Multiple Secret Leaders

Authors here

June 26, 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 $\texttt{sorted}$
```

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 Oblivious PQ

3.1 Oblivious Priority Queue

We will make extensive use of an oblivious priority queue which works over FHE.

Definition 3.1 (Oblivious Priority Queue). An oblivious priority queue is a priority queue which can be "run" by a party oblivious to the actual messages of the priority queue. More formally, an oblivious priority queue has functionality

- $\texttt{PQInsert}(\texttt{ct}, \texttt{PQ}) = \texttt{PQ}'$ which takes in a cipher text $\texttt{ct}$ and current priority queue and outputs $\texttt{PQ}'$ where the ciphertext is properly inserted into the priority queue according to the underlying plaintext of $\texttt{ct}$.

- $\texttt{PQFront}(\texttt{PQ}) = \texttt{ct}$ which takes as input a priority queue and outputs the ciphertext $\texttt{ct}$ who's plaintext is the minimum element in the queue.

- $\texttt{PQExtractFront}(\texttt{PQ}) = \texttt{ct}, \texttt{PQ}'$ which takes as input a priority queue and outputs the ciphertext with minimum plaintext as well as an updated queue without $\texttt{ct}$.

We also require that for N elements in $\texttt{PQ}$, operation $Q \in \{\texttt{PQInsert}, \texttt{PQFront}, \texttt{PQExtractFront}\}$, and input $\texttt{ct}$ into Q such that there is some simulator, Sim_{PQ} where

$$\{\text{Sim}_{\text{PQ}}(1^\lambda, N, Q, \texttt{PQ})\} \overset{c}{\approx} \{Q(\texttt{PQ}, \texttt{ct})\}. \tag{1}$$

In words, Sim_{PQ} can simulate a distribution of the output $\texttt{PQ}$ independent of the input element. LS: I am not quite sure about the above: definitions that I saw are for adaptive adversaries, but we are not adaptive.

3.2 Multi SLE Algorithm

For n parties, we want to output a list of k elements where each element is chosen randomly (without replacement) from a set of messages submitted by each party. Let the function for each party denote f_i and we will say that $f_i(\texttt{Enc}_{\text{TFHE}}(b_1), \texttt{Enc}_{\text{TFHE}}(b_2), ..., \texttt{Enc}_{\text{TFHE}}(b_n)) = y$ where $y \in \{\perp, 1, ..., k\}$ such that if $y = a$ then party i is the leader for the ath round and if not $y = \perp$.

To build this functionality, we will use an oblivious streaming sampler. We can construct one as follows, closely following [Shi20] but modifying randomness sampling to use a PRF.

- When an item, $\texttt{ct}$, arrives as a TFHE cipher text, we will produce a random value, $\texttt{Enc}_{\text{TFHE}}(\alpha)$ by evaluating $\texttt{PRF}(k, \texttt{ct})$ in FHE.
- If the item is the m-th item and $m \leq k$, insert the item into the queue along with α as its label via $\texttt{PQInsert}(\texttt{ct})$.
- If $m > k$, we will evaluate the PRF (in FHE) to get an encoded random coin which is 1 with probability $1/m$ and 0 otherwise. Then, run algorithm 4 which will
  - replace the smallest labeled item in the queue with the new item if the coin is 1.
  - not replace the smallest item in the queue if the coin is 0.
- At the end of the stream, we output all the items in the priority queue one by one using $\texttt{PQExtractFront}$.

The simulation proof follows almost exactly as in [Shi20] but now we have to deal with the randomness from the PRF rather than public randomness. We can follow the proof directly but add one more hybrid where we replace the output of the PRF with a random oracle in the FHE cipher text space. Then, we can use the PRF indistinguishably property to show that the hybrid is indistinguishable from the previous hybrid.

We can then implement multi secret leader election 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 one 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.

Algorithm 4 Oblivious Gated Insert

```
Input: $\texttt{Enc}_{\text{TFHE}}(\text{coin}), \texttt{PQ}, \texttt{ct}_a$
Output: PQ'
  $\texttt{ct}_b, \texttt{PQ}' \leftarrow \texttt{PQExtractFront}(\texttt{PQ})$
  $\texttt{ct}_{\text{new}} = \texttt{Enc}_{\text{TFHE}}(\text{coin}) \cdot \texttt{ct}_a + (\texttt{Enc}_{\text{TFHE}}(\text{coin}) - 1) \cdot \texttt{ct}_b$
  return $\texttt{PQInsert}(ct_{\text{new}})$
```

3.3 Correctness

For correctness, we want to show that for each of the k elections, a unique leader is chosen for each round and that the probability of being selected leader is uniform out of the potential leaders.

More formally, let E_i be the event in which party i is elected for some election and $E_{a,i}$ be the event which party i is elected for the a-th election. We want

$$\mathbf{Pr}\left[E_i\right] = \frac{k}{n}, \tag{2}$$

$$\mathbf{Pr}\left[E_{a,i} \mid E_i\right] = \frac{1}{k}, \tag{3}$$

$$\mathbf{Pr}\left[E_{b,i} \mid E_{a,i}\right] = 0. \tag{4}$$

where $b \neq a, b \in [k]$. LS: I think that here we can prove the uniqueness and fairness property of multi-leader election

Theorem 3.2 (Correctness)**.** *We have correctness...*

Proof. By definition... should be pretty easy to show. □

3.4 Simulation Security

We will show security in a semi-honest model via simulation. We will show that each user's view of the protocol can be simulated by a polynomial-time algorithm that only has access to the user's input. Note that view of publishing a message is the same as the view of as each user publishing a random LWE ciphertext assuming the TFHE is secure.

Then, upon evaluation, we need to show that the oblivious queue can be simulated. Note that every step in the oblivious queue algorithm which does not involve b_i involves manipulating TFHE samples under a secret key unknown to the simulator. Thus, we can simulate all steps not involving b_i. Every step in the oblivious queue algorithm which involves b_i and some b_j, for $i \neq j$ requires a homomorphic evaluation where one of the inputs in encrypted under a secret key unknown to the simulator. Thus, by the security of TFHE, the output of any of these steps is indistinguishable from a using a b_j encoding a different message.

Proof of Simulation Security

More formally, $\forall i \in [n]$, we can create a simulator, Sim_i, which takes as input the user's secret value b_i and publishes random TFHE encrypted message, $\mathtt{Enc}_{\mathrm{TFHE}}(b_j)$ for all $j \neq i$, for all other parties such that

$$\{\mathrm{Sim}_i(1^\lambda, s_i, \mathtt{comm}_i, r_i, f_i(\mathtt{Enc}_{\mathrm{TFHE}}(b_1), ..., \mathtt{Enc}_{\mathrm{TFHE}}(b_n))\} \overset{c}{\approx} \{\mathtt{view}_1^\pi(s_i, \mathtt{comm}_i, r_i, y)\} \tag{5}$$

with non-negligible probability.

To achieve eq. (5), the simulator simply follows a simple procedure

1. The simulator randomly samples b_j for all $j \neq i$
2. The simulator uses $\mathrm{Sim}_{\mathrm{PQ}}$ to simulate the oblivious streaming sampler for all inputs $\mathtt{Enc}_{\mathrm{TFHE}}(b_j)$ for $j \neq i$. For input $\mathtt{Enc}_{\mathrm{TFHE}}(b_i)$, the simulator honestly runs the oblivious streaming sampler.
3. After all n parties have submitted their inputs, the simulator runs the oblivious streaming sampler to get the outputs $\mathtt{Enc}_{\mathrm{TFHE}}(b_{a_1}), ...\mathtt{Enc}_{\mathrm{TFHE}}(b_{a_k})$.

4. If the output of the oblivious streaming sampler
5. If $y \neq \perp$, the simulator then "decrypts" $\mathtt{Enc}_{\mathrm{TFHE}}(b_{a_y})$ to get b_i and also pretends to decrypt the rest of the $\mathtt{Enc}_{\mathrm{TFHE}}(b_{a_j})$'s by setting the decryption to a random value. It $y = \perp$, then the simulator will "decrypt" all $\mathtt{Enc}_{\mathrm{TFHE}}(b_{a_j})$'s to random values.
6. The simulator simply outputs the view of the protocol with the "decryptions" of $\mathtt{Enc}_{\mathrm{TFHE}}(b_j)$'s.

We will now show that the above indeed is a polynomial time simulator with overwhelming probability.

Lemma 3.3. *The above algorithm satisfies the computational indistinguishably requirment of eq. (5).*

Proof. First we will show that $\mathcal{S}_i$ runs in polynomial time. There are only $k+1$ possible outcomes for y, each with inverse polynomial probability. Thus, with overwhelming probability, the simulator can sample the b_i's where the outcome is y.

We now show simulatability via hybrids. Note that by the fairness property, the simulator has a non-negligible probability of being the true winner of the election. Thus, we will assume that the simulator is the true winner of election y if $y \neq \perp$. If $y = \perp$, then we assume that i did not win any election. Moreover, we also have that the encoded FHE elements (where party i does not know the plaintext) are indistinguishable from encodings of random elements. We will now construct our hybrids:

$\mathbf{Hyb}_1$: Consider a hybrid where we replace the FHE encoded elements, $\mathtt{Enc}_{\mathrm{TFHE}}(b_j)$ for $j \neq i$ with random elements. By definition of IND-CPA security, this is indistinguishable from the original hybrid. We can then propogate the output of the oblivious priority queue to use these random elements.

$\mathbf{Hyb}_2$: Consider a hybrid where we replace the TFHE decoding of all elements which do not equal to y with a random value. As the b_j for $i \neq i$ are random, this hybrid is indistinguishable from the previous hybrid.

$\mathbf{Hyb}_3$: Consider a hybrid where we replace the the output of the priority queue with the output from simulating the priority queue with the random FHE encoded elements.

Note that $\mathbf{Hyb}_3$ is indistinguishable from the original hybrid and is the same as the output of the simulator. □

References

[Shi20] Elaine Shi. Path oblivious heap: Optimal and practical oblivious priority queue. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 842–858. IEEE, 2020. 3.2