Multiple Secret Leaders

Authors here

June 29, 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 Sketching Stuff Out

Say we want to elect k leaders (maybe with or without repetition) secretly out of n parties, we can run SSLE k times but that'd be painful. Let's not do that.

2.1 Simple sorting

A simple approach with runtime (assuming $k \leq n$) $O(k + n \log n)$ is to use sorting. At a high level, the idea is to somehow randomly sort the n parties then elect the top k parties in the sort. To do this we can use TFHE in a very similar way to the SSLE paper.

Algorithm 1 Simple Sorting Evaluation

```
Input: List of bids, $b_1, b_2, ..., b_n$
  $\alpha \leftarrow \texttt{ROM}(b_1, b_2, ...)$
  for $i \in [n]$ do
      $b'_i \leftarrow (\texttt{Enc}_{\text{TFHE}}(r_i) - \alpha, \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_i))$
  $\texttt{sorted} = [b'_{j_1}, b'_{j_2}, ...] \leftarrow \texttt{Sort}_{\text{TFHE}}([b'_1, b'_2, ...])$
  return first $k$ elements of sorted
```

1. **Setup**: Same exact setup with TFHE in SSLE. Setup TFHE and distribute shares. Each party also has some secret s_i and commitment $\texttt{comm}_i = \texttt{comm}(s_i)$.
2. **Publish Bid**: Each party samples $r_i \leftarrow U$ and publish $b_i = (\texttt{Enc}_{\text{TFHE}}(r_i), \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_i))$.

3. **Evaluation**: This is done by one party whom may even be an external party. See algorithm 1 for details.

4. **Decoding**: At least t parties publish their threshold decryptions of the return from the evaluation.

5. **Proving**: For a party i to prove that they are the selected leader for the jth slot, they prove that they know the secret to produce $\texttt{comm}_i$ where $\texttt{comm}_i$ is the decrypted commitment for the ith slot.

2.2 Distributed/ Streaming Simple Sorting

Assume that n, k are a power of 2. We will keep a lot of the same from sorting but now look at selection as a sort of tournament. As a note, security is slightly reduced here as a participating party can know that they did not win before the final outcome of the election. I do not know if this matters.

For completeness, we will write out each step again

Algorithm 2 `PlayGame`. Homomorphically plays a game to decide which incoming bid "wins"

```
Input: Point $\alpha \in \mathbb{F}_q$,
  two bids, $b_1 = (\texttt{Enc}_{\text{TFHE}}(r_1), \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_1)), (\texttt{Enc}_{\text{TFHE}}(r_2), \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_2))$
Output: Winning bid. The winning bid should be indistinguishable from a random bid to the
  non-players
  $\texttt{closer} \leftarrow \texttt{Enc}_{\text{TFHE}}(r_1 - \alpha > r_2 - \alpha)$
  return $\texttt{closer} \cdot b_1 + (1 - \texttt{closer}) \cdot b_2$
```

Algorithm 3 `EvalStream`. Evaluates a stream of bids as they come in.

```
Input: $L$, set of $(\texttt{bid}, \texttt{level})$ pairs. New bid and level, $(b, \ell)$. Also, stop level $\ell_{\text{stop}}$
Output: New list $L$ with remaining evaluations
  if there is some for some $b', (b', \ell) \in L$ then
    $\alpha \leftarrow \texttt{ROM}(b, b', \ell)$
    $\bar{b} \leftarrow \texttt{PlayGame}(b, b')$
    $L' \leftarrow L \setminus \{ (b', \ell) \}$
    if then$\ell_{\text{stop}} = \ell - 1$
      return $L \bigcup \{ (\bar{b}, \ell - 1) \}$
    else
      return $\texttt{EvalStream}(L', (\bar{b}, \ell - 1))$
  else
    $L' \leftarrow L \bigcup \{ (b, \ell) \}$
    return $L'$
```

1. **Setup**: Same exact setup with TFHE in SSLE. Setup TFHE and distribute shares. Each party also has some secret s_i and commitment $\texttt{comm}_i = \texttt{comm}(s_i)$.

2. **Publish Bid**: Each party samples $r_i \leftarrow U$ and publish $b_i = (\texttt{Enc}_{\text{TFHE}}(r_i), \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_i))$ each tagged with level $\log_2 n + 1$.

3. **Evaluation**: This step is different than that in the simple sorting version (section 2.1). Now, we can evaluate in a streaming fashion and even distribute evaluation (though we'll not go into details about the distributed implementation). See algorithm 3 for the algorithm. The idea is that we keep a list of bids and their levels; when a new bid comes in, we greedily remove any bids from the list which we can. We can run algorithm 3 until there is only one bid left in the list or until all bids up to a level have been processed. So, say that we have $k = 4$, we can run the algorithm until we are two levels away (in a tournament tree) from the final level. More formally, when we have processed all bids up to level $\ell_{\text{stop}} = \log_2 k + 2$ (and we only have bids at level $\log_2 k + 1$ in the list) we return the list and stop processing. When there are k elements in this list, all at level $\log_2 k + 1$, we are done.

   LS: I may have an off by one error here though I do not think so.

4. **Decoding**: At least t parties publish their threshold decryptions of the return from the evaluation

5. **Proving**: For a party i to prove that they are the selected leader for the jth slot, they prove that they know the secret to produce $\texttt{comm}_i$ where $\texttt{comm}_i$ is the decrypted commitment for the ith slot

3 Data Independent Streaming Sampler

Our construction makes heavy use of a data independent streaming sampler which we will define below. The streaming sampler relies on a data independent priority queue and in turn makes use of LS: Cite Shi protocol for an oblivious priority queue.

Data Independent Queue Ideal Functionality

Encrypted Data Independent Queue Ideal Functionality $\mathcal{F}_{\text{MSLE}}$: Initialize L to be an empty list

- $\texttt{Insert}(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$ upon receiving $(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$, place $(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$ into an initially empty list L and broadcast $(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$ to all parties as well as termination of insert.

- $y \leftarrow \texttt{ExtractMin}()$ Delete $(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$ with the lowest p value. Then, set y to be a re-encryption of x such that y is indistinguishable from all other elements in L.

- $y \leftarrow \texttt{GetMin}()$ Get $(\texttt{Enc}_{\text{TFHE}}(p), \texttt{Enc}_{\text{TFHE}}(x))$ with the lowest p value. Then, set y to be a re-encryption of x such that the encryption of x and y are indistinguishable.

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

.

Streaming Sampler Ideal Functionality

The ideal functionality of a data independent sampler is given by $\mathcal{F}_{\text{SAMPLE}}$ (TODO: cite Elaine) which outputs a random subset of size k from a stream S with a random ordering.

Encrypted Data Independent Streaming Sampler Ideal Functionality $\mathcal{F}_{\text{MSLE}}$: Initialize L to be an empty list

- $\texttt{Insert}(\texttt{Enc}_{\text{TFHE}}(x))$ upon receiving $\texttt{Enc}_{\text{TFHE}}(x)$, place $\texttt{Enc}_{\text{TFHE}}(x)$ into an initially empty list L
- $S \leftarrow \texttt{Sample}()$ Randomly sample an ordered and re-encrypted subset $S \subseteq L$ such that $|S| = k$, S has a random ordering and S_a is indistinguishable from all elements in L.

Figure 2: Description of ideal sampling functionality

.

Protocol for Streaming Sampler

This protocol is identical to that of (TODO: cite Elaine) except that the we replace the oblivious queue with the stronger notion of a data independent queue and public coin randomness with randomness generated within a PRF.

4 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 $E, R \leftarrow \emptyset, \leftarrow 0$. Fix some $k \in \mathbb{N}$ to denote the number of rounds. Upon receiving,

- $\texttt{register}$ from party P_i, set $R \leftarrow R \cup \{(i, n)\}$, broadcast $(\texttt{register}, i)$ to all parties and set $n \leftarrow n + 1$
- $\texttt{elect}(eid)$ k leaders from all honest parties. If $|R| \geq k$ and eid was not requested before, randomly sample $W^{eid} \subseteq R$ 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 $E \leftarrow E \bigcup \{E^{eid}\}$.
- $\texttt{reveal}(eid, a)$ from P_i: for $E_{eid} \in E$, if $i = E^{eid}_a$, broadcast $(\texttt{result}, eid, a, i)$. Otherwise, broadcast $(\texttt{rejected}, eid, a, i)$.

Figure 3: Description of the MSLE functionality, heavily based on the description of SSLE in LS: CITE Dario and CFG21

.

We wil realize MSLE via the following algorithm:

- Each party i will submit a message $\texttt{Enc}_{\text{TFHE}}(b_i)$ to the oblivious streaming sampler for some secret b_i which is a commitment to a secret known only by party i.
- After all parties submitted there messages, the oblivious streaming sampler will output a list of k messages: $\texttt{Enc}_{\text{TFHE}}(b_{a_1}), \texttt{Enc}_{\text{TFHE}}(b_{a_2}), ..., \texttt{Enc}_{\text{TFHE}}(b_{a_k})$.
- Then, at least t parties will submit decryption shares to decrypt $\texttt{Enc}_{\text{TFHE}}(b_{a_1}), ..., \texttt{Enc}_{\text{TFHE}}(b_{a_k})$.

- Each party will then build check if they won an election by seeing if their commitment is in the list of decrypted messages. A party can then prove that they won an election a by showing that they know the opening to b_a.

4.1 Correctness

4.2 Simulation Security

To show simulation security, we will prove that, given a party's input and output in the ideal model, a simulator can simulate the distribution of the view in the protocol for the party. Note that because the protocol is deterministic, it suffices to prove simulation for only the party's output.

More formally we will show that for party i,

$$\mathrm{Sim}_i\left(s_i, r_i, \mathcal{F}_{\mathrm{MSLE}}.\mathtt{register}_i\right) \overset{c}{\approx} \mathtt{view}_{\mathtt{register}_i}((s_0, r_0), ..., (s_n, r_n)), \tag{1}$$

$$\mathrm{Sim}_i\left(s_i, b_i, r_i, \mathcal{F}_{\mathrm{MSLE}}.\mathtt{elect}(eid)_i\right) \overset{c}{\approx} \mathtt{view}_{\mathtt{elect}(eid)_i}((s_0, b_0), ..., (s_n, b_n)), \tag{2}$$

and

$$\mathrm{Sim}_i\left(b_{a,1}, \ldots, b_{a,k}, s_i, r_i, \mathcal{F}_{\mathrm{MSLE}}.\mathtt{reveal}(eid, a)_i\right) \overset{c}{\approx} \mathtt{view}_{\mathtt{reveal}(eid,a)_i}(), \tag{3}$$

Lemma 4.1. *We will first show that eq. (1) is simulation secure.*

Proof. □

Lemma 4.2. *We will now show that eq. (2) is simulation secure.*

Proof. □

Lemma 4.3. *We will now show that eq. (3) is simulation secure.*

Proof. □

References