Multiple Secret Leaders

Authors here

July 20, 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 λ, 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 λ, 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 $i \in [k]$, $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(pk, C, \texttt{ct}_1, ..., \texttt{ct}_k)$,

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

Definition 1.3 (Semantic Security [JRS17])**.** We have that a $\texttt{TFHE}$ scheme satisfies semantic security for for all λ, 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 λ:

1. On input 1^λ 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 μ'. 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 λ, 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 λ. The experiments are defined as follows:

- $\texttt{Expt}_{\mathcal{A},\text{Real}}(1^\lambda, 1^d)$:
  1. On input the security parameter 1^λ 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 μ_i, the adversary is given $\texttt{TFHE.Encrypt}(pk, \mu_i) \to \texttt{ct}_i$.
  5. The adversary issues a polynomial number of queries, $(S_i \subseteq \{ P_1, ..., P_N \}, C_i)$. for circuites $C_i : \{0, 1\}^k \to \{0, 1\}$. After each query the adversary receuves for $l \in S_i$ the value
  $$\texttt{TFHE.PartDec}(\texttt{TFHE.Eval}(C_i, \texttt{ct}_1, ..., \texttt{ct}_k), sk_l) \to p_l$$
  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^λ 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 μ_i, the adversay is given $\texttt{TFHE.Encrypt}(pk, \mu_i) \to \texttt{ct}_i$.
  5. $\mathcal{A}$ issues a polynomial number of queries of the form $(S_i \subseteq \{ P_1, ..., P_N \}, C_i)$
  6. for circuits $C_i : \{0, 1\}^k \to \{0, 1\}$. After each query, the simulator computes
  $$\texttt{Sim}_{\texttt{TFHE}}(C_i, \{ \texttt{ct}_l \}_{l=1}^k, C_i(\mu_1, ..., \mu_k), \texttt{state}) \to \{ p_l \}_{l \in S_i}$$
  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 π, 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 $\text{Sim}_1, \text{Sim}_2$ are PPT algorithms such that for all PPT adversaries $\mathcal{A}_1, \mathcal{A}_2$ we have that $\left|\mathbf{Pr}[\texttt{Expt}_{\mathcal{A},\text{Real}}(\lambda) = 1] - \mathbf{Pr}[\texttt{Expt}_{\mathcal{A},\text{Sim}}(\lambda) = 1]\right|$ is negligible in λ. The experiments are defined as follows:

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

- `Resevoir.Init`(k) initialize the resevoir sampling algorithm and data structure
- `Resevoir.Insert`$(\mu_i, e_i, \texttt{coin}_i)$ where μ_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 `PQ.Insert`(e_i, μ_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.
- $\mu_{a_1}, \mu_{a_2}, ..., \mu_{a_k} \leftarrow$ `Resevoir.Output`() where $a_1, ..., a_k$ are a uniformly random ordered susbset of $[n]$
  - Call `PQ.ExtractFront` k times setting μ_{a_ℓ} to the ℓ-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.

3 Semi Honest Reservoir Sampling Based Protocol

We outline a semi-honest protocol in section 3 for the MSLE functionality.

The MSLE functionality $\mathcal{F}_{\text{MSLE}}$: Initialize $E, R \leftarrow \emptyset, b \leftarrow 0$. Initialize S to denote the set of sets of active participants in each round. 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,

- `register` from party P_i, set $R \leftarrow R \cup \{(i, b)\}$, broadcast $(\texttt{register}, i)$ to all parties and set $b \leftarrow b + 1$
- `register_elect`(eid, w) from party P_i. If $eid \in \mathcal{E}$ send $\perp$ to P_i and do nothing. If $(i, w) \notin R$ or $(i, w) \in S_{eid}$, send $\perp$ to P_i and do nothing. If S_{eid} is not defined, set $S_{eid} \leftarrow \{(i, w)\}$. Otherwise, set $S_{eid} \leftarrow S_{eid} \cup \{(i, w)\}$.
- `elect`(eid) Elect k leaders from $S_{eid} \subseteq R$ parties. If $|S| \geq k$ and $eid \notin \mathcal{E}$, 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 $(\texttt{outcome}, eid, a)$ to P_j for all $E^{eid}_a = (j, \cdot)$ and $(\texttt{outcome}, eid, \perp)$ to P_i if $(i, \cdot) \notin E^{eid}$. Store $E \leftarrow E \bigcup \{E^{eid}\}$. Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{eid\}$.
- `reveal` from P_i: for $E_{eid} \in E$, 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

.

3.1 Semi Honest Simulation Security

We can show semi-honest simulation security by showing that the view of each party i in the real protocol can be simulated throughout the course of one election and then that this simulation can be extended for a polynomial number of elections.

We will now show that $\texttt{Sim}_i$ is indeed a simulator for the view of π_{MSLE} for `register_elect`, `elect`, and `reveal`.

Lemma 3.1 (`register_elect` is simulation secure)**.** *Proof.* The simulator gets as input party i's secret value, s_i, party i's `TFHE` share sk_i, eid, and ticket number w. We note that if $eid \in \mathcal{E}$ or $(i, w) \notin R$, then the simulator can simply output $\perp$ and is identical to the real protocol. Otherwise, if party P_i calls `register_elect`, then the simulator knows the input of P_i and can simulate the protocol honestly. If party P_j call `register_elect` for some $j \neq i$, then the view of the protocol is that of

$$(\texttt{Enc}_{\text{TFHE}}(c_j), \texttt{Enc}_{\text{TFHE}}(e_i, \texttt{coin}_i), \texttt{Resevoir}^{eid}.\texttt{Insert}(\texttt{Enc}_{\text{TFHE}}(c_j, e_i, \texttt{coin}_i))).$$

Note that this view is completely determined by $\texttt{Enc}_{\text{TFHE}}(c_j)$. Also, note that c_j is drawn from a random distribution and is not in the view of the real protocol. Thus, by semantic security of `TFHE` (definition 1.3) we have that $\texttt{Enc}_{\text{TFHE}}(c_j) \overset{c}{\equiv} \texttt{Enc}_{\text{TFHE}}(c'_j)$ where c'_j is a commitment to a random value. □

Lemma 3.2 (`elect` is simulation secure)**.** *Proof.* If the view of the real protocol is $\perp$ because eid was already called, then the simulator can simply output $\perp$ and is thus identical to the real protocol. Otherwise, note that the view of the protocol is

$$(c_{a_1}, ..., c_{a_k}, p_1, ..., p_t, \texttt{view}(C(\texttt{Enc}_{\text{TFHE}}(c_{a_1}, e_1, \texttt{coin}_1), ..., \texttt{Enc}_{\text{TFHE}}(c_{a_k}, e_k, \texttt{coin}_k)))).$$

We will now show that simulator can simulate the above view. The simulator needs to "fix" the output of `elect` to yield a list of commitments such that, if $a'_\ell = j$ if party j wins election ℓ. The

The MSLE Protocol π_{MSLE}: Initialize $b = 0$. Fix some $k \in \mathbb{N}$ to denote the number of rounds. Initialize an empty set of tickets, R. Initialize an empty lookup of sets of parties in each election, S. Initialize an empty lookup of reservoir sampling data structures $\mathcal{R}$ and a set of finished elections $\mathcal{E}$ Initialize an empty lookup of election results, E.
Initialize $S \leftarrow \emptyset$ to denote the set of sets of active participants in each round.

- `initialize`
  - Set $n = 0$
  - Sample a random `TFHE` secret key, public key pair sk, pk and publish pk.
  - Sample some secret $s_{\texttt{TFHE}}$ and publish $\texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}})$. This will be the key to the PRF
- `register` from party P_i
  - If party P_i does not already have a TFHE share, create a TFHE share, sk_i, for P_i and send the share over a secure channel to P_i.
  - $R \leftarrow R \cup \{(i, b)\}$, set $b \leftarrow b + 1$.
- `register_elect`$(eid, \texttt{Enc}_{\text{TFHE}}(c_i), w)$ from party P_i where $c_i = \texttt{comm}(s_i)$ for a randomly sampled secret s_i.
  - If $eid \in \mathcal{E}$, then send $\perp$ to P_i and do nothing.
  - If $(i, w) \in S_{eid}$ or $(i, w) \notin R$, send $\perp$ and do nothing.
  - Otherwise, if $\mathcal{R}_{eid} \notin \mathcal{R}$, $\mathcal{R}_{eid} \leftarrow \texttt{Resevoir.Init}(k)$
  - The CRS will be used to run a PRF to get $\texttt{Enc}_{\text{TFHE}}(e_i), \texttt{Enc}_{\text{TFHE}}(\texttt{coin}_i) = \texttt{Enc}_{\text{TFHE}}(\texttt{PRF}(c_i, s_{\texttt{TFHE}}))$ for $i \in S$
  - $\texttt{Enc}_{\text{TFHE}}(c_i)$ will be fed to the encrypted streaming sampler along with randomness via $\texttt{Resevoir}^{eid}\texttt{.Insert}(\texttt{Enc}_{\text{TFHE}}(c_i), \texttt{Enc}_{\text{TFHE}}(e_i), \texttt{Enc}_{\text{TFHE}}(\texttt{coin}_i))$.
- `elect`(eid)
  - If $eid \in \mathcal{E}$, then return $\perp$ and do nothing.
  - The encrypted streaming sampler will output a list of k messages via calling $\texttt{Resevoir}^{eid}\texttt{.Output}()$ k times: $\texttt{Enc}_{\text{TFHE}}(c_{a_1}), \texttt{Enc}_{\text{TFHE}}(c_{a_2}), ..., \texttt{Enc}_{\text{TFHE}}(c_{a_k})$.
  - Then, at least t parties will submit decryption shares, $p_i = \texttt{TFHE.PartDec}(\texttt{Enc}_{\text{TFHE}}(c_{a_1}), ..., \texttt{Enc}_{\text{TFHE}}(c_{a_2}))$ to get $c_{a_1}, ..., c_{a_k} = \texttt{TFHE.FinDec}(\{p_i\})$.
  - Add $E_{eid} = \{c_{a_1}, ..., c_{a_k}\}$ to E.
- `reveal`$(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c_i'))$ from P_i
  - P_i submits a proof that they know the opening to c_{a_ℓ}. If this proof verifies, send out $(\texttt{result}, eid, \ell, i)$ to all parties. Otherwise, send out $(\texttt{rejected}, eid, \ell, i)$ to all parties.

Figure 2: Description of the MSLE protocol

Simulator for Threshold MSLE $\texttt{Sim}_i$: Initialize $b = 0$. Fix some $k \in \mathbb{N}$ to denote the number of rounds. Initialize an empty set of tickets, R, an empty lookup of sets of parties in each election, S, a set of finished elections $\mathcal{E}$, an empty lookup of election results, E, and a set $S \leftarrow \emptyset$ to denote the set of sets of active participants in each round. Moreover, set a random tape for the simulator. The simulator knows, input for party P_i, sk_i and s_i.

- `register` from party P_j
  - $R \leftarrow R \cup \{(i, b)\}$, set $b \leftarrow b + 1$.
  - Simulate the secure channel communication with party P_j if $i \neq j$.
  - Run `register` honestly if $i = j$ by sending sk_i to P_i
- `register_elect`(eid, w, s_i, sk_i) from party P_j.
  - If $j \neq i$, the simulator samples a random value s'_j and follows the protocol directly using $s'_{\texttt{TFHE}}$ and $\texttt{Enc}_{\text{TFHE}}(c'_j)$ where $c'_j = \texttt{comm}(s'_j)$. Store c'_j in a lookup table as well as its committed message.
  - If $j = i$, the simulator will use the commitment $c_i = \texttt{comm}(s_i)$. And run `register_elect` as is in the protocol
- `elect`$(eid, \texttt{out}, s_i, sk_i)$ where the `out` is output $\perp$ or $(\texttt{outcome}, eid, q)_i$ for all $i \in [n]$ where $q \in \{1, ..., k, \perp\}$.
  - Let $\overline{n}$ be the total number of successful calls to `register_elect`$(eid, ., .)$ before the election for eid
  - If the output is $\perp$, return $\perp$.
  - Let $a'_1, ..., a'_k$ be the ordered set of parties that won the election. I.e. if $q_i = \ell$ then $a'_\ell = i$.
  - The simulator then gives $\texttt{Sim}_{\texttt{TFHE}}(C, \{\texttt{Enc}_{\text{TFHE}}(c'_j)\}_{j \in [\overline{n}]}, c'_{a'_1}, ..., c'_{a'_\ell}, S, \texttt{st})$ to the TFHE simulator to get a list of partial decryptions, $p_1, ..., p_t$ where S is a qualified set and C is the reservoir sampling circuit with PRF seed $s_{\texttt{TFHE}}$ hardcoded. Note that this sets the decryption of the output of the TFHE circuit sampling to $c'_{a'_\ell}$ for $\ell \in [k]$.
  - Add $E_{eid} = \left\{c'_{a'_1}, ..., c'_{a'_k}\right\}$ to E.
- `reveal`$(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c'_j))$ from P_j and output $\perp$ or $(\texttt{result}, eid, \ell, j)$.
  - As the simulator has knowledge of the openings to all of the commitments c'_j used, the simulator to honestly run `reveal`$(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c'_j))$ with commitment c'_j and opening s'_j.
  - Note that in the semi-honest setting, the output is never $\perp$

Figure 3: Description of the MSLE protocol

TFHE simulator can simply output the decryption shares, $p_1, ...p_t$ using

$$\mathtt{Sim}_{\mathtt{TFHE}}(C, \{\, \mathtt{Enc}_{\mathrm{TFHE}}(c'_j, e_j, \mathtt{coin}_j) \,\}_{j\in[\overline{n}]}, c'_1, ..., c'_{a_\ell}, S, \mathtt{st}).$$

Note that the reservoir sampling circuit is indeed simulated by $\mathtt{Sim}_{\mathtt{TFHE}}$ and the decryption shares are simulated as well such that the cipher texts decrypt to $c'_{a'_1}, ..., c'_{a'_k}$. We also have that, by the ideal functionality of `elect`, $\{\, (a'_\ell, .) \,\}_{\ell\in[k]}$ is a randomly chosen ordered subset from S_{eid}. We can note that, by the correctness of reservoir sampling, the output of $\mathtt{Resevoir}^{eid}.\mathtt{Output}()$ in the protocol is a randomly chosen ordered subset from S_{eid}. We then have that the distribution of the simulator's $\{\, (a'_\ell, .) \,\}_{\ell\in[k]} \equiv \{\, (a_\ell, .) \,\}_{\ell\in[k]}$ where a_ℓ is the party that won election ℓ in the protocol. We then have that, by the simulation security of `register_elect` (lemma 3.1), $c'_{a'_1}, ..., c'_{a'_k} \overset{\mathrm{c}}{\equiv} c_{a_1}, ..., c_{a_k}$. □

Lemma 3.3 (`reveal` is simulation secure)**.** *Proof.* Note that the simulator can simply run `reveal` honestly as the simulator has knowledge of the openings to all of the commitments c'_j used and thus has an identical view to that of the real protocol. □

4 Malicious Adversary Secure Protocol

4.1 Static Malicious Adversary Security

Consider parties P_i to be corrupted where $i \in C$ and $C \subseteq [n]$ such that. $|C| < t$.

Lemma 4.1 (`register_elect` is simulation secure)**.** *Idea: if $\perp$, just do same If: $j \notin C$, just do same If: $j \in C$, then use the NIZK to extract out the commitment from the FHE ciphertext or bot Then: S follows protocol*

Lemma 4.2 (`elect` is simulation secure)**.** *Idea: if $\perp$, just do same If: $j \notin C$, just do same Output list from streaming sampler using TFHE sim, set to correct sequence (irrespective of advers) Ask advers to generate decryption shares for each corrupted party and broadcast proof For non-corrupted parties, use TFHE sim to generate decryption shares, simulate proofs For corrupted parties, ask advers to generate decryption shares and proofs If proofs check, use them, if not discard the decryption shares*

Lemma 4.3 (`reveal` is simulation secure)**.** *If an honest party sends the req, we simulate a commitment opening according to the stored commitment (that we know) If corrupted party sends req, pretending to know opening to not their commitment, we say $\perp$ (this is likely as we have the hiding property) If corrupted party sends req correctly, we use the de-commitment that they send*

The MSLE Protocol π_{MSLE}: Initialize $b = 0$. Fix some $k \in \mathbb{N}$ to denote the number of rounds. Initialize an empty set of tickets, R, an empty lookup of sets of parties in each election, S, an empty lookup of reservoir sampling data structures $\mathcal{R}$, and a set of finished elections $\mathcal{E}$. Initialize an empty lookup of election results, E.

- `initialize`
  - Set $n = 0$
  - Sample a random `TFHE` secret key, public key pair sk, pk and publish pk.
  - Sample some secret $s_{\texttt{TFHE}}$ and publish $\texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}})$. This will be the key to the PRF
- `register` from party P_i
  - If party P_i does not already have a TFHE share, create a TFHE share, sk_i, for P_i and send the share over a secure channel to P_i.
  - $R \leftarrow R \cup \{(i, b)\}$, set $b \leftarrow b + 1$.
- `register_elect`$(eid, \texttt{Enc}_{\text{TFHE}}(c_j), w)$ from party P_j where $c_j = \texttt{comm}(s_j)$ for a randomly sampled secret s_j. Every honest party does the following:
  - If $eid \in \mathcal{E}$, then broadcast $\perp$ and do nothing.
  - TODO: NEED TO CHECK VALID ENCRYPTION
  - If $(i, w) \in S_{eid}$ or $(i, w) \notin R$, broadcast $\perp$ and do nothing.
  - Otherwise, if $\mathcal{R}_{eid} \notin \mathcal{R}$, $\mathcal{R}_{eid} \leftarrow \texttt{Resevoir.Init}(k)$
  - If `Inps` is not initialize, set it to an empty list. Then, set $\texttt{Inps}[w] \leftarrow \texttt{Enc}_{\text{TFHE}}(c_j)$
  - The CRS will be used to run a PRF to get $\texttt{Enc}_{\text{TFHE}}(e_j), \texttt{Enc}_{\text{TFHE}}(\texttt{coin}_j) = \texttt{Enc}_{\text{TFHE}}(\texttt{PRF}(c_j, s_{\texttt{TFHE}}))$
  - $\texttt{Enc}_{\text{TFHE}}(c_j)$ will be fed to the encrypted streaming sampler along with randomness via $\texttt{Resevoir}^{eid}\texttt{.Insert}(\texttt{Enc}_{\text{TFHE}}(c_j), \texttt{Enc}_{\text{TFHE}}(e_j), \texttt{Enc}_{\text{TFHE}}(\texttt{coin}_j))$.
- `elect`(eid). Each party, P_i does the following:
  - If $eid \in \mathcal{E}$, then broadcast $\perp$ and do nothing.
  - The encrypted streaming sampler will output a list of k messages via calling $\texttt{Resevoir}^{eid}\texttt{.Output}()$ k times: $\texttt{Enc}_{\text{TFHE}}(c_{a_1}), \texttt{Enc}_{\text{TFHE}}(c_{a_2}), ..., \texttt{Enc}_{\text{TFHE}}(c_{a_k})$.
  - TODO: NEED TO CHECK ELECTION SET SIZE
  - Set $n_\perp \leftarrow 0$
  - Generate decryption share p_i and set $\texttt{Shares}_{eid} \leftarrow \{ p_i \}$
  - Send $(\texttt{prove}, sid, (p_i, \texttt{Inps}, \texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}})))$ to $\mathcal{F}_{\texttt{NZIK}}^{Rel}$.
  - Upon receiving $\perp$, set $n_\perp \leftarrow n_\perp + 1$.
  - Upon receiving $(\texttt{proof}, sid, j, (p_j, \texttt{Inps}', \texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}})'))$ from $\mathcal{F}_{\texttt{NZIK}}^{Rel}$, check that $\texttt{Inps} = \texttt{Inps}$ and that $\texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}}) = \texttt{Enc}_{\text{TFHE}}(s_{\texttt{TFHE}})'$. If the equalities hold, set $\texttt{Shares}_{eid} \leftarrow \texttt{Shares}_{eid} \bigcup \{ p_j \}$.
  - If $|\texttt{Shares}_{eid}| \geq t$, use the decryption shares to get $\{ c_{a_1}, ..., c_{a_k} \} = \texttt{TFHE.FinDec}(\texttt{Shares})$. And add $E_{eid} = \{c_{a_1}, ..., c_{a_k}\}$ to E.
  - If $n_\perp \geq (|\texttt{Inps}| - t) + 1$, then abort and output $\perp$.
  - TODO: When to abort? At time out, at next election? Etc.
- `reveal`$(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c_i'))$ from P_i
  - P_i submits a proof that they know the opening to c_i. If this proof verifies, send out

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