Multiple Secret Leaders

Authors here

June 22, 2023

1 Some Notation or Smthng

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 shtuff 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.

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 $b_i = (\texttt{Enc}_{\text{TFHE}}(r_i), \texttt{Enc}_{\text{TFHE}}(\texttt{comm}_i))$.

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

Figure 1: Simple Sorting Evaluation

3. **Evaluation**: This is done by one party whom may even be an external party. See fig. 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 1 `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 2 `EvalStream`. Evaluates a stream of bids as they come in.

Input: L, set of $(\texttt{bid}, \texttt{level})$ pairs. New bid and level, (b, ℓ). Also, stop level ℓ_{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 2 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 2 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.

   Remark 2.1. I may have an off by one error here.

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

References