Multiple Secret Leaders

Authors here

July 17, 2023

1 Some Notation

1. We will have n parties
2. We will have k leaders elected
3. We will have a "bid" published by a user $i \in [n]$ be denoted as b_i
4. We will denote a commitment as $\texttt{comm}_i$
5. We will denote some generic CRHF as h
6. We will say $\texttt{Enc}_{\text{TFHE}}$ and $\texttt{Dec}_{\text{TFHE}}$ for TFHE encoding and decoding respectively

2 Preliminaries

2.1 Threshold FHE

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

Definition 2.1 (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 and simulation security as outlined in, [BEHG20].

Definition 2.2 (Evaluation Correctness [JRS17])**.** . We have that TFHE scheme is correct if for all λ, depth bounds d, access structure $\mathbb{A}$, circuit $C : \{ 0, 1 \}^k \to \{ 0, 1 \}$ 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 2.3 (Simulation Security [JRS17])**.** We say that a 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.

2.2 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 2.4 (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 2.5 (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.

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

3 TFHE MSLE Protocol

We use a similar notion of ideal functionality for a multi-secret leader election from the ideal functionality of single secret leader election of LS: CITE.

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 $R \leftarrow R \cup \{(i, n)\}$, broadcast $(\texttt{register}, i)$ to all parties and set $b \leftarrow b + 1$
- $\texttt{register\_elect}(eid, w)$ 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 $S_{eid} \leftarrow S_{eid} \cup \{(i, w)\}$.
- $\texttt{elect}(eid)$ 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 $E_a^{eid} = (j, \cdot)$ 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 $i = E_\ell^{eid}$, broadcast $(\texttt{result}, eid, \ell, i)$. Otherwise, broadcast $(\texttt{rejected}, eid, \ell, i)$.

Figure 1: Description of the MSLE functionality, heavily based on the description of SSLE in LS: CITE Dario and CFG21

.

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.

Let $\overline{n}$ be the total number of calls to $\texttt{register\_elect}(eid, ., .)$ before the election for eid, $\texttt{elect}(eid)$ is called and that $k \leq \overline{n} \leq n$ as if $\overline{n} < k$, then `elect` returns $\perp$ and is trivial to simulate. Further, let $\overline{k}$ be the total number of calls to $\texttt{reveal}(eid, ., .)$ which did not reject, again we have that $\overline{k} \leq k$. We will now show that a simulator can simulate the view of each party i after $\overline{n}$ calls to $\texttt{register\_elect}(eid, .)$, a call to $\texttt{elect}(eid)$, and $\overline{k}$ calls to $\texttt{reveal}(eid, .)$ for a fixed eid.

The simulator, $\texttt{Sim}_i$, will proceed as follows with input s_i, sk_i, eid, w and output of election $(\texttt{outcome}, eid, q)_i$ for $i \in [\overline{n}]$ and $(\texttt{result}, eid, \ell, j)$ for $\ell \in K$ where $K \subseteq [k]$ and $|K| = \overline{k}$:

1. $\texttt{Sim}_i$ sets a random, local tape
2. The simulator samples c'_j for all $j \neq i$ where c'_j is a commitment to a random value. The simulator also computes $c_i = \texttt{comm}(s_i)$.
3. The simulator runs $\texttt{register\_elect}(eid, \texttt{Enc}_{\text{TFHE}}(c'_j), w_j)$ for $i \in [\overline{n}]$ as in the protocol.
4. The simulator runs the first two steps of $\texttt{elect}(eid)$, having the reservoir sample output $\texttt{Enc}_{\text{TFHE}}(c'_{a_1}), ..., \texttt{Enc}_{\text{TFHE}}(c'_{a_k})$. The simulator then gives $\texttt{Sim}_{\texttt{TFHE}}(C, \{\texttt{Enc}_{\text{TFHE}} c'_j\}_{j \in [\overline{n}]}, c'_1, ..., c'_{\overline{n}}, S, \texttt{st})$ to the TFHE simulator to get a list of partial decryptions, $p_1, ..., p_{t-1}$ 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 reservoir sampling to c'_{a_ℓ} for $\ell \in [k]$.
5. For all calls to $\texttt{reveal}(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c'_j))$ for $\ell \in K$, the simulator simply runs the protocol as the simulator has knowledge of the openings to all of the commitments c'_j used.

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 $\texttt{Enc}_{\mathrm{TFHE}}(s_{\texttt{TFHE}})$. 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 $b \leftarrow b + 1$.
- $\texttt{register\_elect}(eid, \texttt{Enc}_{\mathrm{TFHE}}(c_i), w)$ 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 $\texttt{Enc}_{\mathrm{TFHE}}(e_i), \texttt{Enc}_{\mathrm{TFHE}}(\texttt{coin}_i) = \texttt{Enc}_{\mathrm{TFHE}}(\texttt{PRF}(c_i, s_{\texttt{TFHE}}))$ for $i \in S$
  - $\texttt{Enc}_{\mathrm{TFHE}}(c_i)$ will be fed to the encrypted streaming sampler along with randomness via $\texttt{Resevoir}^{eid}\texttt{.Insert}(\texttt{Enc}_{\mathrm{TFHE}}(c_i), \texttt{Enc}_{\mathrm{TFHE}}(e_i), \texttt{Enc}_{\mathrm{TFHE}}(\texttt{coin}_i))$.
- $\texttt{elect}(eid)$
  - 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: $\texttt{Enc}_{\mathrm{TFHE}}(c_{a_1}), \texttt{Enc}_{\mathrm{TFHE}}(c_{a_2}), ..., \texttt{Enc}_{\mathrm{TFHE}}(c_{a_k})$.
  - Then, at least t parties will submit decryption shares, $p_i = \texttt{TFHE.PartDec}(\texttt{Enc}_{\mathrm{TFHE}}(c_{a_1}), ..., \texttt{Enc}_{\mathrm{TFHE}}(c_{a_2}))$ to get $c_{a_1}, ..., c_{a_k} = \texttt{TFHE.FinDec}(\{p_i\})$.
  - Add $E_{eid} = \{c_{a_1}, ..., c_{a_k}\}$ to E.
- $\texttt{reveal}(eid, \ell, \texttt{Enc}_{\mathrm{TFHE}}(c'_i))$ 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 $\mathcal{S}_i$: 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. sk.

- `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
  - $R \leftarrow R \cup \{(i, b)\}$, set $b \leftarrow b + 1$.
  - Call the ?simulator?
  - Simulate the secure channel communication with party 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 $q \in \{1, ..., k, \perp\}$.
  - 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 $a_\ell = i$.
  - Setup the $\texttt{Sim}_{\texttt{TFHE}}$ such that it controls S parties with $|S| = t - 1$
  - Set the output of the ℓ-th call, for $\ell \in [k]$ to $\texttt{Resevoir}^{eid}\texttt{.Output}()$ to $\texttt{Enc}_{\text{TFHE}}(c'_{a_\ell})$
  - Call $\texttt{Resevoir}^{eid}\texttt{.Output}$ k times
  - Call the TFHE simulator to compute, $\texttt{Sim}_{\texttt{TFHE}}(C, \{\texttt{Enc}_{\text{TFHE}}(c'_{a_\ell})\} \bigcup \{s'_{\texttt{TFHE}}\}, \{c'_{a_\ell}\}, \texttt{state})$ and get a list of partial decryptions, $p_1, ..., p_{t-1}$.
  - Set p_t to a decryption such that $\texttt{Enc}_{\text{TFHE}}(c'_{a_\ell})$ decrypts to c'_{a_ℓ}.
  - Add $E_{eid} = \{c_{a_1}, ..., c_{a_k}\}$ to E.
- `reveal`$(eid, \ell, \texttt{Enc}_{\text{TFHE}}(c'_j))$ from P_j and output $\perp$ or $(\texttt{result}, eid, \ell, j)$.
  - If the outcome is $\perp$, return $\perp$.
  - Otherwise, if $j = i$ then we run `reveal` honestly by proving knowledge of the opening, s_i, to c_i. If $j \neq i$ prove knowledge of opening to c'_j using the stored lookup table from `register_elect`.

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. 2.1

[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. 2.1

[JRS17] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. *Cryptology ePrint Archive*, 2017. 2.1, 2.1, 2.2, 2.3

[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. 2.2, 2.2

[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. 2.2, 2.4, 2.5

[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. 2.2

[Vit85] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 2.3