A Framework for Constructing Single Secret Leader Election from MPC

Michael Backes[1], Pascal Berrang[2], Lucjan Hanzlik[1], and Ivan Pryvalov[1,3](✉)

[1] CISPA Helmholz Center for Information Security, Saarbrücken, Germany
{backes,hanzlik}@cispa.de
[2] University of Birmingham, Birmingham, UK
P.P.Berrang@bham.ac.uk
[3] University of Luxembourg, Esch-sur-Alzette, Luxembourg
ivan.pryvalov@uni.lu

Abstract. The emergence of distributed digital currencies has raised the need for a reliable consensus mechanism. In proof-of-stake cryptocurrencies, the participants periodically choose a closed set of validators, who can vote and append transactions to the blockchain. Each validator can become a leader with the probability proportional to its stake. Keeping the leader private yet unique until it publishes a new block can significantly reduce the attack vector of an adversary and improve the throughput of the network. The problem of Single Secret Leader Election (SSLE) was first formally defined by Boneh et al. in 2020.

In this work, we propose a novel framework for constructing SSLE protocols, which relies on secure multi-party computation (MPC) and satisfies the desired security properties. Our framework does not use any shuffle or sort operations and has a computational cost for N parties as low as $O(N)$ of basic MPC operations per party. We improve the state-of-the-art for SSLE protocols that do not assume a trusted setup. Moreover, our SSLE scheme efficiently handles weighted elections. That is, for a total weight S of N parties, the associated costs are only increased by a factor of $\log S$. When the MPC layer is instantiated with techniques based on Shamir's secret-sharing, our SSLE has a communication cost of $O(N^2)$ which is spread over $O(\log N)$ rounds, can tolerate up to $t < N/2$ of faulty nodes without restarting the protocol, and its security relies on DDH in the random oracle model. When the MPC layer is instantiated with more efficient techniques based on garbled circuits, our SSLE requires all parties to participate, up to $N-1$ of which can be malicious, and its security is based on the random oracle model.

1 Introduction

In 2008, Bitcoin [21] laid the foundation for the increasingly important areas of cryptocurrencies and distributed ledgers. One of the main advantages of distributed ledgers is that there is no single central authority that controls the transaction flow (*censorship resistance*). Anyone can access the public ledger, which is a sequence of blocks that contains transactions. For example, in Bitcoin, participants called "miners" are randomly selected to produce and append

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 672–691, 2022.
https://doi.org/10.1007/978-3-031-17146-8_33

a new block to the chain. This selection process relies on the "proof-of-work" concept (PoW). To append a block to the chain, the participant has to find a value, such that a cryptographic hash function is evaluated below some threshold.

To avoid extreme energy consumption induced by PoW protocols [22], an alternative approach, "proof-of-stake" (PoS), has been proposed. Here, the probability of being selected for appending the chain depends on the stake (i.e., coins) a party owns. It does not matter whether the party owns an account with some stake v, or several accounts whose accumulated stake amounts to v. The protocol consensus works as long as the majority of all stake is controlled by honest users.

In cryptocurrencies based on proof-of-stake [15,16,19,20], a single party that produces a block is chosen randomly from a set of participants, called validators (which is the equivalent to miners in a PoW protocol). In a PoS cryptocurrency there could be potentially thousands or millions users, who may come and go. It is up to a PoS protocol to determine and fix a relatively small (typically tens or hundreds) set of validators [19] from which a validator is selected that can append a block within a given time frame. To create a consistent picture for all validators, this selection has to be deterministic, but pseudo-random – properties often achieved by relying on Verifiable Random Functions (VRF). However, if an adversary knows in advance which of the validators is selected, it can launch a targeted attack and cause a denial-of-service.

Previous approaches to solving this issue aim to run the selection process in private, with the selected participant publishing a proof alongside the block. Until recently, these approaches failed to guarantee only a single participant to be chosen [19]. After much interest in a solution that provides such a guarantee [25], Boneh et al. proposed a formal definition and several instantiations of a Single Secret Leader Election [4].

The primary motivation of having a single leader is a simple consensus design, as there are no forks in the blockchain (assuming some reasonable connectivity between parties). This property encourages the leader to solely perform heavy computations, which may even exceed the running time of SSLE and/or require multiple cores. For example, the leader's task may consist of prover-heavy computations, whereas verification is very fast (SNARKs). Many protocols (e.g., [15,20]) assume uniqueness, and it is easy to update them with a SSLE solution. They may require a full redesign if the uniqueness assumption no longer holds.

1.1 Our Contribution

1. In this work, we propose a framework for constructing an efficient Single Secret Leader Election (SSLE), which relies on secure multi-party computation (MPC). We formulate a simulation-based definition of the SSLE problem.
2. We present two instantiations of our framework, which improve the state-of-the-art for SSLE protocols that do not require a trusted setup. The first instantiation a t-threshold SSLE scheme that is based on Shamir's secret sharing in the random oracle model. We prove that our construction is secure in the honest-but-curious and malicious adversary models. For the latter, we additionally assume DDH. For N parties, the leader election requires $O(\log N)$

communication rounds and $O(N)$ of basic operations on the underlying primitives. Furthermore, we instantiate our SSLE scheme using the MPC framework by Wang et al. [29], which is secure against any number of malicious parties and is more scalable, but requires all parties to be online.

3. Our SSLE framework can efficiently handle arbitrary stake distributions. For N parties and the overall sum of their stake units S, our construction achieves $O(N \log S)$ cost of the election. Compared with a standard multi-registration technique, in which a party registers multiple times for the election proportionally to her stake, this cost may go up to $O(S)$, which makes our solution exceptionally efficient if $N << S$.
4. We implemented and microbenchmarked our solution using two different MPC frameworks. The performance evaluation indicates that our DDH-based SSLE protocol can be used in practical scenarios up to 30–40 parties when instantiated with the textbook $O(N^2)$ techniques using the verifiable secret sharing scheme (VSS). Furthermore, we implemented our SSLE in the MPC framework based on garbled circuits [29]. The overall time to set up and complete the protocol for 128 parties in a practical scenario is less than 7 min.

Note that, due to space limitations, we refer to the full-version [3] for most proofs. Only the security analysis can be found in the appendix.

1.2 Background

The idea of proof-of-stake was first discussed on the Bitcoin forum[1] in 2011. Kiayaias et al. presented a provably-secure PoS protocol “Ouroboros” at CRYPTO 2017 [20], in which the participants that produce the blocks are elected publically. Such a leader election may be public as in Ouroboros or private as in Algorand [19]. In a private leader election, each node needs to check whether it will be the next leader using its private information but then can prove to others using only public information that it is indeed the next leader. Such a design makes it impossible for others to predict and carry out DoS attacks against the next leader until it is too late.

Algorand achieves this private leader election using Verifiable Random Functions, for which a participant has to prove the outcome to be below a certain threshold. This, however, can result in either no participant or multiple participants being elected. Another protocol employing a private leader election has been formalized by Ganesh et al., whose protocol Ouroboros Praos [16] does not guarantee existence and uniqueness of the leader either.

To mitigate these shortcomings of previous private leader elections, a problem statement of a single secret leader election was first posed at a GitHub page [25] in the form of a research proposal in the context of the Filecoin cryptocurrency. The protocol’s goal is to elect a *single* leader among a finite set of participants. Moreover, the protocol should be reasonably efficient, i.e., on-chain $O(\log n)$ bits per block, $O(n)$ communication complexity (per active party).

[1] https://bitcointalk.org/index.php?topic=27787.0 (accessed 31.01.2022).

Table 1. Comparison of SSLE protocols, assuming all N users participate in election, amortized per one election. On-chain asymptotics include a security parameter λ; PEKS-based on-chain asymptotic is shown assuming the parameter choice suggested in [7].

Construction	Assumptions	Security notion	Setup	Rounds	Computation/ Communication	On-chain
Obfuscation-based [4]	iO	Game-based	Trusted	0 + beacon	$O(\lambda)$, feasibility result	$O(1)$
TFHE-based [4]	TFHE, weak PRF	Game-based, t-threshold	Trusted	1 + beacon	Depends on a particular instance	$O(N)$
Shuffle-based [4]	ROM, DDH	Game-based, *weak unpredictability*	–	1 + beacon	$O(\sqrt{N})$ pub./group el.	$O(\sqrt{N})$
Shuffle-based [4]	ROM, DDH	Game-based	–	1 + beacon	$O(N)$ pub./group el	$O(N)$
PEKS-based [7]	ROM, SXDH	UC, t-threshold	Trusted	≤ 2 + beacon	$O(N)$ pub./group el	$O(\log^2 N)$
Our Construction 1	ROM, DDH	UC, t-threshold	–	$O(\log N)$	$O(N)$ MPC op.	$O(1)$
Our Construction 2	ROM	UC	-	$O(\log N)$	$O(N)$ MPC op.	$O(1)$

1.3 Related Work

Following this call, Boneh et al. [4] formalized the problem of Single Secret Leader Election (SSLE) and presented three constructions: 1) a feasibility result based on indistinguishability obfuscation, 2) a construction based on threshold fully homomorphic encryption (TFHE), and 3) a construction based on DDH that achieves a weaker notion of security. Subsequently, Catalano et al. [7] proposed a UC-secure SSLE based on public key encryption with keyword search (PEKS).

We begin by first comparing how arbitrary stake distributions are handled in previous and our work. While a scenario with equal stakes is easier to analyze, in practice one has to also account for arbitrary stake distributions and how they affect the overall performance of the scheme. Boneh et al. [4] suggest a multi-registration technique (one registration corresponds to one unit of stake) to address arbitrary stake distributions, which makes the associated costs grow linearly with the user's stake. In contrast, our construction offers a more efficient tree-based solution to this setting with the associated costs grow logarithmically in the total stake S of participating parties.

We compare our constructions with Boneh et al. [4] and Catalano et al. [7] in Table 1. By *pub.* we denote the number of public key operations such as exponentiation, by *MPC op.* we denote basic MPC operations such as multiplication. The most notable differences are that (1) our scheme does require neither a trusted setup nor a randomness beacon, and (2) requires only a constant amount of data to be posted on-chain.

In the discussed schemes except for iO- and PEKS-based the leader has to re-register before next election, since she reveals a secret that was generated and used for the registration.

Concurrently to our work, Catalano et al. [8] revisit the shuffle-based SSLE realization from [4] and propose two UC-secure SSLE constructions from DDH.

Their first construction is secure against static adversaries and their second achieves adaptive security with erasures.

On the Practicality of Our SSLE Framework. The number of validators depends on the PoS protocol and can vary from dozens to a few hundred and in limited cases thousands. It does not necessarily correlate with the total number of users. Stake disbalances also vary, and therefore they need to be approximated in our framework by a tree of a sufficient height (Sect. 7.1). Our tree optimization technique has a better effect when applied to a smaller set of validators.

In our SSLE framework, we rely on existing MPC techniques. If a more efficient MPC protocol than the ones used in our constructions emerges, it will help to further improve the running time of the SSLE.

2 Definitions

2.1 Preliminaries

DDH Assumption [13]. Let g be a generator of a group G of a prime order q. For any probabilistic polynomial time (PPT) machine $\mathcal{A}$ and $(x, y, z) \leftarrow (\mathbb{Z}_q)^3$, $|Pr[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - Pr[\mathcal{A}(g, g^x, g^y, g^z) = 1]| \leq negl(\lambda)$.

Secret Sharing. Secret sharing schemes allow a dealer to share a secret s among parties such that later a qualified set of parties can jointly reconstruct s, whereas a non-qualified set of parties learns no information about it. We use Shamir's Secret Sharing [27], which is a t-threshold scheme. We denote Share a protocol to share a secret x as $[x]$, and Rec to reconstruct x from $[x]$. Whereas Shamir's Secret Sharing is only secure against passive adversaries, Verifiable Secret Share (VSS) schemes [23] can protect against active.

Communication and Adversary Models. We assume secure point-to-point communication channels between parties. An adversary is allowed to corrupt up to $t < N$ parties. We consider two models of adversaries: honest-but-curious and malicious. In the honest-but-curious model, adversaries follow the protocol honestly and try to learn as much as possible from observed communication by corrupted parties. In the malicious model, the parties controlled by an adversary can stop communicating or send arbitrary messages to other parties, not necessarily following the prescribed protocols.

Secure Multi-Party Computation (MPC). MPC allows a set of parties $\mathcal{P} = \{P_1, ..., P_N\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [30]. Our SSLE scheme is based on MPC.

We borrow the standard definitions of $VIEW$ and t-Privacy from [1].

We instantiate our SSLE scheme using the following underlying protocols:

1. The VSS-based MPC protocols [9,12,17,18,23,24], in which secrets are shared between the parties using Shamir's secret sharing scheme:
   - Protocols for adding shares, subtracting, and multiplying by a scalar: $[x] + [y]$, $[x] - [y]$, $[\alpha \cdot x]$,

   - RndFld to generate a share of a random field element in $\mathbb{Z}_p$,
   - RndBit to generate a share of a random bit,
   - Mul to compute $[x \cdot y]$ given $[x]$ and $[y]$.
2. Garbled circuit based MPC [29] on boolean circuits, where each party can privately input her input to a computing circuit.

2.2 Single Secret Leader Election

We consider the following problem. Given a set of N parties. The parties do some interactive pre-computation. Then, each party can run a local function that takes the transcript as input to determine whether it is the leader or not. The leader can show a proof that it is the leader.

Game-Based Formulation of the SSLE Problem. Our syntax and security properties of SSLE are based on that of [4], with a slight difference that we do not have an external source of randomness (random beacon) and we allow multiple rounds of communication between the parties during the election, whereas the definition of SSLE in [4] allows a single round of communication.

Informally, we capture the following security properties:

1. **Uniqueness** – an adversary wins this experiment if in at least one election in a series of consecutive elections there is more than one verifiable leader.
2. **Unpredictability** – the adversary asks for a challenge election after a series of elections. The challenger does not send to the adversary the outcome of this election. The adversary has to guess the leader in this challenge election. If some honest party is the leader, the adversarial chances to correctly guess the leader should not be significantly greater than pure guessing.
3. **Fairness** – the adversary asks for a challenge election after a series of elections. The probability of winning this challenge election by one of the corrupted parties should not be significantly greater than c/n, where c is the number of corrupted parties, and n is the number of parties registered for the challenge election.

Due to page limits, we postpone the formal game-based definition to the full version of this paper [3].

Simulation-Based Definition of the SSLE Problem. We now formulate the SSLE problem as an ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, which is presented in Fig. 1. In the description of the ideal functionality, we denote election id as *eid*, and registration numbers as C_i. We then show that the simulation-based definition implies the game-based one.

Our modeling of the ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ for N parties with an adversary statically corrupting up to t of them is influenced by the corresponding game-based definition, which defines the registration and verification algorithms that surround the election itself. We follow the same approach and define messages in the ideal functionality for registration, election, and their verification.

In $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, the parties send messages to the ideal functionality that correspond to a specific stage of the election. First, the parties register for an election with

$\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ for a set of parties $\mathcal{P} = \{P_1, \ldots, P_N\}$, c of which are corrupted by an adversary, consists of the following steps:

- Upon receiving a message $(eid, \mathsf{register}, C)$ from P_i, check if $(eid, P_i, \cdot)$ or $(eid, elected, \cdot, \cdot)$ is stored. If so, ignore the message. Otherwise, store (eid, P_i, C). When storing tuples, we write P_i to denote the party's unique identifier. Send $(eid, registered, P_i)$ to all parties and the environment.
- Upon receiving a message $(eid, \mathsf{regVerify})$ from P_i, reply 0 if there exist two stored tuples (eid, P_j, C_j) and (eid, P_k, C_k) such that $j \neq k$ and $C_j = C_k$. Otherwise, reply 1.
- Upon receiving a message (eid, elect) from P_i, check if there are at least ℓ registered parties that have corresponding stored tuples $(eid, \cdot, \cdot)$. If not, ignore the message, otherwise proceed. Check if $(eid, elected, P_u, C_u)$ is stored. If not, pick one of the stored tuples $(eid, \cdot, \cdot)$ uniformly at random as (eid, P_u, C_u), append it as $(eid, elected, P_u, C_u)$, and send $(eid, elected, C_u)$ to the environment. Send $(eid, elected, C_u)$ to P_i.
- Upon receiving a message $(eid, \mathsf{verify}, P_j, C)$ from P_i, check if $(eid, elected, P_u, C_u)$ is stored. If such a tuple exists, reply 1 if $P_u = P_j$ and $C_u = C$. In all other cases, reply 0.

Fig. 1. Ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$.

id eid via sending $\mathsf{register}$ messages containing the registration number C. They receive notifications from the ideal functionality for every registered party. To verify registration, the parties send messages $\mathsf{regVerify}$ to the ideal functionality, which outputs 1 if all registered numbers are distinct, otherwise it outputs 0 and the execution of $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ stops. If $\mathsf{regVerify}$ returned 1, the parties participate in the election by sending messages elect to $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, which returns one of the registered numbers as the elected number. Finally, the parties can verify whether some party P_i is the elected leader by sending a message verify with the identifier for P_i and the elected number.

Next, we discuss some of the design choices that we made in $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$:

1. With the explicit inputs associated to parties, the definition naturally captures the adversarial ability to register multiple parties using the same private material and thereby break the uniqueness property.
2. The result of the election is returned to the parties as one of the numbers, used for the registration. In this way we model the information leakage, which suggests an efficient way of running multiple elections by the same parties. To run a subsequent election, the leader has to simply re-register, while other parties can keep their previously registered numbers.

Intuitively, the security properties from the game-based definitions are captured in the ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ as follows:

1. **Uniqueness** – provided by answering $\mathsf{regVerify}$ messages, which excludes the case that two parties register the same number, and elect messages are answered with exactly one number.
2. **Unpredictability**– provided by answering elect messages with one of n registered numbers, which are known only to the respective parties. In the beginning, party P_i sends her input C_i only to the ideal functionality and never

discloses C_i to other parties until the election is finished. P_i discloses her registered number only when P_i is the elected leader.

3. **Fairness** – provided by answering elect messages by *uniformly at random* selecting one of n registered distinct numbers as the elected value.

We formally prove that the ideal functionality implies the game-based definitions by showing the non-existence of a simulator given any of the game-based attackers.

Proposition 1. *The ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ implies the game-based definitions for uniqueness, unpredictability, and fairness.*

Due to space limitations, we refer to the full-version [3] for the proof of Proposition 1 and subsequent theorems. Only the security analysis can be found in the appendix.

In this work, we only consider SSLE schemes with *expiring registration*. In such schemes, in a single SSLE instance elections are run sequentially and the eventual leader has to re-register for subsequent elections. In the remainder of the paper we will only consider the modified ideal functionality that ensures sequentiality. To this end, the ideal functionality keeps track of the current election id eid^*. As soon as it receives a message with $eid' \neq eid^*$, it stops responding to any further messages with eid^* and updates the current election id to eid'. In contrast to the real world, in the ideal world non-leaders have to register for subsequent elections explicitly using the same registration number C.

3 (Non-secret) Single Leader Election Constructions

In this section, we start by discussing how naive solutions to the problem of SSLE fail in keeping the leader secret. We then gradually introduce the basis for our final SSLE protocol. Note that, while the constructions in this section do not yet meet our requirements and are considered non-secret, they will form the basis of the protocol presented in Sect. 4.

Oblivious Select. We begin by defining a *two-party Oblivious Select* ($\mathsf{OSelect}$) protocol, whose goal is to secretly select one out of two commitments. Once the commitment is selected, the parties can open the selected commitment. Let PSwap be an algorithm that on input commitments C_0 and C_1 computes $(C_i' = \mathsf{Com}(C_i, r_i))_{i\in\{0,1\}}$ and outputs (C_b', C_{1-b}') for a random bit b. Let $\mathsf{PSelect}$ be an algorithm that on input commitments C_0 and C_1 outputs $C' = \mathsf{Com}(C_b, r)$. It is easy to see that if the commitment scheme is hiding, then an adversary cannot find the value of b significantly better than pure guessing.

We now describe $\mathsf{OSelect}$ between Alice and Bob. The protocol consists of the select and the opening phases. In the select phase, Alice publishes her commitment C_A and Bob C_B, then Alice performs PSwap on (C_A, C_B) and sends the result (C_0, C_1) to Bob; Bob now performs $\mathsf{PSelect}$ on those values and outputs C'. In the opening phase, the two parties reveal their randomness so that the complete transcript of computing C' could be reconstructed by anyone.

The protocol can be naturally extended to N parties, where N is a power of two; let us call the resulting protocol $\mathsf{OSelect}_N$. It consists of $(\log N)$ rounds; in the first round $N/2$ pairs of parties are formed that run $\mathsf{OSelect}$, thereby reducing two commitments into one. In the following round, $N/4$ pairs of parties are formed, etc., until there is a single commitment left. We will use the logical tree-like structure used in $\mathsf{OSelect}_N$ as the basis for our final SSLE construction.

Leader Election Based on Oblivious Select. We define $\mathsf{LeaderElection}$, our intermediate non-secret protocol, which essentially uses $\mathsf{OSelect}_N$ in a black-box manner. In the selection phase, each user U_i initially holds a distinct number m_i and commits to it as $C_i = \mathsf{Com}(m_i; r_i)$. Then, the users run $\mathsf{OSelect}_N$. Thanks to the properties of $\mathsf{OSelect}_N$, its output $\bar{C}$ is a commitment to one of the user's inputs. If $\bar{C}$ is a commitment to m_i, then U_i is the elected user. Since there are in total $N-1$ calls to $\mathsf{OSelect}$, we achieve an amortized cost $O(1)$ per party. In the opening phase, all users broadcast their input message and randomness, so that the execution of $\mathsf{OSelect}_N$ could be verified by anyone.

Problem. The resulting protocol is still a *non-secret* leader election, as the leader does not learn the output of the protocol exclusively. Moreover, the *unpredictability* property does not hold: an adversary controlling two parties in a single instance of $\mathsf{OSelect}$ can exclude certain parties as potential leaders. Lastly, all parties are required to participate in the protocol in at least one instance of $\mathsf{OSelect}$, which makes it impossible to tolerate a single faulty party. In the next section, we will address these problems and present our secure SSLE protocol.

Upgrading to Secret Leader. We now modify $\mathsf{LeaderElection}$ by adding an intermediate representation layer in order to let the secret leader actually check whether she is the elected leader. Here, we make use of a distributed key generation and threshold decryption. The resulting secret leader election protocol does not satisfy all our requirements to SSLE but serves as an intermediate point towards our final construction in Sect. 4.

Distributed Key Generation (DKG) [23] allows several parties to agree on a joint secret key. The corresponding public key is computed and published jointly by the honest majority of the parties. In a t-out-of-N DKG protocol [17], the secret key is shared according to Shamir's secret sharing scheme. The protocol can be efficiently simulated against passive and active adversaries, which can corrupt up to t parties. In *threshold cryptography*, parties jointly generate a group public key to encrypt messages and a qualified subset of parties can collaboratively decrypt ciphertexts encrypted using that key. We consider Shamir's t-out-of-N threshold ElGamal-based decryption schemes, for which any coalition of t parties cannot decrypt a given ciphertext or learn any information about the plaintext, whereas any coalition of $t+1$ parties can recover it, even if the remaining $N-t-1$ parties stop communicating.

Let g be a generator of a group G of a prime order $\mathbb{Z}_p$. User U_i registers for the election by generating a registration key $k_i \in \mathbb{Z}_p$ and computing a registration token as $e_i \leftarrow (g^r, g^{k_i \cdot r})$ for some random r. The values k_i are e_i are kept private. Next, the users generate a temporary shared public key using as t-out-of-N DKG protocol, $pk_{\mathcal{G}} = g^{sk_{\mathcal{G}}}$. The corresponding group secret key, $sk_{\mathcal{G}}$, is shared between N parties, such that $t+1$ parties have to collaborate to decrypt a ciphertext C.

Instead of $\mathsf{OSelect}$, we use a new subroutine $\mathsf{OSelectD}$, which is a two-party verifiable oblivious select protocol in the discrete log setting. Unlike $\mathsf{OSelect}$, the users can publicly verify that a $\mathsf{OSelectD}$ instance was executed correctly without learning which input was selected. The input to $\mathsf{OSelectD}$ is an Elgamal encryption of two group elements $e_i := (g^r, g^{k_i \cdot r})$ under a group public key $y_{\mathcal{G}}$ for some user's registration key k_i and randomness r; these two encryptions can be represented as a tuple $(g^{r'}, (y_{\mathcal{G}})^{r'} \cdot g^r, (y_{\mathcal{G}})^{r'} \cdot g^{k_i \cdot r}) \in (G_q)^3$, for some r', and we will call such tuples *valid*. $\mathsf{OSelectD}$ relies on the discrete log variants of PSwap and $\mathsf{PSelect}$, which we call PSwapD and $\mathsf{PSelectD}$. Let PSwapD be an algorithm that on input two tuples C_0 and C_1 computes $C'_0 = (C_0)^{r_0}$, $C'_1 = (C_1)^{r_1}$ and outputs (C'_b, C'_{1-b}) for a random bit b, accompanied with appropriate NIZK proofs that computation is done correctly. Let $\mathsf{PSelectD}$ be an algorithm that on input two tuples C_0 and C_1 outputs $C' = (C_b)^r$ and appropriate NIZK proofs. These proofs are generalizations [5,11] of Schnorr signature [26] and can be efficiently instantiated in the random oracle model using the Fiat-Shamir transform [14]. It is straightforward to see that if the inputs to $\mathsf{OSelectD}$ are valid tuples w.r.t. k_i and k_j, then so is the output of $\mathsf{OSelectD}$ w.r.t. $k \in \{k_i, k_j\}$.

Expanding $\mathsf{OSelectD}$ to N users, we get $\mathsf{OSelectD_N}$. Users jointly run $\mathsf{OSelectD_N}$ and decrypt its output to obtain $\bar{C}$. There will be a unique pair $(e_i, \bar{e})$, which forms a valid DDH tuple, for which the elected leader knows an exponent; all other pairs $(e_j, \bar{e})$, where $j \neq i$, are random tuples. The leader presents the exponent as proof of leadership. She will have to re-register to get a fresh k_i before participating in another election.

Problem. While the leader can learn the outcome of the election in private, there remain several problems to address. First, an adversary can run a *duplicate key* attack [4], where she obtains multiple registration tokens that correspond to a single registration key, and thus break fairness. The mitigation measures proposed in [4] work in our setting, too. Second, a malicious adversary can use biased coins when computing $\mathsf{OSelectD}$. If both parties are under her control, she can break the obliviousness of $\mathsf{OSelectD}$ and, in turn, the unpredictability and fairness of the SSLE. Finally, even an *honest-but-curious* adversary, who controls both parties in $\mathsf{OSelectD}$ and follows the protocol, exactly knows which input has been selected, thus *breaking* unpredictability of SSLE. Since our goal is to satisfy *all* the three properties (uniqueness, unpredictability, and fairness), we will need one more modification to our current construction, which we present in the following section.

4 Our SSLE from DDH

In this section, we define our full SSLE construction; to this end we modify the secret leader election from Sect. 3 by replacing $\mathsf{OSelect}$ with its MPC variant, $\mathsf{OSelectM}$. Thereby we ensure that no adversary in our model can learn

```
$\mathsf{OSelectM}([C_A], [C_B])$
$[b] \leftarrow \mathsf{RandBit}()$
 do    // run in parallel
     $[b \cdot C_A] \leftarrow \mathsf{Mul}([b], [C_A])$
     $[(1-b) \cdot C_B] \leftarrow \mathsf{Mul}([1-b], [C_B])$
$[C'] \leftarrow [b \cdot C_A] + [(1-b) \cdot C_B]$
output $[C']$
```

Fig. 2. $\mathsf{OSelectM}$: Oblivious Select in the MPC setting.

$$\begin{tikzcd}
{[C_1]} \arrow[d] & {[C_2]} \arrow[dl] & {[C_3]} \arrow[dd] & {[C_4]} \arrow[ddl] \\
{[C_1'] \leftarrow \mathsf{OSelectM}([C_1],[C_2])} \arrow[ddr] & & & \\
& & {[C_3'] \leftarrow \mathsf{OSelectM}([C_3],[C_4])} \arrow[dl] & \\
& {[\bar{C}] \leftarrow \mathsf{OSelectM}([C_1'],[C_3'])} & &
\end{tikzcd}$$

Fig. 3. $\mathsf{OSelectM_N}$: Extension $\mathsf{OSelectM}$ to N inputs. Example for $N = 4$.

the outcome of a $\mathsf{OSelectM}$ protocol instance. The extension of $\mathsf{OSelectM}$ to N inputs, which we call $\mathsf{OSelectM_N}$, retains the (binary) tree layout of inputs and outputs. Each $\mathsf{OSelectM}$ instance is now executed by all parties simultaneously. This modification incurs additional communication costs compared to the previous (insecure) version of our SSLE construction. Fortunately, the number of communication rounds needed for a leader election remains $O(\log N)$, as $\mathsf{OSelectM}$ instances on the same level in the tree can run in parallel.

$\mathsf{OSelectM}$ is an oblivious select protocol in the MPC setting, which can be completed as long as at least $t+1$ parties remain online and honestly execute the protocol. It takes two secret shares $[C_A], [C_B]$ as input and outputs a new secret share $[C']$ such that the output secret C' is either C_A or C_B with equal probability, depending on the selection bit b. The description of $\mathsf{OSelectM}$ protocol is shown in Fig. 2. $\mathsf{OSelectM}$ extension to N inputs, called $\mathsf{OSelectM_N}$, follows a binary-tree structure of inputs and outputs to $\mathsf{OSelectM}$; see Fig. 3.

To prevent duplicate key attacks, we incorporate into our SSLE scheme a technique used in [4]. The technique works as follow. The registration key k_i is now used to produce a secret part k_{iL} and a public fingerprint k_{iR} using a cryptographic hash function H, where $(k_{iL}, k_{iR}) \leftarrow H(k_i)$. Before the election starts, each user verifies that there are no duplicate fingerprints in the public state st. The security properties of the hash function ensure that chances for an adversary to succeed in a duplicate key attack are negligible.

The election proceeds as follows. For each $i \in \{1, \ldots, N\}$, the parties jointly generate $[C_i]$, a MPC version of the secret part k_{iL} of the registration key k_i, which is $[k_{iL}]$. In the MPC setting we do not need to additionally hide the key using Elgamal encryption, since secret sharing already hides the results of the computation.

The parties then proceed with $\mathsf{OSelectM_N}$ and obtain $[\bar{C}]$, which is a secret share of one of the secret inputs to $\mathsf{OSelectM_N}$. The parties jointly reconstruct two group elements $(\bar{e}_1, \bar{e}_2)$ from $[\bar{C}]$, which turn out to be a randomization of the secret part of a party participated in the election, which we denote $\bar{k}_L$. If P_i is the elected leader, the following equation will hold $\bar{k}_L = k_{iL}$, i.e. each party learns the secret key k_{iL} of the leader, but does not know which one. The leader P_i sends the registration key k_i as a proof of leadership. To verify a proof π, one recomputes the secret part π_L of the registration key and its fingerprint π_R and checks that the computed fingerprint matches the one stored as st_i, and that the equation $\pi_L = \bar{k}_L$ holds.

In the malicious adversary model, we can use standard techniques [10,18] to protect the underlying MPC primitives used in the scheme against active adversaries.

We now formally define our fully-fledged SSLE construction.

Construction 1 (Single secret leader election (SSLE). *Our* (N, N, t)-*SSLE scheme is a tuple of PPT algorithms* $\mathsf{SSLE} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{RegisterVerify}, \mathsf{Elect}, \mathsf{Verify})$ *that use a group* G *of a prime order* p. *Let* g *be a generator of* G, *let* H *be a function that maps* $\{0,1\}^\lambda$ *to* $\mathbb{Z}_p \times \{0,1\}^{r(\lambda)}$. *The description of the algorithms is shown in Fig. 4.*

```
Setup(1^λ, t, N)
1 :  $p, g \leftarrow \mathsf{FindParam}(1^\lambda)$
2 :  for $i \in 1..N$
3 :      $st_i \leftarrow \bot$
4 :  return $p, g, st_1, \ldots, st_N$
```

```
Register(i)
1 :  $k_i \leftarrow \mathbb{Z}_p$
2 :  $k_{iL}, k_{iR} \leftarrow H(k_i)$
3 :  $st_i \leftarrow k_{iR}$
4 :  $[k_{iL}] \leftarrow \mathsf{Share}(k_{iL})$
5 :  return $k_i$
```

```
RegisterVerify(i, k_i)
1 :  for $j_1 \in 1..(N-1)$
2 :    for $j_2 \in (j_1+1)..N$
3 :      if $st_{j_1} = st_{j_2} \neq \bot$
4 :        return 0
5 :  $k_{iL}, k_{iR} \leftarrow H(k_i)$
6 :  if $k_{iR} \neq st_i$
7 :    return 0
8 :  return 1
```

```
Verify(i, π)
1 :  $\pi_L, \pi_R \leftarrow H(\pi)$
2 :  if $\pi_L = \bar{k}_L$ and $\pi_R = st_i$
3 :      return 1
4 :  return 0
```

```
Elect(i, k_i)
1 :  for $i \in 1..N$
2 :      $[C_i] := [k_{iL}]$
3 :  $[\bar{C}] \leftarrow \mathsf{OSelectM_N}([C_1],$
4 :    $\ldots, [C_N])$
5 :  $\bar{k}_L \leftarrow \mathsf{Rec}([\bar{C}])$
6 :  if $\bar{k}_L \neq k_{iL}$
7 :      return $\bot$
8 :  return $\pi := k_i$
```

Fig. 4. Single Secret Leader Election construction SSLE instantiated with $\mathsf{OSelectM_N}$.

Theorem 1. *Assuming the underlying MPC primitives are secure in the honest-but-curious adversary model, H is a random oracle, then Construction 1 implements functionality* $\mathcal{F}_{\mathsf{SSLE}}$.

5 Our SSLE Based on Garbled Circuits

In this section, we present our SSLE protocol, instantiated in the MPC framework by Wang et al. [29], which can tolerate up to $N-1$ corrupted parties.

We use the MPC protocol [29] to instantiate our SSLE in a black-box manner. The SSLE construction shown in Fig. 4 needs to be updated to account for the technical details specific to the MPC part in the Elect algorithm.

To implement the Oblivious Select, we use a part of the input as selection bits. Each party contributes to these bits, via bitwise-xor; the selection bits are therefore secret-shared. The modified version of the Elect algorithm and a pseudocode of OSelectM instantiated in the framework [29] are shown in the full version [3].

Construction 2 (Single secret leader election (SSLE)). *Our SSLE scheme is a tuple of PPT algorithms SSLE = (Setup, Register, RegisterVerify, Elect, Verify). Let H be a function that maps $\{0,1\}^{\lambda}$ to $\{0,1\}^{l(\lambda)} \times \{0,1\}^{r(\lambda)}$. The description of the algorithms is shown in fig.4, and Elect is appropriately modified.*

Theorem 2. *Assuming the underlying MPC primitives are secure in the malicious adversary model, H is a random oracle, then Construction 2 implements functionality $\mathcal{F}_{\mathsf{SSLE}}$.*

6 Evaluation

6.1 Experimental Setup

We evaluate our SSLE framework, we implemented Constructions 1 and 2 and ran two kind of tests: in a local setting (LAN) and in a global setting (WAN). In the LAN setting, we used machines located in the same Amazon EC2 region. In the WAN setting, we used machines located in four different regions (Europe, North America, South America, and Asia). If not specified otherwise, each machine is a t2.large instance with 2 cores Intel Xeon E5-2686v4 2.3 GHz, 8 Gb of RAM, and installed Ubuntu 20.04. In some regions t2.large instances are not available; instead we used t3.large instances with 2 cores Xeon Platinum 8175 2.5 GHz, 8 Gb of RAM.

In our experiments, we evaluate a complete OSelect tree in our SSLE framework, that is the number of users being a power of two, starting from 8 parties, and each party holding one unit of stake. For each experiment we take average of 10 runs, except that for lengthy experiments with a running time more than 1 min we perform a single run. Next, we present implementation details and the evaluation results individually for each construction.

6.2 Construction 1 (Sect. 4)

Implementation Details. We implemented our Construction 1 in C++ in the honest-but-curious and malicious adversary models. We implemented the underlying MPC primitives for secret sharing, adding shares, substracting, multiplying by a scalar: $[x] + [y]$, $[x] - [y]$, $[\alpha \cdot x]$, protocols RndFld, RndBit, Mul [9,12,17,18,23,24]. In the malicious adversary model, these primitives are accompanied with verifiable secret sharing (VSS). We set the threshold $t = N/2 - 1$ in all experiments. Our implementation uses the Relic toolkit [2] for operations on elliptic curves in groups of a prime order of 256 bits, the Boost and OpenSSL libraries for secure communication.

Table 2. Experimental results for Construction 1 in the honest-but-curious and malicious adversary models (left), and for Construction 2 in the malicious adversary model (right).

N	t	Algorithm	HbC time, sec.	Mal. time, sec.
8	3	Register	<0.01	0.11
		RegisterVerify	<0.01	<0.01
		Elect	0.1	3.56
		Verify	<0.01	<0.01
16	7	Register	0.01	0.56
		RegisterVerify	<0.01	<0.01
		Elect	0.34	28.1
		Verify	<0.01	<0.01
32	15	Register	0.02	3.83
		RegisterVerify	<0.01	<0.01
		Elect	1.45	356.6
		Verify	<0.01	<0.01
64	31	Register	0.08	n.a.
		RegisterVerify	<0.01	
		Elect	7.63	
		Verify	<0.01	
128	63	Register	0.21	n.a.
		RegisterVerify	<0.01	
		Elect	54.4	
		Verify	<0.01	

N	$l(\lambda)$	LAN time, sec.	WAN time, sec.
8	48	2.73	23.42
	64	2.76	24.03
	80	2.80	24.27
16	48	4.28	38.95
	64	4.50	39.92
	80	4.86	40.61
32	48	8.25	73.34
	64	8.35	75.93
	80	8.80	77.81
64	48	17.64	145.87
	64	18.62	153.34
	80	23.90	150.67
128	48	64.33	300.77
	64	74.54	326.09
	80	83.54	317.46

Experimental Results. We performed LAN tests for up to 128 parties in the honest-but-curious adversary model, and up to 32 parties in the malicious model. Timings are shown in Table 2.

Analysis. The experimental results show that up to 128 parties can complete Elect protocol in under a minute. The running time grows rapidly as the number of parties increases. This is due to expensive public key operations for generating and reconstructing Shamir's secret shares. The explosion of running time is more visible in the malicious adversary model. In order to protect against such adversaries, we have to use verifiable secret sharing, which requires $O(N^2)$ public key operations in the textbook implementation. While Elect is the most heavy algorithm, the rest of the SSLE protocol is essentially for free. We conclude that Construction 1 offers a practical t-robust solution to the SSLE problem for a small number of parties (up to 32, according to our evaluation).

6.3 Construction 2 (Sect. 5)

Implementation Details. We implemented and evaluated Oblivious Select part of the Elect algorithm, as it is the most heavy part of the SSLE protocol (see experimental results for Construction 1 in the honest-but-curious adversary model in Sect. 6.2). Our implementation fully relies on the implementation of the MPC framework by Wang et at. [29], which is available as [28]. We can trade-off security for efficiency by controlling how many bits each party inputs to Oblivious Select.

Experimental Results. In the MPC framework, the evaluator of the garbled global circuit requires more RAM than any other party. Therefore, we set up

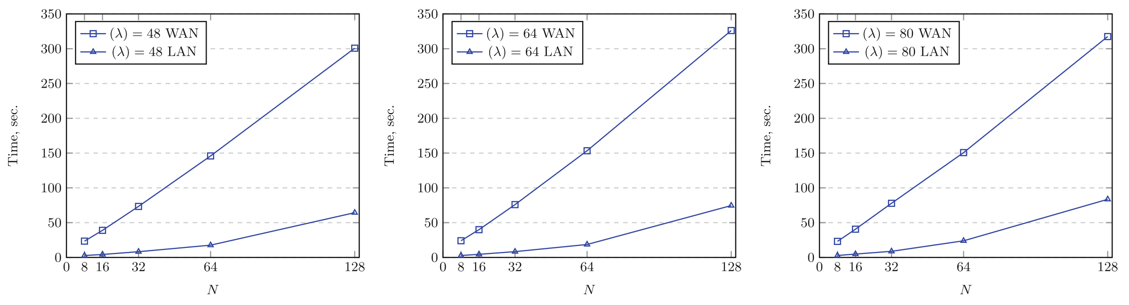

Fig. 5. Comparison of timings for Oblivious Select in Construction 2 in the LAN and WAN settings.

one machine as a m5a.4xlarge instance with 16 cores and 64G of RAM, while the rest of machines remain t2.large or t3.large instances. We run experiments for each N up to 128. For the trade-off, we choose the length of user inputs to Oblivious Select, $l(\lambda)$, as 48, 64, and 80 bits. Additionally, each party provides 8 bits of selection bits, which satisfies the constraint that it should be at least as big as $\log(N)$ in all test cases. Timings the LAN and WAN settings are shown in Table 2 and in Fig. 5.

Analysis. The experimental results show that the running time of Oblivious Select algorithm (and in turn, Elect) grows almost linearly as the number of parties gets increased. As we ran only 1 iteration for long test cases, we can see some unexpected fluctuations in the running time, which we think are caused by fluctuations in the network and normally should be eliminated after averaging multiple iterations.

The LAN and WAN settings have identical computational and communication cost, as they only differ in the location of machines. We suspect that higher latency between machines in the WAN settings accounts for the increased running time. In the LAN setting, 128 parties can compute a leader in under 1.5 min, where as in the WAN setting, this number approaches 7 min.

7 Practical Considerations

There are several constraints in Constructions 1 and 2 that affect its practicality. First, the definition of SSLE says that the probability for a party being elected should be equal among all participants. In practice, the stakeholders may have different stakes, and the probability for a party to be elected should be proportional to her stake. A straightforward solution to this constraint would be to adapt our SSLE construction to work with stake units and let each party control several units. If implemented naively, this approach results in a linear blow-up in computation and required storage (in the number of stake units). In the following, we will show an efficient technique to extend Construction 1 to support arbitrary (non-uniform) probability distributions in the election.

Second, we assumed the number of parties to be a power of two, in order to construct a complete binary tree in Oblivious Select. However, if the number of parties is arbitrary, the tree structure will likely unbalance the tree leaves, as some inputs will not be matched on the first level with other inputs. Therefore, such inputs would proceed to the next round without competition, i.e., with the probability of 1, whereas input C_i in a binary tree will proceed with the probability of 1/2. We will show that the technique from the previous point addresses this concern, too.

7.1 Non-uniform Distributions

$$\begin{tikzcd}
{[C_1]} \arrow[dr] & & {[C_2]} \arrow[dl] & & {[C_2]} \arrow[dr] & & {[C_2]} \arrow[dl] \\
& {[\ldots]} \arrow[drr] & & & & {[\ldots]} \arrow[dll] & \\
& & & {[\bar{C}]} \arrow[d, Rightarrow] & & & \\
& & {[C_1]} \arrow[dr] & {} & {[C_2]} \arrow[dl] & & \\
& & & {[\ldots]} \arrow[dr] & & {[C_2]} \arrow[dl] & \\
& & & & {[\bar{C}]} & &
\end{tikzcd}$$

Fig. 6. Tree optimization technique. Example $\mathsf{OSelectM_N}$ for P_1 and P_2 with stakes $(1, 3)$.

We observe that it is possible to unbalance almost-for-free the probability of being selected (among two parties) if the sum of the weights is a *power of two*. To illustrate this idea, assume that the weights are $(1, 3)$, i.e., the probabilities for two parties being selected are determined by the ratio 1:3. We can construct a tree-structure with the probabilities 1/4 and 3/4, as shown in Fig. 6.

Basically, we introduce a special case for $\mathsf{OSelectM}$ when handling shares of the same secret for free, $\mathsf{OSelectM}(\mathsf{Share}C, [C]) \rightarrow [C]$. The resulting tree can be optimized significantly by dropping the nodes with the same inputs.

Using this technique, we can handle weights of the form $(w, 2^L - w)$ with a logarithmic overhead, for some $L \geq 1$ and $1 \leq w < 2^L$. However, we cannot naturally handle arbitrary weight ratios. For example, weights such as $(1, 2)$ are problematic. Nevertheless, we can approximate the probabilities in the election according to any weights (a, b) by having a tree of sufficient depth.

Arbitrary N and Stakes. Let N be the number of parties participating in the election with their stakes $(s_1, ..., s_N)$, and let $S = \sum_{i=1}^{N} s_i$ be the sum of parties' stakes. The multi-registration solution may lead to $O(S)$ complexity of the election algorithm. We extend our technique to an arbitrary number of users.

We start with a similar idea: each party has a sequence of stake units on the first level in a $\mathsf{OSelectM_N}$ tree. If $N \ll S$, there will be many pairs of inputs that represent the same party. We observe that in this case, there is no need to run $\mathsf{OSelectM}$ on such inputs. Instead, we can pick any input and advance it to the next level in the tree. The worst case complexity (the number of $\mathsf{OSelectM}$ instances) of this technique is $O(N \log S)$, since each party P_i's inputs will be matched in a tree of depth $O(\log S)$ at most two times, against P_{i-1} and P_{i+1}. With a tree of depth L we can get the absolute precision up to $2^{-L} \cdot S$.

Acknowledgements. We thank anonymous reviewers for their helpful reviews. This work has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 610150-imPACT, was supported by the German Federal Ministry of Education and

Research (BMBF) through funding for CISPA and the CISPA-Stanford Center for Cybersecurity (FKZ: 16KIS0762), and by the Luxembourg National Research Fund (FNR) under the CORE Junior project FP2 (C20/IS/14698166/FP2/Mueller).

A Security Analysis

Lemma 1. *Let $[C_A]$ and $[C_B]$ be the inputs to* $\mathsf{OSelectM}$ *protocol, and let $[C']$ be the output. Then, assuming the underlying secret sharing scheme is linearly homomorphic and the primitives for multiplication secret shares and generating a random shared bit are secure, it holds that $C' \in \{C_A, C_B\}$.*

Proof. The underlying $\mathsf{RandBit}$ primitive produces shares of a random bit $[b]$. By homomorphic properties of the secret sharing scheme and security of the multiplication primitive, it follows that, if $b = 0$, C' evaluates to C_A, otherwise, if $b = 1$, C' evaluates to C_B.

Lemma 2. *Let $[C_1], ..., [C_N]$ be the inputs to* $\mathsf{OSelectM_N}$ *protocol, and let $[\bar{C}]$ be the output. Then, it holds that $\bar{C} \in \{C_1, ..., C_N\}$.*

Proof. It follows from Lemma 1 and the binary tree structure of $\mathsf{OSelectM}$ instances in $\mathsf{OSelectM_N}$.

Lemma 3. *Assuming secret sharing is secure, algorithm* $\mathsf{Register}$ *in Construction 1 called by some party, securely implements sending a* ($\mathsf{register}$) *message in the ideal model.*

Proof. The party uses the output value from $\mathsf{Register}$ as input to the ($\mathsf{register}$) message in the ideal model. The proof follows from simulatability of the secret sharing scheme.

Lemma 4. *Assuming H is a random oracle, algorithm* $\mathsf{RegisterVerify}$ *in Construction 1 securely implements sending a* ($\mathsf{regVerify}$) *message in the ideal model.*

Proof. By the properties of the random oracle, we have that the probability that $C_i \neq C_j$ in the ideal model and $k_{iR} = k_{jR}$ is $1/2^\lambda$, which is negligible in λ.

Lemma 5. *Algorithm* Elect *in Construction 1 securely implements sending a* (elect) *message in the ideal model.*

Proof. We construct a simulator $\mathcal{S}$ for an ideal adversary $\mathcal{A}$. $\mathcal{S}$ recovers the adversarial input from party P_i by reconstructing it from the shares available to the simulator ($\mathcal{S}$ controls enough honest parties to reconstruct any shared secret).

$\mathcal{S}$ sends all inputs from honest parties and the recovered adversarial inputs and receives $C_{U_1^N}$ from the ideal functionality as the result of the election. It is the same for all parties, including those controlled by the adversaries, so the simulator forwards this value to $\mathcal{A}$. In order to let the adversary believe it interacts with the real protocol, the simulator has to produce a transcript of the $\mathsf{OSelectM_N}$ protocol that will result in a specific value U_N to be chosen and output. To do that, the simulator fixes the shares of the honest parties for random bits $[b]$ in $\mathsf{OSelectM}$ instances so that the reconstruction would output the specific fixed b, that will result $\mathsf{OSelectM_N}$ to select precisely the U_N-th element of the sequence $(st_1, \ldots, st_N)$. The underlying secret sharing scheme allows to simulate the transcripts for the honest parties that share a simulator-chosen secret.

In the real protocol, it is possible that for some $i \neq j$, $k_{iL} = k_{jL}$, while $k_{iR} \neq k_{jR}$ and so the parties would pass the registration. However, this only happens with a low probability that we can control.

Lemma 6. *Assuming H is a random oracle, Construction 1 produces a unique leader with the probability at least* $1 - e^{-\frac{N(N-1)}{2p}}$.

Proof. The probability that there exist two parties P_i and P_j such that $k_{iL} = k_{jL}$ and $k_{iR} \neq k_{jR}$ can be estimated by the birthday paradox. Specifically, this probability is bounded by $e^{-\frac{N(N-1)}{2p}}$.

Lemma 7. *Algorithm* Verify *in Construction 1 securely implements sending a* (verify) *message in the ideal model.*

Proof. It follows by a similar argument as in the proof of Lemma 4.

Proof (Proof of Theorem 1). Since we only consider sequential execution, we need to show that the adversarial view in the real and the ideal worlds is indistinguishable for one instance of the protocol, and the security of the whole protocol will follow by Canetti's composition theorem [6]. To this end, we construct a simulator for a real-world adversary as follows.

- For the registration, the real-world adversary and honest users use a call to the random oracle H for some (random) input and then share a string. Sharing algorithm can be simulated by a secure secret sharing scheme. Moreover, a t-private secret sharing scheme for $t < N$ allows $\mathcal{S}$ to reconstruct the input used by the corrupted user. Hence, all the numbers shared by the corrupted and honest users during registration are known to $\mathcal{S}$.
- To simulate the verification of registration, $\mathcal{S}$ verifies that there is no duplicate numbers recorded during registration. If this is the case, it outputs 1, otherwise 0.
- To simulate the election, $\mathcal{S}$ first consults the ideal functionality to elect the leader and then we use Lemma 5 to simulate the corresponding transcript.
- To simulate the verify algorithm, $\mathcal{S}$ compares the elected number with the number registered by the user (honest or malicious) and outputs 1 if the numbers are equal, otherwise it outputs 0.

We argue that any PPT environment $\mathcal{Z}$ cannot distinguish between the ideal world and the real world significantly better than negligible probability via a series or games.

Due to space limits, we complete the proof in the full version.

Proof (Proof of Theorem 2.). Since both our constructions Construction 1 and Construction 2 rely on secure MPC primitives and differ only in specifics of the used MPC frameworks, we simply follow the steps in the proof of Theorem 1 to prove the theorem.

References

1. Aliasgari, M., Blanton, M., Zhang, Y., Steele, A.: Secure computation on floating point numbers. In: NDSS (2013)
2. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an Efficient LIbrary for Cryptography (2014). https://github.com/relic-toolkit/relic
3. Backes, M., Berrang, P., Hanzlik, L., Pryvalov, I.: A framework for constructing single secret leader election from MPC (full version). eprint 2022/1040 (2022)
4. Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, pp. 12–24 (2020)
5. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical report/Department of Computer Science, ETH Zürich 260 (1997)
6. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
7. Catalano, D., Fiore, D., Giunta, E.: Efficient and universally composable single secret leader election from pairings. IACR Cryptol. ePrint Arch. 2021/344 (2021)
8. Catalano, D., Fiore, D., Giunta, E.: Adaptively secure single secret leader election from DDH. In: ACM PODC 2022, pp. 430–439 (2022)
9. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_6
10. Cramer, R., Damgård, I., Maurer, U.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_22
11. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_19
12. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_15
13. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
14. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
15. França, B., Wissfeld, M., Berrang, P., von Styp-Rekowsky, P., Trinkler, R.: Albatross: an optimistic consensus algorithm. arXiv preprint arXiv:1903.01589 (2019)

16. Ganesh, C., Orlandi, C., Tschudi, D.: Proof-of-stake protocols for privacy-aware blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 690–719. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_23
17. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)
18. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC 1998, pp. 101–111 (1998)
19. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), pp. 51–68 (2017)
20. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
21. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
22. O'Dwyer, K.J., Malone, D.: Bitcoin mining and its energy footprint. In: InISSC 2014/CIICT 2014, pp. 280–285. IET (2014)
23. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
24. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (STOC), pp. 73–85 (1989)
25. Single-Leader Election (SSLE) (2019). https://github.com/protocol/research-grants/blob/master/RFPs/rfp-006-SSLE.md
26. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
27. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
28. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: efficient MultiParty computation toolkit (2016). https://github.com/emp-toolkit
29. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: CCS 2017, pp. 39–56 (2017)
30. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (SFCS), pp. 160–164 (1982)