Multiple Secret Leaders

Authors here

July 21, 2023

# 1 Preliminaries

## 1.1 Threshold FHE

[JRS17, BGG$^+$18] defines threshold FHE encryption. For the sake of completeness, we will define it here.

**Definition 1.1** (TFHE [JRS17])**.** Let $P = \{ P_1, ..., P_N \}$ be a set of $N$ parties and $\mathbb{S}$ be a class of access structures on $P$. A TFHE scheme for $\mathbb{S}$ is a tuple of PPT algorithms

$$(\texttt{TFHE.Setup}, \texttt{TFHE.Encrypt}, \texttt{TFHE.Eval}, \texttt{TFHE.PartDec}, \texttt{TFHE.FinDec})$$

such that the following specifications are met

- $(pk, sk_1, ..., sk_N) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^s, \mathbb{A})$: Takes as input a security parameter $\lambda$, a depth bound on the circuit, and a structure $\mathbb{A} \in \mathbb{S}$. Outputs a public key $pk$ and a secret key $sk_i$ for each party $P_i$.
- $\texttt{ct} \leftarrow \texttt{TFHE.Encrypt}(pk, \mu)$: Takes as input a public key and a message $\mu \in \{ 0, 1 \}$ and outputs a ciphertext $\texttt{ct}$.
- $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(C, \texttt{ct}_1, ..., \texttt{ct}_k)$: Takes as input a circuit $C$ of depth at most $d$ and $k$ ciphertexts $\texttt{ct}_1, ..., \texttt{ct}_k$. Outputs a ciphertext $\hat{\texttt{ct}} = C(\texttt{ct}_1, ..., \texttt{ct}_k)$.
- $p_i \leftarrow \texttt{TFHE.PartDec}(\texttt{ct}, sk_i)$: Takes as input a ciphertext $\texttt{ct}$ and a secret key $sk_i$ and outputs a partial decryption $p_i$.
- $\hat{\mu} \leftarrow \texttt{TFHE.FinDec}(B)$: Takes as input a set $B = \{ p_i \}_{i \in S}$ for some $S \subseteq [N]$ and deterministically outputs a message $\hat{\mu} \in \{ 0, 1, \perp \}$.

Further, we remember the definitions of evaluation correctness, semantic security, and simulation security as outlined in, [BEHG20].

**Definition 1.2** (Evaluation Correctness [JRS17])**.** . We have that $\texttt{TFHE}$ scheme is correct if for all $\lambda$, depth bounds $d$, access structure $\mathbb{A}$, circuit $C : \{ 0, 1 \}^k \rightarrow \{ 0, 1 \}$ of depth at most $d$, $S \in \mathbb{A}$, and $\mu_i \in \{ 0, 1 \}$, we have the following. For $(pk, sk_1, ..., sk_N) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$, $\texttt{ct}_i \leftarrow \texttt{TFHE.Encrypt}(pk, \mu_i)$ for $i \in [k]$, $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(pk, C, \texttt{ct}_1, ..., \texttt{ct}_k)$,

$$\mathbf{Pr}\left[\texttt{TFHE.FinDec}(pk, \{ \texttt{TFHE.PartDec}(pk, \hat{\texttt{ct}}, sk_i) \}_{i \in S}) = C(\mu_1, ..., \mu_k)\right] = 1 - \texttt{negl}(\lambda).$$

**Definition 1.3** (Semantic Security [JRS17])**.** We have that a $\texttt{TFHE}$ scheme satisfies semantic security for for all $\lambda$, and depth bound $d$ if the following holds. There is a stateful PPT algorithm $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for any PPT adversary $\mathcal{A}$, the following experiment outputs 1 with negligible probability in $\lambda$:

1. On input $1^\lambda$ and depth $1^d$, the adversary outputs $\mathbb{A} \in \mathbb{S}$
2. The challenger runs $(pk, sk_1, ..., sk_N) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ and provides $pk$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a set $S \subseteq \{ P_1, ..., P_N \}$ such that $S \notin \mathbb{A}$.
4. The challenger provides $\{ sk_i \}_{i \in S}$ and $\texttt{TFHE.Encrypt}(pk, \mu)$ to $\mathcal{A}$ where $\mu \xleftarrow{\$} \{ 0, 1 \}$.
5. $\mathcal{A}$ outputs a guess $\mu'$. The experiment outputs 1 if $\mu' = \mu$.

**Definition 1.4** (Simulation Security [JRS17])**.** We say that a $\texttt{TFHE}$ scheme is simulation secure if for all $\lambda$, depth bound $d$, and access structure $\mathbb{A}$ if there exists a stateful PPT simulator, $\mathcal{S}$, such that for any PPT adversary $\mathcal{A}$, we have that the experiments $\texttt{Expt}_{\mathcal{A},\text{Real}}(1^\lambda, 1^d)$ and $\texttt{Expt}_{\mathcal{A},\text{Sim}}(1^\lambda, 1^d)$ are statisically close as a function of $\lambda$. The experiments are defined as follows:

- $\texttt{Expt}_{\mathcal{A},\text{Real}}(1^\lambda, 1^d)$:
  1. On input the security parameter $1^\lambda$ and depth bound $d$, the adversary outputs $\mathbb{A} \in \mathbb{S}$.
  2. Run $\texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ to obtain $(pk, sk_1, ..., sk_N)$. The adversary is given $pk$.
  3. The adversary outputs a set $S \subseteq \{ P_1, ..., P_N \}$ such that $S \notin \mathbb{A}$ together with plaintest messages $\mu_1, ..., \mu_k \in \{ 0, 1 \}$. The adversary is handed over $\{sk_i\}_{i \in S}$
  4. For each $\mu_i$, the adversary is given $\texttt{TFHE.Encrypt}(pk, \mu_i) \rightarrow \texttt{ct}_i$.
  5. The adversary issues a polynomial number of queries, $(S_i \subseteq \{ P_1, ..., P_N \}, C_i)$. for circuites $C_i : \{ 0, 1 \}^k \rightarrow \{ 0, 1 \}$. After each query the adversary receeuves for $l \in S_i$ the value
  $$\texttt{TFHE.PartDec}(\texttt{TFHE.Eval}(C_i, \texttt{ct}_1, ..., \texttt{ct}_k), sk_l) \rightarrow p_l$$
  6. $\mathcal{A}$ outputs $\texttt{out}$, the experiment's output.
- $\texttt{Expt}_{\mathcal{A},\text{Sim}}(1^\lambda, 1^d)$:
  1. On input the security parameter $1^\lambda$ and depth bound $d$, the adversary outputs $\mathbb{A} \in \mathbb{S}$.
  2. Run $\texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ to obtain $(pk, sk_1, ..., sk_N)$. The adversary is given $pk$.
  3. $\mathcal{A}$ outputs a set $S^* \subseteq \{ P_1, ..., P_N \}$ such that $S \notin \mathbb{A}$ and plaintexts $\mu_1, ..., \mu_k \in \{ 0, 1 \}$. The simulator is given $pk, \mathbb{A}, S^*$ as input and outputs $\{ sk_i \}_{i \in S^*}$ and the state $\texttt{state}$. The adversary is given $\{sk_i\}_{i \in S^*}$
  4. For each $\mu_i$, the adversay is given $\texttt{TFHE.Encrypt}(pk, \mu_i) \rightarrow \texttt{ct}_i$.
  5. $\mathcal{A}$ issues a polynomial number of queries of the form $(S_i \subseteq \{ P_1, ..., P_N \}, C_i)$
  6. for circuits $C_i : \{ 0, 1 \}^k \rightarrow \{ 0, 1 \}$. After each query, the simulator computes
  $$\texttt{Sim}_{\texttt{TFHE}}(C_i, \{ \texttt{ct}_l \}_{l=1}^k, C_i(\mu_1, ..., \mu_k), \texttt{state}) \rightarrow \{ p_l \}_{l \in S_i}$$
  and sends $\{ p_l \}_{l \in S_i}$ to the adversary.
  7. $\mathcal{A}$ outputs $\texttt{out}$, the experiment's output.

## 1.2 Non Interactive Zero-Knowledge

The following definition is taken almost verbatim from [CFG22]. A non-Interactive Zero-Knowledge (NZIK) proof system for relationship *Rel* is a tuple of PPT algorithms $(\texttt{NZIK.G}, \texttt{NZIK.P}, \texttt{NZIK.V})$ such that $\texttt{NZIK.G}$ generates a common reference string, $\texttt{NZIK.crs}$, $\texttt{NZIK.P}(\texttt{NZIK.crs}, x, w)$ given $(x, w) \in Rel$, outputs a proof $\pi$, and $\texttt{NZIK.V}(\texttt{NZIK.crs}, x, \pi)$ outputs 1 if $(x, w) \in Rel$ and 0 otherwise. A $\texttt{NZIK}$ is correct is for every $\texttt{NZIK.crs}$ and all $(x, w) \in Rel$, we have that $\texttt{NZIK.V}(\texttt{NZIK.crs}, x, \texttt{NZIK.P}(\texttt{NZIK.crs}, x, w)) = 1$ holds with probability 1. We also require that our NZIKs satisfy the notions of weak simulation extractability [Sah99] and zero-knowledge [FLS90].

Weak simulation extractability guarentees the extractability of proofs produced by the adversary that are not equal to proofs previously observed. Thus, we make each proof "unique" by implicitly adding a session ID to the statement. For more of a commentary, see section 2.7 of [CFG22]. We will not detail how to handle these session IDs.

We recall the notion of simulation security from [Sah99]:

**Definition 1.5** (NZIK simulation security)**.** We say that a proof system for relationship *Rel* is simulation secure if proof system $(\texttt{NZIK.G}, \texttt{NZIK.P}, \texttt{NZIK.V})$ is a non-interactive proof system and $\mathrm{Sim}_1, \mathrm{Sim}_2$ are PPT algorithms such that for all PPT adversaries $\mathcal{A}_1, \mathcal{A}_2$ we have that $\left|\mathbf{Pr}[\texttt{Expt}_{\mathcal{A},\mathrm{Real}}(\lambda) = 1] - \mathbf{Pr}[\texttt{Expt}_{\mathcal{A},\mathrm{Sim}}(\lambda) = 1]\right|$ is negligible in $\lambda$. The experiments are defined as follows:

- $\texttt{Expt}_{\mathcal{A},\mathrm{Real}}$:
  1. $\texttt{NZIK.crs} \leftarrow \texttt{NZIK.G}(1^\lambda)$
  2. $\mathcal{A}_1$ outputs $(x, w, \texttt{state}_1)$
  3. $\pi \leftarrow \texttt{NZIK.P}(\texttt{NZIK.crs}, x, w)$
  4. return $\mathcal{A}_2(\pi, \texttt{state}_1)$
- $\texttt{Expt}_{\mathcal{A},\mathrm{Sim}}$:
  1. $\texttt{NZIK.crs} \leftarrow \mathrm{Sim}_1(1^\lambda)$
  2. $\mathcal{A}_1$ outputs $(x, w, \texttt{state})$
  3. $\pi \leftarrow \mathrm{Sim}_2(\texttt{NZIK.crs}, x, w)$
  4. return $\mathcal{A}_2(\pi, \texttt{state})$

We also have that a NZIK has ideal functionality $\mathcal{F}_{\texttt{NZIK}}^{Rel}$ which is defined as follows:

**The $\mathcal{F}_{\texttt{NZIK}}^{Rel}$ functionality for zero-knowledge**:
Upon receiving $(\texttt{prove}, sid, x, w)$ from $P_i$, with $sid$ being used for the first time, if $(x, w) \in Rel$, broadcast $(\texttt{proof}, sid, i, \pi)$, otherwise broadcast $\perp$.

## 1.3 Data Independent Priority Queue

In this work, we will use data independent queues as studied in [Tof11, MZ14, MDPB23]. Data independent data structures are unique as their control flow and memory access do not depend on input data ([MZ14]).

**Definition 1.6** (Word RAM model [MZ14])**.** In the word RAM model, the RAM has a constant number of public and secret registers and can perform arbitrary operations on a constant number of registers in constant time.

**Definition 1.7** (Data Independent Data Structure [MZ14]). In the word RAM model, a data independent data structure is a collection of algorithms where all the algorithms uses RAM such that the RAM can only set its control flow based on registers that are public.

Data independent queues are especially useful as they allow for efficient computation within MPC and FHE as control flow is not dependent on underlying ciphertexts data. We use a data independent queue as outlined in [MDPB23] which allows for

- `PQ.Insert`: Inserts a tag and value, $(p, x)$ into `PQ` according to the tag's priority.
- `PQ.ExtractFront`: Removes and returns the $(p, y)$ with highest tag priority.
- `PQ.Front`: Returns the $(p, y)$ with highest tag priority without removing the element.

Moreover, we note that the order is stable. I.e. the first inserted among equal tagged elements has a higher priority.

### 1.4 Resevoir Sampling

Resevoir sampling is an online algorithm which allows for randomly selecting $k$ elements from a stream of $n$ elements while using $\tilde{O}(k)$ space. Algorithm R ([Vit85]) is a simple algorithm which relies on a priority queue with interface:

- $\texttt{Resevoir.Init}(k)$ initialize the resevoir sampling data structure $\mathcal{R}$ such that $\mathcal{R}.\texttt{PQ}$ is an empty priority queue.
- $\texttt{Resevoir.Insert}(\mathcal{R}, \mu_i, e_i, \texttt{coin}_i)$ where $\mu_i$ is $i$-th item, $e_i$ is independentally sampled randomness, and $\texttt{coin}_i$ is a random coin with probability $1/m$ of equaling 1.
  - If $i \leq k$, insert the item into the queue along with $e_i$ as its tag via $\mathcal{R}.\texttt{PQ.Insert}(e_i, \mu_i)$.
  - If $i > k$ and $\texttt{coin}_i = 1$, replace the smallest labeled item in the queue with the new item if the coin is 1.
  - If $i > k$ and $\texttt{coin}_i = 0$, do nothing.
  - Return $\mathcal{R}$
- $\mu_{a_1}, \mu_{a_2}, ..., \mu_{a_k} \leftarrow \texttt{Resevoir.Output}(\mathcal{R})$ where $a_1, ..., a_k$ are a uniformly random ordered susbset of $[n]$ if $e_i, \texttt{coin}_i$ are independentally sampled uniformly at random for all $i \in n$ where $n$ is the number of `Resevoir.Insert` calls.
  - Call `PQ.ExtractFront` $k$ times setting $\mu_{a_\ell}$ to the $\ell$-th call to `PQ.ExtractFront` where $\ell \in [k]$.

## 2 Ideal MSLE Functionality

We use a similar notion of ideal functionality for a multi-secret leader election from the ideal functionality of single secret leader election of [CFG22] except that we add a `register_elect` phase for each election.

**The MSLE functionality** $\mathcal{F}_{\text{MSLE}}$: Set $\mathcal{E} \leftarrow \emptyset$ to denote the set of finished elections. Fix some $k \in \mathbb{N}$ to denote the number of rounds. Upon receiving,

- $\texttt{register\_elect}(eid)$ from party $P_i$. If $eid \in \mathcal{E}$ send $\perp$ to $P_i$ and do nothing. If $S_{eid}$ is not defined, set $S_{eid} \leftarrow \{i\}$. Otherwise, set $S_{eid} \leftarrow S_{eid} \cup \{i\}$ and store $S_{eid}$.
- $\texttt{elect}(eid)$ from all honest participants.
  If $|S_{eid}| \geq k$, randomly sample $W^{eid} \subseteq S_{eid}$ where $|W^{eid}| = k$. Then, assign a random ordering to $W^{eid}$ to get ordered set $E^{eid}$. Next, send $(\texttt{outcome}, eid, \ell)$ to $P_j$ if $E^{eid}_\ell = j$ and $(\texttt{outcome}, eid, \perp)$ to $P_i$ if $i \notin E^{eid}$. Store $E^{eid}$ and set $\mathcal{E} \leftarrow \mathcal{E} \cup \{eid\}$.
- $\texttt{reveal}(eid, \ell)$ from $P_i$: If $E^{eid}$ is not defined, send $\perp$ to $P_i$ and do nothing. Otherwise, retrieve $E^{eid}$. If $i = E^{eid}_\ell$, broadcast $(\texttt{result}, eid, \ell, i)$. Otherwise, broadcast $(\texttt{rejected}, eid, \ell, i)$.

Figure 1: Description of the Multi Secret Leader Election functionality

.

# 3 Semi Honest Reservoir Sampling Based Protocol

We outline a semi-honest protocol in fig. 2 in the common reference string model for the MSLE functionality.

We define the setup, $\texttt{setup}$, for the protocol as follows:

- Set $k_{\texttt{TFHE}} \xleftarrow{\$} U_q$.
- Publish CRS, $CRS = \texttt{Enc}_{\text{TFHE}}(k_{\texttt{TFHE}})$.

**The MSLE Protocol**, Party $P_i$ realizing $\mathcal{F}_{\mathrm{MSLE}}$ in the semi-honest setting:
Each party has as input a TFHE secret key share $sk_i$ of secret key $sk$ with associated public key $pk$. Initialize a set of finished elections $\mathcal{E}$.

- $(\texttt{register\_elect}, eid)$
  - Sample $s_i, r_i \xleftarrow{\$} \mathbb{F}_q$, $\texttt{ct}_i \leftarrow \texttt{Enc}_{\mathrm{TFHE}}(c_i)$
  - Broadcast $(\texttt{register\_elect\_add}, eid, \texttt{ct}_i)$ to all parties and run $(\texttt{register\_elect\_add}, eid, \texttt{ct}_i)$
- $(\texttt{register\_elect\_add}, eid, \texttt{ct}_j)$ from party $P_j$ for $j \in [n]$
  - If $b_{eid}$ is not stored, set $b_{eid} = 0$
  - $b_{eid} \leftarrow b_{eid} + 1$
  - If $\mathcal{R}_{eid}$ is not stored, $\mathcal{R}_{eid} \leftarrow \texttt{Resevoir.Init}(k)$
  - Set $\texttt{Enc}_{\mathrm{TFHE}}(e_i), \texttt{Enc}_{\mathrm{TFHE}}(\texttt{coin}_i) \leftarrow \texttt{TFHE.Eval}(C_{PRF}, CRS, \texttt{ct}_j)$ where $C$ is the PRF circuit with polynomial stretch and $CRS$ is the key to the PRF.
  - Call $\mathcal{R}_{eid} \leftarrow \texttt{TFHE.Eval}(C, \mathcal{R}_{eid}, \texttt{ct}_j, \texttt{Enc}_{\mathrm{TFHE}}(e_i), \texttt{Enc}_{\mathrm{TFHE}}(\texttt{coin}_i))$ where $C$ is the reservoir sampling circuit for $\texttt{Resevoir.Insert}$. $\texttt{Resevoir}^{eid}\texttt{.Insert}(\texttt{Enc}_{\mathrm{TFHE}}(c_j), \texttt{Enc}_{\mathrm{TFHE}}(e_i), \texttt{Enc}_{\mathrm{TFHE}}(\texttt{coin}_i))$.
  - Store $\mathcal{R}_{eid}$.
- $(\texttt{elect}, eid)$ called by user $P_i$ when $b_{eid} = n$
  - For $\ell \in [k]$, set $\texttt{ct}_\ell \leftarrow \texttt{TFHE.Eval}(C_{\texttt{Resevoir.Output}}, \mathcal{R}_{eid})$
  - Set $p_i \leftarrow \texttt{TFHE.PartDec}(\texttt{ct}_1, \ldots, \texttt{ct}_k, sk_i)$
  - Broadcast $(\texttt{elect\_done}, eid, p_i)$ and call $(\texttt{elect\_done}, eid, p_i)$
- $(\texttt{elect\_done}, eid, p_j)$ from party $P_j$
  - Retrive $\mathcal{P}^{eid}$ if it is stored, otherwise set $\mathcal{P}^{eid} \leftarrow \emptyset$.
  - Set $\mathcal{P}^{eid} \leftarrow \mathcal{P}^{eid} \bigcup \{p_j\}$
  - If $|\mathcal{P}^{eid}| = n$, set $c_{a_1}, \ldots, c_{a_\ell} \leftarrow \texttt{TFHE.FinDec}(\mathcal{P}^{eid})$
  - Store $\mathcal{P}^{eid}$.
- $(\texttt{reveal}, eid, \ell)$ from party $P_i$ if $c_{a_\ell} = c_i$.
  - Broadcast $(\texttt{result}, eid, \ell, i)$

Figure 2: Description of the MSLE protocol

### 3.1 Semi Honest Simulation Security

We will now show that the above protocol is semi-honest simulation secure.

**Theorem 3.1** (Semi-honest simulation security). *Assuming the existence of a PRF with polynomial stretch, a* **TFHE** *scheme with semantic security (definition 1.3) and simulation security (definition 1.4), then the protocol outline in fig. 2 is semi-honest simulation secure.*

*Proof.* We will show that the simulator outlined in fig. 3 is a simulator for the view of corrupt parties $C \subset [n]$ and $|C| < t$ by showing that the simulator's view and the protocol's view are indistinguishable for all calls via the following three lemmas.

**Lemma 3.2.** *The views of* `register_elect`, `register_elect_add` *are simulated by the simulator.*

*Proof.* The simulator firsts simulates the view of the protocol for all calls to `register_elect` from $P_i$, $i \in C$. Note that the simulator uses the same inputs, $s_i, r_i$ as in the protocol and thus the view is identical. Moreover, the view of $(\texttt{register\_elect\_add}, eid, \texttt{ct}'_i)$ is identical to the real protocol as $\texttt{ct}'_i = \texttt{ct}_i$. For $j \notin C$, note that $c'_j \stackrel{c}{\equiv} c_j$ by the hiding property of commitments and thus $\texttt{ct}'_j \stackrel{c}{\equiv} \texttt{ct}_j$. Then, assuming that prior calls to `register_elect` by $P_j$ are simulated by the simulator, the view of $(\texttt{register\_elect\_add}, eid, \texttt{ct}'_j)$ is indistinguishable to the real protocol as `register_elect_add` is completely determined by prior calls to `register_elect_add` and $\texttt{ct}'_j$.

Let $\mathcal{R}_{eid,\texttt{Sim}_C}$ denote the reservoir sampling data-structure in the simulator. We then note that, after $\alpha$, for $\alpha \in [n]$, calls to `register_elect_add`, the simulator's $\mathcal{R}_{eid,\texttt{Sim}_C} \stackrel{c}{\equiv} \mathcal{R}_{eid}$ of the protocol as $\mathcal{R}$ is completly determined by the calls to `register_elect_add`. $\square$

**Lemma 3.3.** *The views of* `elect`, `elect_done` *are simulated by the simulator.*

*Proof.* When `elect` is called by $P_i$ for $i \in C$, we have that for $\texttt{ct}'_\ell = \texttt{TFHE.Eval}(C_{\texttt{Resevoir.Output}}, \mathcal{R}_{eid,\texttt{Sim}_C})$, $\texttt{ct}'_\ell \stackrel{c}{\equiv} \texttt{ct}_\ell$ for all $\ell \in [k]$. This is because $\mathcal{R}_{eid,\texttt{Sim}_C}$ is indistinguishable from that of the real protocol. We also have that, by the ideal functionality of `elect`, $\{ a'_\ell \}_{\ell \in [k]}$ is a random subset of $[n]$ with random order with size $k$. Then, note that $e_i, \texttt{coin}_i = PRF(k_{\texttt{TFHE}}, c_i)$ in the protocol is indistinguishable from a randomly sampled $e_i, \texttt{coin}_i$ by the semantic security of PRFs. Thus, by the correctness of $\texttt{Resevoir.Output}(\mathcal{R})$, $\{ a_\ell \}_{\ell \in [k]}$ is a randomly chosen ordered subset of $[n]$. So then, $\{ a'_\ell \}_{\ell \in [k]} \stackrel{c}{\equiv} \{ a_\ell \}_{\ell \in [k]}$. Remembering that $c'_\beta \stackrel{c}{\equiv} c_\beta$ for all $\beta \in [n]$, we have that $c'_{a_1}, ..., c'_{a_k} \stackrel{c}{\equiv} c_{a_1}, ..., c_{a_k}$. By the simulation security of `TFHE`, we have that the decryption shares, $p'_1, ..., p'_n = \texttt{Sim}_{\texttt{TFHE}}(C_{\texttt{Ident}}, (\{ ct'_\ell \}, c'_{a_1}, ..., c'_{a_k}), [n], \texttt{st})$, for all $i \in [n]$ are simulated by the simulator such that $p'_i \stackrel{c}{\equiv} p_i$ in the real protocol. And thus, all calls to `elect_done` are indistinguishable from the real protocol as `elect_done` is completely determined by $\{ p_i \}_{i \in [n]}$. $\square$

**Lemma 3.4** (`reveal` is simulation secure)**.** *Proof.* Finally, note that `reveal` in the simulator is identical to that of the real protocol as in the semi-honest setting, only parties which have won an election will call `reveal`. $\square$

We thus have that the view of the simulator is identical to that of the real protocol by the above three lemmas. $\square$

**Simulator for Threshold MSLE** $\mathtt{Sim}_C$ where $C \subseteq [n]$ and $|C| < t$:
Initializie a set of finished elections $\mathcal{E}$ and set a random tape for the simulator.

- $(\mathtt{register\_elect}, eid)$ for party $P_j$
  - If $j \in C$ Follow the protocol's $\mathtt{register\_elect}$ using $P_j$'s $s_j, r_j, c_j, \mathtt{ct}_j$ from the real protocol and call $(\mathtt{register\_elect\_add}, eid, \mathtt{ct}_j)$. Set $c'_j \leftarrow c_j$ and store $c'_j$.
  - If $j \notin C$, $s'_j, r'_j \xleftarrow{\$} \mathbb{F}_q, c'_j \leftarrow \mathtt{comm}(s'_j, r'_j), \mathtt{ct}'_j \leftarrow \mathtt{Enc}_{\mathrm{TFHE}}(c'_j)$ Store $c'_j$ and follow the protocol by calling $(\mathtt{register\_elect\_add}, eid, \mathtt{ct}'_j)$.
- $(\mathtt{elect}, eid, \mathtt{out})$ from party $P_j$ where $\mathtt{out}$ is $\mathcal{F}_{\mathrm{MSLE}}.\mathtt{elect}$ output $(\mathtt{outcome}, eid, q)_i$ for all $i \in [n]$ where $q \in \{1, ..., k, \perp\}$.
  - For $\ell \in [k]$, set $\mathtt{ct}'_\ell \leftarrow \mathtt{TFHE.Eval}(C_{\mathtt{Resevoir.Output}}, \mathcal{R}_{eid})$
  - Let $a'_1, ..., a'_k \in [n]$ such that if $q_i = \ell$ then $a'_\ell = i$.
  - Call $p'_1, ..., p'_n \leftarrow \mathtt{Sim}_{\mathtt{TFHE}}(C_{\mathtt{Ident}}, (\{\mathtt{ct}'_\ell\}, c'_{a'_1}, ..., c'_{a'_\ell}), [n], \mathtt{st})$
  - Call $(\mathtt{elect\_done}, eid, p_j)$ from the protocol.
- $(\mathtt{reveal}, eid, \ell)$ for $P_j$ and $\mathcal{F}_{\mathrm{MSLE}}.\mathtt{reveal}$ output $(\mathtt{result}, eid, \ell, j)$ or $(\mathtt{reject}, eid, \ell, j)$.
  - Broadcast $(\mathtt{result}, eid, \ell, j)$

Figure 3: Description of the MSLE protocol

## References


[BEHG20] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020. 1.1

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*, pages 565–596. Springer, 2018. 1.1

[CFG22] Dario Catalano, Dario Fiore, and Emanuele Giunta. Adaptively secure single secret leader election from ddh. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 430–439, 2022. 1.2, 2

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 308–317. IEEE, 1990. 1.2

[JRS17] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. *Cryptology ePrint Archive*, 2017. 1.1, 1.1, 1.2, 1.3, 1.4

[MDPB23] Sahar Mazloom, Benjamin E Diamond, Antigoni Polychroniadou, and Tucker Balch. An efficient data-independent priority queue and its application to dark pools. *Proceedings on Privacy Enhancing Technologies*, 2:5–22, 2023. 1.3, 1.3

[MZ14] John C Mitchell and Joe Zimmerman. Data-oblivious data structures. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014. 1.3, 1.6, 1.7

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 543–553. IEEE, 1999. 1.2

[Tof11] Tomas Toft. Secure data structures based on multi-party computation. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 291–292, 2011. 1.3

[Vit85] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 1.4