3[x]$. □

Lemma 26. Suppose that F is an puncturable PRF (Definition 2). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_3[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes between $\mathsf{H}_3[x]$ and $\mathsf{H}_4[x]$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of punctured PRF F. Algorithm $\mathcal{B}$ begins by sending the punctured PRF challenger the value s^* as the point at which to puncture the PRF. It receives the punctured key $k(s^*)$ and values (w^*, r^*, r'^*) in return, where (w^*, r^*, r'^*) are either the output of $F(k, s^*)$ or a uniformly random string. Algorithm $\mathcal{B}$ completes the setup of $\mathsf{H}_3[x]$ and $\mathsf{H}_4[x]$ (which are identical outside of the choice of (w^*, r^*, r'^*)) according to the description of their challengers, using the values of (w^*, r^*, r'^*) that it received from the punctured PRF challenger. For the rest of the experiment it simulates the (identical) challengers of the two hybrids exactly, and passes on the output of $\mathcal{A}$ as its own output.

Since $\mathcal{A}$ never sees the PRF key k, $\mathcal{B}$ provides a perfect simulation of $\mathsf{H}_3[x]$ when it receives values $(w^*, r^*, r'^*) \leftarrow F(k, s^*)$, and a perfect simulation of $\mathsf{H}_4[x]$ when it receives uniformly random values. Thus $\mathcal{B}$ wins the punctured PRF security game with the same advantage that $\mathcal{A}$ distinguishes between hybrids $\mathsf{H}_3[x]$ and $\mathsf{H}_4[x]$. □

Lemma 27. Suppose that $\mathcal{O}$ is an indistinguishability obfuscator (Definition 1). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_5[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes between $\mathsf{H}_5[x]$ and $\mathsf{H}_4[x]$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of the obfuscator $\mathcal{O}$. $\mathcal{B}$ simulates the unpredictability challengers of $\mathsf{H}_4[x]$ and $\mathsf{H}_5[x]$ exactly except for in the invocation of $\mathcal{O}(\cdot)$, where the two challengers differ. For this step, it uses the challenger implied by the obfuscation security definition, and sends the obfuscation challenger the circuits P_4 and P_5 used in $\mathsf{H}_4[x]$ and $\mathsf{H}_5[x]$ respectively. Algorithm $\mathcal{B}$ passes on the output of $\mathcal{A}$ as its own output. If $\mathcal{B}$ receives $\mathcal{O}(P_4)$ from its challenger, it provides a perfect simulation of $\mathsf{H}_4[x]$, and if it receives $\mathcal{O}(P_5)$, it provides a perfect simulation of $\mathsf{H}_5[x]$. Thus $\mathcal{B}$ distinguishes between $\mathcal{O}(P_4)$ and $\mathcal{O}(P_5)$ with the same advantage that $\mathcal{A}$ distinguishes between $\mathsf{H}_4[x]$ and $\mathsf{H}_5[x]$. □

Lemma 28. Suppose that PKE is a CPA-secure public key encryption scheme (Definition 22). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_6[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_5[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes between $\mathsf{H}_6[x]$ and $\mathsf{H}_5[x]$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the CPA security of PKE. During setup, $\mathcal{B}$ sets pk_i to be the public key received from the CPA security adversary. Then it samples $r'' \xleftarrow{R} \mathcal{R}$ as well as the uniformly random values (w^*, r^*, r'^*) before sending the CPA security challenger r^* and r''. It sets ct^* to be the encryption it gets back. From this point on $\mathcal{B}$ behaves identically to the unpredictability challengers in $\mathsf{H}_5[x]$ and $\mathsf{H}_6[x]$, which behave identically after setting the value of ct^* in the circuit P. At the end of the experiment, $\mathcal{B}$ passes on the output of $\mathcal{A}$ as its own output.

Since the only difference between hybrids $\mathsf{H}_5[x]$ and $\mathsf{H}_6[x]$ is in the value of ct^*, $\mathcal{B}$ presents a perfect simulation of the $\mathsf{H}_5[x]$ challenger when it receives an encryption of r^* from the CPA security challenger, and a perfect simulation of the $\mathsf{H}_6[x]$ challenger when it receives an encryption of r'', so long as $\mathcal{A}$ is not given the secret key sk_i. But if $\mathcal{A}$ is given sk_i, then the game always outputs 0 anyway, and the adversary can have no advantage. Thus $\mathcal{B}$ wins the CPA security game with the same advantage that $\mathcal{A}$ distinguishes between $\mathsf{H}_5[x]$ and $\mathsf{H}_6[x]$. □

Lemma 29. Suppose that $\mathcal{O}$ is an indistinguishability obfuscator (Definition 1). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_7[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_6[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. This proof is analogous to the proof of Lemma 27, so we omit a full proof. □

Lemma 30. Suppose that COM is a hiding commitment scheme (Definition 23). Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_8[x](\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{H}_7[x](\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. This proof is analogous to the proof of Lemma 28 except that we invoke the indistinguishability definition associated with the hiding property of the commitment scheme COM instead of that of the CPA-security of encryption. We thus omit a full proof. □

Fairness. The proof of selective fairness is similar to the first part of the proof of unpredictability where we puncture the PRF F at the point where it will be called in the challenge election, replacing its output with a random value.

- H_0: This hybrid corresponds to the real selective fairness experiment $\mathsf{FAIR}[\mathcal{A}, \lambda, \ell, N, c, n]$.
- H_1: In this hybrid, the challenger picks the randomness R to be used in the challenge election in the setup phase before running Setup. The output of this game is identical to the preceding hybrid because R is chosen uniformly at random in both hybrids.

  For the following hybrids, let s^* be the value of $(R, \mathsf{pk}_0, ..., \mathsf{pk}_{n-1})$ to be used in the challenge election.
- H_2: In this hybrid, we modify step 2 of the circuit P where we previously set $(w, r, r') \leftarrow F(k, s)$. We replace this operation with a conditional branch where if $s = s^*$, then $(w, r, r') \leftarrow (w^*, r^*, r'^*)$ for hard-coded values $(w^*, r^*, r'^*) \leftarrow F(k, s^*)$. Otherwise $(w, r, r') \leftarrow F(k(s^*), s)$, where $k(s^*)$ is the key k punctured at s^*. This hybrid is indistinguishable from H_1 by the security of the indistinguishability obfuscator via a proof identical to that of Lemma 25.
- H_3: In this hybrid, instead of setting $(w^*, r^*, r'^*) \leftarrow F(k, s^*)$ when $s = s^*$ in the modified second step of P, we set (w^*, r^*, r'^*) to be uniformly random values. This hybrid is indistinguishable from H_2 by the security of the puncturable PRF F via a proof identical to that of Lemma 26.

Once the outputs of F are replaced by truly random values, it is clear that the winner of the challenge election is chosen uniformly at random from among all participants in the election. Since each user has a copy of $\widetilde{P}$ generated during an honest setup phase, it is not possible for an adversary to tamper with $\widetilde{P}$ in order to change the election result. Thus the probability that any participant wins the election is at most $\frac{1}{n}$, and since the adversary controls c participants, it can produce proof that it controls the winner with probability at most $\frac{c}{n}$. If the adversary does not control the winner, then one of the uncorrupted users can produce a proof that it is the winner. Since H_3 can only be distinguished from the real fairness game with negligible probability, this completes the proof.

C Proof of Theorem 17

Uniqueness. We will first prove uniqueness. Since H is a random function with a large enough output length, no efficient adversary can find values $k_L, k_R \leftarrow H(k)$, $k'_L, k'_R \leftarrow H(k')$, such that $k_L = k'_L$ unless $k = k'$. Colliding values of $k_L = k'_L$ arising from the case where $k = k'$ are ruled out because they would result in $k_R = k'_R$, causing the $\mathsf{RegisterVerify}$ checks to fail when k'_R was added to st.

The process of expanding the $\log N$ random bits into N bits always results in a vector v that is zero at all but one point, so the dot product we compute between v and the user secrets s_i will select exactly one value from the set of user secrets to choose the leader. Decryption of s_i will always succeed so long as there are t uncorrupted users to send valid values of l_i, and by the robustness of the encryption scheme, there will be only one possible decryption of the chosen s_i value. But if there are no values of k_L, k'_L such that $k_L = k'_L$ encrypted among $s_1, ..., s_n$, then each time an element of the list is chosen to select an election winner, there is exactly one (k_L, k_R) pair in st, and therefore exactly one value of k among all the registered users, that can successfully be used to prove leadership.

Unpredictability. Intuitively, TSSLE provides unpredictability because the threshold FHE ensures that no coalition of fewer than t malicious users can reveal the inputs or internal wire values of the circuit C_e. We will prove unpredictability through a series of hybrids. Suppose the adversary $\mathcal{A}$ makes at most $\mathcal{Q}_R = \mathsf{poly}(\lambda)$ requests for an uncorrupted user to register for an election.

- $\mathsf{H}_0[x]$: The real unpredictability game $\mathsf{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$ except the experiment outputs 0 if the challenge election has a winner, but the uncorrupted user $\mathcal{U}_{j^*}$ (the x^{th} uncorrupted user to register) does not win.
- $\mathsf{H}_1[x]$: This hybrid changes the user which the challenger counts as the "winner" of the election. Instead of the winner being the user that can produce a proof of leadership that will be accepted by Verify, the winner is the user $\mathcal{U}_i$ for which $v_i = 1$ when evaluating C_e. This hybrid is indistinguishable from the preceding hybrid because these two definitions of "winner" are identical in the construction.
- $\mathsf{H}_2[x]$: In this hybrid, the experiment outputs 0 if the adversary ever queries the random oracle on the secret k_i of an uncorrupted user who participates in the challenge election. It is otherwise identical to $\mathsf{H}_1[x]$.

  Any values of k_{iL}, k_{iR} belonging to an uncorrupted user $\mathcal{U}_i$ appear independently random to the adversary until k_i is revealed to prove that $\mathcal{U}_i$ has been elected leader, unless the adversary queries H at k_i. Since the view of the adversary is independent of k_i until it is revealed, it queries H at k_i with probability at most $\frac{q}{2^\lambda}$ if it makes q queries. Taking a union bound over the secrets of all uncorrupted users, the probability that the adversary queries H at a point corresponding to any uncorrupted user's secret is at most $\frac{Nq}{2^\lambda} \leq \mathsf{negl}(\lambda)$. As such, no PPT adversary could distinguish between the previous hybrid and this one.
- $\mathsf{H}_3[x]$: In this hybrid, the challenger runs the extractor $\mathcal{E}$ provided by the plaintext extractability of the TFHE scheme to extract the values of k_{iL} for each user as well as their contributions to rk, aborting if extraction fails. This is indistinguishable from the previous hybrid by the plaintext extractability of the TFHE scheme.
- $\mathsf{H}_4[x]$: In this hybrid, the challenger replaces its invocations of $\mathsf{TFHE.Setup}$ with the simulator $\mathcal{S}_1$ and invocations of $\mathsf{TFHE.Eval}(\mathsf{pk}, C_e, \mathsf{rk}, s_1, ..., s_n)$ and $\mathsf{TFHE.PartDec}(\mathsf{pk}, p_i, \mathsf{sk}_i)$ with the simulator $\mathcal{S}_2(C_e, \{\mathsf{rk}, s_1, ..., s_n\}, C_e(r, k_{1L}, ..., k_{nL}), M)$ (using the plaintext and randomness values obtained from the extractor to send the plaintexts and randomnesses required by the real and ideal experiments and to evaluate C_e). These simulators are guaranteed to exist and be indistinguishable from the functions they replace by the simulation security of TFHE.
- $\mathsf{H}_5[x]$: In this hybrid, the challenger replaces its invocations of $\mathsf{TFHE.Encrypt}(\mathsf{pk}, k_{j^*L})$ during registration of the winner of the challenge election with invocations of $\mathsf{TFHE.Encrypt}(\mathsf{pk}, r')$, where $r' \xleftarrow{\mathrm{R}} \mathbb{F}_2^\lambda$ is a freshly chosen random value. The input to the simulator is unchanged from $\mathsf{H}_4[x]$ – only the values $s_{j^*}, \pi_{s_j^*}$ reflect this change. This hybrid is indistinguishable from the preceding hybrid by the semantic security of TFHE.

  We could use an adversary $\mathcal{A}$ that distinguishes between $\mathsf{H}_4[x]$ and $\mathsf{H}_5[x]$ to construct an adversary $\mathcal{B}$ that wins the semantic security game. $\mathcal{B}$ acts as the challenger in the unpredictability game while also playing as the adversary in the semantic security game. $\mathcal{B}$ sends the semantic security challenger the plaintexts r' and k_{j^*L} as potential challenges, and gets back an encryption ct and a proof π_s. It reproduces the unpredictability game of the preceding hybrid exactly except it appends ct and π_s to st at the end of registration. At the end of the unpredictability game, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own output for the semantic security game. The semantic security challenger's choice of challenge determines which of the two successive hybrids $\mathcal{A}$ interacts with, so $\mathcal{B}$ wins the semantic security game with exactly the same advantage that $\mathcal{A}$ distinguishes between the hybrids.

From $\mathsf{H}_5[x]$, the output of the circuit C_e in the challenge election will be a uniformly random value different from any registered user's registration secret. Thus the view of the adversary $\mathcal{A}$ is independent of the winner of the challenge election, and it cannot do better than guessing the index of the election winner with probability $\frac{1}{n-c}$ (if the winner is uncorrupted).

Since no efficient adversary $\mathcal{A}$ can distinguish between each pair of hybrids above with more than negligible advantage, we have that

$$\left|\mathsf{Pr}[\mathsf{H}_5[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M]\right| \leq \mathsf{negl}(\lambda).$$

Next, since all our hybrids were parameterized by the condition that $\mathcal{U}_{j^*}$, the x^{th} uncorrupted user to register, wins the challenge election, we take the union bound over all $\mathcal{Q}_R$ registrations to get

$$\begin{aligned}
&\mathsf{Pr}[\mathsf{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]= 1 \mid i \in [N] \setminus M] \\
&= \frac{1}{n-c} + \Sigma_{x=1}^{\mathcal{Q}_R} \Big| \mathsf{Pr}[\mathsf{H}_5[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] \Big| \\
&\leq \frac{1}{n-c} + \Sigma_{x=1}^{\mathcal{Q}_R} \mathsf{negl}(\lambda) \\
&\leq \frac{1}{n-c} + \mathcal{Q}_R \mathsf{negl}(\lambda).
\end{aligned}$$

This completes the proof of unpredictability since $\mathcal{Q}_R \mathsf{negl}(\lambda)$ is still negligible in λ.

Fairness. Intuitively, TSSLE provides fairness because generating the PRF key inside the threshold FHE ensures that no coalition of fewer than t malicious users can see the key. Thus the choice of leader should appear random to any set of fewer than t malicious users. We will prove fairness through a series of hybrids.

- H_0: The real fairness game $\mathsf{FAIR}[\mathcal{A}, \lambda, \ell, N, c, n]$.
- H_1: Same as H_0, except the experiment outputs 0 if the adversary ever queries the random oracle on the secret k_i of an uncorrupted user who participates in the challenge election.
  Any values of k_{iL}, k_{iR} belonging to an uncorrupted user $\mathcal{U}_i$ appear independently random to the adversary until k_i is revealed to prove that $\mathcal{U}_i$ has been elected leader, unless the adversary queries H at k_i. Since the view of the adversary is independent of k_i until it is revealed, it queries H at k_i with probability at most $\frac{q}{2^\lambda}$ if it makes q queries. Taking a union bound over the secrets of all uncorrupted users, the probability that the adversary queries H at a point corresponding to any uncorrupted user's secret is at most $\frac{Nq}{2^\lambda} \leq \mathsf{negl}(\lambda)$. As such, no PPT adversary could distinguish between the previous hybrid and this one.
- H_2: Same as H_1, except the challenger runs the extractor $\mathcal{E}$ provided by the plaintext extractability of the TFHE scheme to extract the values of k_{iL} for each user as well as their contributions to rk, aborting if extraction fails. This is indistinguishable from the previous hybrid by the plaintext extractability of the TFHE scheme.
- H_3: Same as H_2, except the challenger replaces its invocations of $\mathsf{TFHE.Setup}$ with the simulator $\mathcal{S}_1$ and invocations of $\mathsf{TFHE.Eval}(\mathsf{pk}, C_e, \mathsf{rk}, s_1, ..., s_n)$ and $\mathsf{TFHE.PartDec}(\mathsf{pk}, p_i, \mathsf{sk}_i)$ with the simulator $\mathcal{S}_2(C_e, \{\mathsf{rk}, s_1, ..., s_n\}, C_e(r, k_{1L}, ..., k_{nL}), M)$ (using the plaintext and randomness values obtained from the extractor to send the plaintexts and randomnesses required by the real and ideal experiments and to evaluate C_e). These simulators are guaranteed to exist and be indistinguishable from the functions they replace by the simulation security of TFHE.
- H_4: Same as H_3, except the challenger outputs 0 if the adversary outputs a proof π_j containing values $l_1, ..., l_t$ such that $\mathsf{TFHE.VerifyDec}(\mathsf{pk}, l_i, p_i)= 1$ for all $i \in [t]$ but $\mathsf{TFHE.FinDec}(\mathsf{pk}, p_i, \{l_1, ..., l_t\})$ is not equal to the value given to $\mathcal{S}_2$ as the simulated output of the circuit C_e. Such a set $l_1, ..., l_t$ could be used to win the robustness game because it is a second set of verified partial decryptions on which $\mathsf{TFHE.FinDec}$ gives a different final decryption of p_i than it did on the simulated partial decryptions. Thus this hybrid is indistinguishable from the preceding hybrid by the robustness of TFHE.
- H_5: Same as H_4, except the challenger replaces the input $C_e(r, k_{1L}, ..., k_{nL})$ given to $\mathcal{S}_2$ with an evaluation of $C'_e(r, k_{1L}, ..., k_{nL})$ where the evaluation of the PRF $f(\mathsf{rk}, \cdot)$ has rk replaced with a hard-coded random value r^* corresponding to the decryption of rk. This function computes an output identical to C_e, so the hybrid is identical to the preceding one.
- H_6: Same as H_5, except the challenger replaces the ciphertext rk with a ciphertext $\mathsf{rk'} = \mathsf{TFHE.Encrypt}(\mathsf{pk}, 0)$. This hybrid is indistinguishable from the preceding hybrid by the semantic security of TFHE.
  We could use an adversary $\mathcal{A}$ that distinguishes between H_5 and H_6 to construct an adversary $\mathcal{B}$ that wins the semantic security game. $\mathcal{B}$ acts as the challenger in the unpredictability game while also playing as the adversary in the semantic security game. $\mathcal{B}$ sends the semantic security challenger the plaintexts 0 and r (the plaintext corresponding to rk) as potential challenges, and gets back an encryption ct and

a proof π_s. It reproduces the unpredictability game of the preceding hybrid exactly except it uses ct in place of rk. At the end of the unpredictability game, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own output for the semantic security game. The semantic security challenger's choice of challenge determines which of the two successive hybrids $\mathcal{A}$ interacts with, so $\mathcal{B}$ wins the semantic security game with exactly the same advantage that $\mathcal{A}$ distinguishes between the hybrids.

– H_7: Same as H_6, except the challenger replaces the input $C'_e(r, k_{1L}, ..., k_{nL})$ given to $\mathcal{S}_2$ with an evaluation of $C''_e(r, k_{1L}, ..., k_{nL})$ where the evaluation of the PRF $f(r^*, \cdot)$ is replaced with invocations of a random function $F(\cdot)$. This is indistinguishable from the previous hybrid by the weak PRF security of f.
We could use an adversary $\mathcal{A}$ that distinguishes between the outputs of H_6 and H_7 to construct another adversary $\mathcal{B}$ that wins the weak PRF security game. $\mathcal{B}$ acts as the challenger in the unpredictability game while also playing as the adversary in the weak PRF security game. It reproduces the unpredictability game of H_6 exactly except that in the description and evaluation of circuit that it gives to $\mathcal{S}$, any query to $f(r^*, \cdot)$ is forwarded to the PRF challenger and has its output replaced with the PRF challenger's output. At the end of the unpredictability game, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own output for the PRF security game.
Since $\mathcal{A}$ never sees r^* (it is chosen randomly by the challenger and only used in its input to $\mathcal{S}_2$), the evaluation of $f(r^*, \cdot)$ is on a key unknown to it. f is only ever evaluated on R, a public random value. As such, if $\mathcal{B}$ is interacting with a PRF f, then it provides $\mathcal{A}$ with exactly H_6. On the other hand, if $\mathcal{B}$ is interacting with a random function F, then $\mathcal{A}$ sees exactly H_7. If $\mathcal{A}$ distinguishes between H_6 and H_7 with non-negligible advantage, then $\mathcal{B}$ distinguishes between the weak PRF f and a random function with non-negligible advantage, breaking the weak PRF security of f.

From hybrid H_7, $\mathcal{A}$ wins the FAIR game with probability $c/n + \mathsf{negl}(\lambda)$ and therefore has negligible advantage. Since the vector u is chosen by a random function $F(\cdot)$, the non-zero index of the vector v output by $\mathsf{Expand}(u)$ is chosen uniformly at random as well. This is because for each possible output of $F(\cdot)$, Expand returns a different value of v. Thus the value s_i selected by v is chosen uniformly at random too, and the value of k_{iL} revealed (of which there can only be one or else the challenger aborts) belongs to a random user, and that random user is the winner of the election. Since the winner is chosen uniformly at random among the users, the adversary can only produce the proof needed to win the fairness game if the winner is corrupted (probability c/n) or if it guesses the correct proof (with probability $\mathsf{negl}(\lambda)$). If the adversary does not control the winner, then one of the uncorrupted users can produce a proof that it is the winner.

Since the adversary in H_7 wins the fairness game with negligible advantage, the advantage of the adversary in H_0 is the sum of its advantage in distinguishing between each successive pair of hybrids plus $\mathsf{negl}(\lambda)$, each of which is itself negligible. Thus the adversary wins the original fairness game with probability $c/n+\mathsf{negl}(\lambda)$, completing the proof.

D Proofs of Theorems 19 and 20

Uniqueness. Since H is a random function with a large enough output length, no efficient adversary can find values $k_L, k_R \leftarrow H(k)$, $k'_L, k'_R \leftarrow H(k')$, such that $k_L = k'_L$ unless $k = k'$. Colliding values of $k_L = k'_L$ arising from the case where $k = k'$ are ruled out because they would result in $k_R = k'_R$, causing the $\mathsf{RegisterVerify}$ checks to fail when k'_R was added to st. But if there are no duplicate values of k_L in l corresponding to different values of k_R, then each time an element of l is chosen to select an election winner, there is exactly one k_R in st, and therefore exactly one corresponding k, that can successfully be used to prove leadership.

Fairness. Fairness follows directly from the construction, as exactly one entry is selected uniformly at random to be the leader in each election. The checks done in $\mathsf{RegisterVerify}$ ensure that each entry in l belonging to an uncorrupted user corresponds to a different secret k_{iL} and therefore to a different user. Thus any given uncorrupted user has probability $\frac{1}{n}$ of being elected, so the adversary, who controls c users, controls the winner of the election with probability at most $\frac{c}{n}$. If the adversary does not control the winner, then one of the uncorrupted users can produce a proof that it is the winner.

Unpredictability. We will prove unpredictability through a series of hybrids. Suppose there are at most $\mathcal{Q}_R = \mathsf{poly}(\lambda)$ registrations of uncorrupted users in the elections phase of the security experiment.

- $\mathsf{H}_0[x]$: The real unpredictability game $\mathsf{UNPRED}[\mathcal{A}, \lambda, \ell, N, n, c]$ with an additional abort condition defined as follows. Let b^* be the bucket from which the winner of the challenge election is chosen. The experiment aborts if the x^{th} registration is not the last registration of an uncorrupted user into bucket b^* before the challenge election. Note that, so long as there is a single uncorrupted user in bucket b^* during the challenge election, such a registration will always exist, and otherwise the winner of the challenge election will not be an uncorrupted user.

- $\mathsf{H}_1[x]$: This hybrid changes the user which the challenger counts as the "winner" of the election. Instead of the winner being the user that can produce a proof of leadership that will be accepted by Verify, the winner is the user $\mathcal{U}_i$ for which $u^{k_{iL}} = v$, where $(u, v) \in l$ is the entry chosen by the election randomness R. This hybrid is indistinguishable from the preceding hybrid because these two definitions of "winner" are identical in the construction.

- $\mathsf{H}_2[x]$: In this hybrid, the experiment outputs 0 if the adversary ever queries the random oracle on the secret k_i of an uncorrupted user who participates in the challenge election.

  Any values of k_{iL}, k_{iR} belonging to an uncorrupted user $\mathcal{U}_i$ appear independently random to the adversary until k_i is revealed to prove that $\mathcal{U}_i$ has been elected leader, unless the adversary queries H at k_i. Since the view of the adversary is independent of k_i until it is revealed, it queries H at k_i with probability at most $\frac{q}{2^\lambda}$ if it makes q queries. Taking a union bound over the secrets of all uncorrupted users, the probability that the adversary queries H at a point corresponding to any uncorrupted user's secret is at most $\frac{Nq}{2^\lambda} \leq \mathsf{negl}(\lambda)$. As such, no PPT adversary could distinguish between H_1 and H_2.

- $\mathsf{H}_3[x]$: In this hybrid, the challenger chooses the value of R to be used in the challenge election during the setup phase instead of during the challenge phase, so the challenger knows at setup time which bucket b^* the leader will be chosen from in the challenge election. This hybrid is indistinguishable from H_2 because it makes no changes to the distribution of messages sent by the challenger. That is, it is identical to H_2 in terms of the adversary's view.

- $\mathsf{H}_4[x]$: In this hybrid, we change the behavior of the challenger in the x^{th} registration. During this registration, instead of replacing each $l_{j\cdot b^*} = (u_{j\cdot b^*}, v_{j\cdot b^*}) = (u_{j\cdot b^*}, u_{j\cdot b^*}^{k_{i'L}})$ with $l_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{k_{i'L} r_j})$ for $r_j \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, for entries corresponding to the secrets $k_{i'L}$ of uncorrupted users $\mathcal{U}_{i'}$, it sets $l_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{k^*_{i'L} r_j})$, for a new random key $k^*_{i'L} \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, which from then on plays the role of $k_{i'L}$ in determining whether the user $\mathcal{U}_{i'}$ has won an election.

  We show that this hybrid is indistinguishable from H_3, assuming the DDH assumption holds in $\mathbb{G}$, in Lemma 31.

In Lemma 32 below, we show that an adversary $\mathcal{A}$ wins the unpredictability game in $\mathsf{H}_4[x]$ with probability at most $\frac{1}{\sqrt{N}-c}$.

Since no efficient adversary $\mathcal{A}$ can distinguish between each pair of hybrids above with more than negligible advantage, we have that

$$\left|\mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M]\right| \leq \mathsf{negl}(\lambda).$$

Next, since all our hybrids were parameterized by x, we take the union bound over all $\mathcal{Q}_R$ uncorrupted registrations to get

$$\begin{aligned}
&\mathsf{Pr}[\mathsf{UNPRED}[\mathcal{A},\lambda,\ell,N,n,c]=1 \mid i \in [N] \setminus M] \\
&\leq \frac{1}{\sqrt{N}-c} + \Sigma_{x=1}^{\mathcal{Q}_R}\left|\mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A})=1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_0[x](\mathcal{A})=1 \mid i \in [N] \setminus M]\right| \\
&\leq \frac{1}{\sqrt{N}-c} + \Sigma_{x=1}^{\mathcal{Q}_R}\mathsf{negl}(\lambda) \\
&\leq \frac{1}{\sqrt{N}-c} + \mathcal{Q}_R\mathsf{negl}(\lambda) \\
&\leq \frac{1}{\sqrt{N}-c} + \mathsf{negl}(\lambda).
\end{aligned}$$

This completes the proof of unpredictability. We now state and prove the remaining Lemmas used in the proof above. For a hybrid experiment $\mathsf{H}[x]$ and an adversary $\mathcal{A}$, we use $\mathsf{H}[x](\mathcal{A})$ to denote the random variable that represents the output of experiment $\mathsf{H}[x]$ with adversary $\mathcal{A}$.

Lemma 31. Suppose that $\mathbb{G}$ is a group in which the DDH problem is hard. Then, for all efficient adversaries $\mathcal{A}$, we have

$$\mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A})=1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_3[x](\mathcal{A})=1 \mid i \in [N] \setminus M] \leq \sqrt{N}\mathsf{negl}(\lambda) = \mathsf{negl}(\lambda).$$

Proof. We prove this lemma through a sequence of $\sqrt{N}$ inner hybrids $\mathsf{H}_{3,\gamma}[x], \gamma \in [\sqrt{N}]$, defined as follows:

- $\mathsf{H}_{3,\gamma}[x]$: In this hybrid, we change the behavior of the challenger in the x^{th} registration. During this registration, if $j < \gamma$, instead of replacing each $l_{j\cdot b^*} = (u_{j\cdot b^*}, v_{j\cdot b^*}) = (u_{j\cdot b^*}, u_{j\cdot b^*}^{k_{i'L}})$ with $l_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{k_{i'L}r_j})$ for $r_j \xleftarrow{R} \mathbb{Z}_q$, for entries corresponding to the secrets $k_{i'L}$ of uncorrupted users $\mathcal{U}_{i'}$, it sets $l_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{k^*_{i'L}r_j})$, for a new random key $k^*_{i'L} \xleftarrow{R} \mathbb{Z}_q$, which from then on plays the role of $k_{i'L}$ in determining whether the user $\mathcal{U}_{i'}$ has won an election.

By definition, we have that $\mathsf{H}_{3,0}[x]$ is identical to $\mathsf{H}_3[x]$ and $\mathsf{H}_{3,\sqrt{N}}[x]$ is identical to $\mathsf{H}_4[x]$. To prove the lemma, we prove that each successive pair of inner hybrids, $\mathsf{H}_{3,\gamma-1}[x]$ and $\mathsf{H}_{3,\gamma}[x]$, are indistinguishable.

Claim. Suppose that $\mathbb{G}$ is a group in which the DDH problem is hard. Then, for all efficient adversaries $\mathcal{A}$ that make at most $\mathcal{Q}_R$ registrations of honest users in the elections phase, we have

$$\mathsf{Pr}[\mathsf{H}_{3,\gamma-1}[x](\mathcal{A})=1 \mid i \in [N] \setminus M] - \mathsf{Pr}[\mathsf{H}_{3,\gamma}[x](\mathcal{A})=1 \mid i \in [N] \setminus M] \leq \mathcal{Q}_R\mathsf{negl}(\lambda) = \mathsf{negl}(\lambda).$$

First, if the entry $l_{\gamma\cdot b^*}$ does not correspond to a secret $k_{i'}$ of an uncorrupted user $\mathcal{U}_{i'}$, the two hybrids are identical. Thus, we only consider the case where this entry corresponds to the secret of an uncorrupted user.

Let $\mathcal{A}$ be an adversary that distinguishes between $\mathsf{H}_{3,\gamma-1}[x]$ and $\mathsf{H}_{3,\gamma}[x]$. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks DDH in $\mathbb{G}$. Algorithm $\mathcal{B}$ begins by receiving a DDH challenge tuple (u^*, v^*, w^*) from the DDH challenger. Then $\mathcal{B}$ behaves as the challenger in $\mathsf{H}_{3,\gamma}[x]$ except it guesses a registration index y and for the y^{th} registration of an uncorrupted user (call this user $\mathcal{U}^*$), and instead of having user $\mathcal{U}^*$ add $(g^r, g^{rk_{iL}})$ to l, it adds (g^r, u^{*r}).

Next, during the x^{th} registration, if $\mathcal{B}$ knows the secret $k_{i'L}$ used in entry $l_{\gamma\cdot b^*} = (u_{\gamma\cdot b^*}, v_{\gamma\cdot b^*}) = (u_{\gamma\cdot b^*}, u_{\gamma\cdot b^*}^{k_{i'L}})$ from registration of user $\mathcal{U}_{i'}$, $\mathcal{B}$ aborts and outputs 0. Note that the only time that $\mathcal{B}$ does not abort and $i \in [N] \setminus M$ is when entry $l_{\gamma\cdot b^*}$ corresponds to user $\mathcal{U}^*$. If $\mathcal{B}$ does not abort, instead of setting $l_{\gamma\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{k^*_{i'L}r_j})$, it sets $l_{\gamma\cdot b^*} \leftarrow (v^*, w^*)$.

During the challenge experiment, if $\mathcal{B}$ does not know the secret of the entry chosen for the winner, it outputs the index of $\mathcal{U}^*$. Note that whenever this happens, either the index $i \notin [N] \setminus M$ (for i being the

index of the winner), or the winner is $\mathcal{U}^*$. At the end of the experiment, $\mathcal{B}$ outputs 1 iff the unpredictability experiment outputs 1, i.e., if $\mathcal{A}$ guesses the index of the winner of the challenge election.

Observe that if $\mathcal{B}$ does not abort, it provides $\mathcal{A}$ a perfect simulation of when the DDH challenger sets $w^* = g^{\alpha\beta}$, for $u^* = g^\alpha, v^* = g^\beta$ with $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, and a perfect simulation of $\mathsf{H}_{3,\gamma}[x]$ when $w^* = g^{\gamma'}$ for $\gamma' \xleftarrow{R} \mathbb{Z}_q$, conditioned on the winner's index $i \in [N] \setminus M$. Since $y \xleftarrow{R} \mathcal{Q}_R$ is chosen uniformly at random from among all uncorrupted registrations, there is a $\frac{1}{\mathcal{Q}_R}$ chance that $\mathcal{B}$ does not abort. Thus $\mathcal{B}$ distinguishes between the DDH experiments with probability $\frac{1}{\mathcal{Q}_R}$ times the probability that $\mathcal{A}$ distinguishes between $\mathsf{H}_{3,\gamma-1}[x]$ and $\mathsf{H}_{3,\gamma}[x]$, completing the proof of the claim.

□

Lemma 32. For all adversaries $\mathcal{A}$, we have (unconditionally) that

$$\mathsf{Pr}[\mathsf{H}_4[x](\mathcal{A}) = 1 \mid i \in [N] \setminus M] \leq \frac{1}{\sqrt{N} - c}.$$

Proof. In $\mathsf{H}_4[x]$, all the uncorrupted users' entries in bucket b^* appear random, i.e., as $(g^a, g^b), a, b \xleftarrow{R} \mathbb{Z}_q$. Thus the contents bucket b^* are distributed independently of the "winning" user $\mathcal{U}_i$. Thus the adversary $\mathcal{A}$ can do no better than choosing an uncorrupted user at random from those users registered in bucket b^*. In the worst case, every corrupted user is in bucket b^*, so the $\mathcal{A}$ wins the unpredictability game in $\mathsf{H}_4[x]$ with probability at most $\frac{1}{\sqrt{N}-c}$.

□

Security for randomly-assigned buckets (Theorem 20). We now describe how to modify the security analysis above to apply to a scheme where users' buckets are assigned randomly at registration time. The arguments for uniqueness and fairness will be exactly the same, but the unpredictability argument will require an additional step in the portion of the proof corresponding to Lemma 32. The reasoning in that lemma proves the claim that this scheme is $1/\hat{h}$-unpredictable, where $\hat{h}$ is the minimum number of honest users in any one bucket. We show that $\hat{h} \leq \frac{h}{\sqrt{N}} - \sqrt{\frac{2\lambda h}{\sqrt{N}}}$ with probability at most $e^{-\lambda}$, where h is the total number of honest users.

The probability that a given user is assigned to a particular bucket is $\frac{1}{\sqrt{N}}$, and users are assigned to buckets independently, resulting in $\frac{h}{\sqrt{N}}$ honest users per bucket in expectation. Thus, by a Chernoff bound, the number of honest users assigned to one particular bucket is bounded by

$$\Pr\left[\hat{h} \leq (1-\delta)\frac{h}{\sqrt{N}}\right] \leq e^{\frac{-\delta^2 h}{2\sqrt{N}}}.$$

Setting $\delta = \sqrt{\frac{2\lambda\sqrt{N}}{h}}$ yields

$$\Pr\left[\hat{h} \leq \frac{h}{\sqrt{N}} - \sqrt{\frac{2\lambda h}{\sqrt{N}}}\right] \leq e^{-\lambda} = \mathsf{negl}(\lambda).$$

We next take a union bound over the $\sqrt{N}$ buckets to find the probability than any bucket has fewer than $\hat{h}$ honest users, but since we showed that a single bucket only has fewer than $\frac{h}{\sqrt{N}} - \sqrt{\frac{2\lambda h}{\sqrt{N}}}$ honest users with negligible probability, the union bound over $\sqrt{N}$ buckets will also have fewer honest users in any bucket with at most negligible probability. Plugging back in to the claim proved above, we get that the scheme is

$$1/\hat{h} \geq \frac{1}{\frac{h}{\sqrt{N}} - \sqrt{\frac{2\lambda h}{\sqrt{N}}}} = \frac{N^{1/4}}{N^{3/2} - c\sqrt{N} - \sqrt{2\lambda(N-c)}}\text{-unpredictable.}$$

The (informal) statement of Theorem 20 follows from using the steps above and substituting this bound for the one found in Lemma 32.