Single Secret Leader Election

Dan Boneh   Saba Eskandarian   Lucjan Hanzlik   Nicola Greco
What is Single Secret Leader Election?

A group of participants want to randomly choose exactly one leader, such that:

1. Identity of the leader is known only to the leader and nobody else

2. Leader can later publicly prove that she is the leader

Should work even if many registered participants don’t send messages.
What is Single Secret Leader Election?

A group of participants want to randomly choose exactly one leader, such that:

1. Identity of the leader is known only to the leader and nobody else

2. Leader can later publicly prove that she is the leader

Should work even if many registered participants don’t send messages.
Applications of SSLE - PoS Blockchains

Need leader to submit blocks

Publicizing leader ahead of time makes the whole protocol vulnerable
Applications of SSLE - PoS Blockchains

Secret Single-Leader Election (SSLE)
Applications of SSLE - PoS Blockchains

Secret Single-Leader Election (SSLE)

Single Secret Leader Election (SSLE) strengthens Eth2 proposers and committees against network-level DoS (and other adaptive attacks). We’re seriously considering the "DDH and Shuffling" construction in section 6 of eprint.iacr.org/2020/025.pdf

7:40 AM - 10 Jan 2020
A Non-Example

Common approach:

1. Everyone picks a random point on number line
A Non-Example

Common approach:

1. Everyone picks a random point on number line
2. Randomness beacon picks a random point on number line
A Non-Example

Common approach:

1. Everyone picks a random point on number line
2. Randomness beacon picks a random point on number line
3. Whoever is closest to the beacon wins
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in F_p$
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in F_p$
3. Any participant with $|R - v_i| < 10 \times 2^{\lambda} / N$ decommits to $v_i$
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in F_p$
3. Any participant with $|R - v_i| < 10 \times 2^\lambda / N$ decommits to $v_i$
4. Winner is participant with minimum $|R - v_i|$
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in F_p$
3. Any participant with $|R - v_i| < 10 \times 2^\lambda / N$ decommits to $v_i$
4. Winner is participant with minimum $|R - v_i|$

This is *almost* what we want.
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in \mathbb{F}_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in \mathbb{F}_p$
3. Any participant with $|R - v_i| < 10 \times 2^\lambda/N$ decommits to $v_i$
4. Winner is participant with minimum $|R - v_i|$

This is almost what we want.

Only the single leader publishes $v_i$ in expectation.
A Non-Example

Setup:
1. Choose $\lambda$-bit prime $p$
2. Randomness beacon that outputs $R \in F_p$

Election:
1. Each participant $i$ picks a secret $v_i$, produces commitment $\text{com}(v_i)$
2. Beacon produces $R \in F_p$
3. Any participant with $|R - v_i| < 10 * 2^\lambda / N$ decommits to $v_i$
4. Winner is participant with minimum $|R - v_i|$

This is *almost* what we want.

Only the single leader publishes $v_i$, *in expectation*
Why **Single** Secret Leader Election?

Having multiple potential leaders wastes effort and impedes consensus

From Protocol Labs RFC:

- Fork grinding
- Faster convergence
- Simpler protocol

Cost: requires a registration step
What Makes SSLE Challenging?

Want to minimize long-term storage
What Makes SSLE Challenging?

Want to minimize long-term storage

Want to minimize communication
What Makes SSLE Challenging?

Want to minimize long-term storage

Want to minimize communication

Want to minimize computation
What Makes SSLE Challenging?

Want to minimize long-term storage

Want to minimize communication

Want to minimize computation

Can’t expect every participant to send messages
What Makes SSLE Challenging?

Want to minimize long-term storage

Want to minimize communication

Want to minimize computation

Can’t expect every participant to send messages

Can’t expect every participant to stay online between rounds
Outline

Introduction

Formalizing SSLE

3 SSLE Constructions:
  - From DDH & Shuffling
  - From obfuscation
  - From tFHE
SSLE Requirements

Three security properties:

1. **Uniqueness**: only one leader is chosen by the election
2. **Unpredictability**: non-winners cannot guess who the winner is
3. **Fairness**: each user has 1/N chance of becoming the leader

Goal: *robust* election where DoS of c/N users disrupts election with probability c/N
SSLE Requirements

Three security properties:

1. **Uniqueness**: only one leader is chosen by the election
2. **Unpredictability**: non-winners cannot guess who the winner is
3. **Fairness**: each user has 1/N chance of becoming the leader

Goal: robust election where DoS of c/N users disrupts election with probability c/N

Our focus will be on the elections, not on using them to build blockchains.
SSLE Syntax

All algorithms assume access to public state $st$

Elections have access to randomness beacon output $R$
SSLE Syntax

All algorithms assume access to public state $st$

Elections have access to randomness beacon output $R$

SSLE Algorithms

1. Setup
2. Registration
3. Registration verification
4. Election
5. Election verification
Formalizing Definitions

**Adversary**

Choose set $M \subseteq [N], |M| = c$

**Setup**

Run setup $\rightarrow pp, st_0, sk_1, \ldots, sk_N$ (if applicable)

**Challenger**

$pp, st_0, \{sk_i\}_{i \in M}$
Formalizing Definitions

**Adversary**

**Setup**
Choose set $M \subseteq [N]$, $|M| = c$

$\leftarrow pp, st_0, \{sk_i\}_{i \in M}$

**Elections**
Register any users

**Challenger**

Run setup $\rightarrow pp, st_0, sk_1, \ldots, sk_N$ (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.
Formalizing Definitions

**Adversary**

**Setup**
Choose set \(M \subseteq [N], |M| = c\)

\(pp, st_0, \{sk_i\}_{i \in M}\)

**Elections**
Register any users

Run an election

(if uncorrupted winner)
Winner index \(i\), proof \(\pi_i\)

**Challenger**

Run setup→\(pp, st_0, sk_1, ..., sk_N\) (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.
Formalizing Definitions

**Adversary**

<table>
<thead>
<tr>
<th>Setup</th>
<th></th>
<th>Challenger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose set $M \subseteq [N],</td>
<td>M</td>
<td>=c$</td>
</tr>
<tr>
<td>[ pp, st_0, {sk}_{i \in M} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ Elections ]</td>
<td>Run an election</td>
<td>(if uncorrupted winner)</td>
</tr>
<tr>
<td>Register any users</td>
<td></td>
<td>Winner index $i$, proof $\pi_i$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Formalizing Definitions

**Adversary**

**Setup**

Choose set $M \subseteq [N]$, $|M| = c$

$pp, st_0, \{sk_i\}_{i \in M}$

**Elections**

Register any users

Run an election

(if uncorrupted winner)

Winner index $i$, proof $\pi_i$

**Challenge**

**Challenger**

Run setup $\rightarrow pp, st_0, sk_1, \ldots, sk_N$ (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.
Formalizing Definitions

**Adversary**

**Setup**
Choose set $M \subseteq [N]$, $|M| = c$

$pp, st_{0^0}, \{sk\}_{i \in M}$

**Elections**
Register any users
Run an election
(if uncorrupted winner)
Winner index $i$, proof $\pi_i$

**Challenge**

(j, $\pi_j$) for $j \in M$, for each election in election phase

**Challenger**

Run setup→$pp, st_{0^0}, sk_1, \ldots, sk_N$ (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.

Uniqueness

Output 1 if for any election, there is more than one tuple $(k, \pi_j)$ for which election verification accepts.
Formalizing Definitions

**Adversary**

**Setup**
Choose set $M \subseteq [N]$, $|M| = c$

$pp, st_0, \{sk_i\}_{i \in M}$

**Elections**
Register any users

Run an election

(if uncorrupted winner)
Winner index $i$, proof $\pi_i$

**Challenge**
Run one last election
Guess winner is user $i \in [N]$

**Challenger**

Run setup $\rightarrow pp, st_0, sk_1, \ldots, sk_N$ (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.

If winner is not in $[N] \setminus M$, output 0.
Otherwise, if winner is user $i$, output 1.
Secure if challenger never outputs 1 with probability greater than $1/(N-c)$. 

Unpredictability
Formalizing Definitions

**Adversary**

**Setup**
Choose set $M \subseteq [N], |M| = c$

$pp, st_0, \{sk_i\}_{i \in M}$

**Elections**
Register any users

Run an election

(if uncorrupted winner)
Winner index $i$, proof $\pi_i$

**Challenge**

Run one last election

**Challenger**

Run setup $\rightarrow pp, st_0, sk_1, \ldots, sk_N$ (if applicable)

Run registration verification for each uncorrupted user. Output 0 if any fails.

If winner is not in $[N] \setminus M$, output 1.

Secure if challenger never outputs 1 with probability greater than $c/N$. 
Three Constructions of SSLE

**Obfuscation**
Ideal solution, but uses theoretical tools

**tFHE**
Closer to realistic, only gives a threshold version of security

**DDH**
“Compromise” solution -- $\sqrt{N}$ communication per election, $1/(\sqrt{N-c})$ unpredictability
Should be suitable for practical use cases
Three Constructions of SSLE

**DDH**
“Compromise” solution -- $\sqrt{N}$ communication per election, $1/(\sqrt{N}-c)$ unpredictability
Should be suitable for practical use cases

**Obfuscation**
Ideal solution, but uses theoretical tools

**tFHE**
Closer to realistic, only gives a threshold version of security
SSLE from DDH

The easiest single non-secret leader election

User 1  User 1  User 1
User 2  User 2  User 2
User 3  User 3  User 3
User 4  User 4  User 4
User 5  User 5  User 5

Registration  Election  $R \in [N]$
SSLE from DDH

The easiest single non-secret leader election

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Registration**

**Election** $R \in [N]$

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How to hide the leader?

- **Uniqueness**
- **Fairness**
- **Unpredictability**
SSLE from DDH

1. Commitments

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>Registration</th>
</tr>
</thead>
</table>

Election $R \in [N]$

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
</tr>
</thead>
</table>

- ✔️ Uniqueness
- ✔️ Fairness
- ✗ Unpredictability
SSLE from DDH

1. Commitments

\[
\begin{array}{c}
\text{com}(u_1) \\
\text{com}(u_2) \\
\text{com}(u_3) \\
\text{com}(u_4)
\end{array}
\quad \xrightarrow{\text{Registration}} \quad
\begin{array}{c}
\text{com}(u_1) \\
\text{com}(u_2) \\
\text{com}(u_3) \\
\text{com}(u_4) \\
\text{com}(u_5)
\end{array}
\quad \xrightarrow{\text{Election } R \in [N]} \quad
\begin{array}{c}
\text{com}(u_1) \\
\text{com}(u_2) \\
\text{com}(u_3) \\
\text{\color{green!50!black!50!white}com}(u_4) \\
\text{com}(u_5)
\end{array}
\]

- ✔️ Uniqueness
- ✔️ Fairness
- ✗ Unpredictability
SSLE from DDH

1. Commitments
2. Shuffling

<table>
<thead>
<tr>
<th>Registration</th>
<th>Election $R \in [N]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>com(u1)</td>
<td>com(u1)</td>
</tr>
<tr>
<td>com(u2)</td>
<td>com(u2)</td>
</tr>
<tr>
<td>com(u3)</td>
<td>com(u3)</td>
</tr>
<tr>
<td>com(u4)</td>
<td>com(u4)</td>
</tr>
<tr>
<td></td>
<td>com(u5)</td>
</tr>
</tbody>
</table>

✅ Uniqueness
✅ Fairness
❌ Unpredictability
SSLE from DDH

1. Commitments
2. Shuffling

✅ Uniqueness
✅ Fairness
❌ Unpredictability
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization

- Uniqueness
- Fairness
- Unpredictability

Registration Part 1:
- \( \text{com(u1)} \)
- \( \text{com(u2)} \)
- \( \text{com(u3)} \)
- \( \text{com(u4)} \)

Registration Part 2:
- \( \text{com(u1)} \)
- \( \text{com(u2)} \)
- \( \text{com(u3)} \)
- \( \text{com(u4)} \)
- \( \text{com(u5)} \)

Election:
- \( R \in [N] \)
- \( \text{com(u2)'} \)
- \( \text{com(u5)'} \)
- \( \text{com(u4)'} \)
- \( \text{com(u1)'} \)
- \( \text{com(u3)'} \)
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization & Reidentification

\[
\text{Registration Part 1:} \\
\text{Registration Part 2:} \\
\text{Election: } R \in [N]
\]
A Rerandomizable & Reidentifiable Commitment

Let $g \in G$, $G$ is a group where DDH is hard

$\text{Com}(k, r) \rightarrow (g^r, g^{rk})$
A Rerandomizable & Reidentifiable Commitment

Let \( g \in G \), \( G \) is a group where DDH is hard

\[
\text{Com}(k, r) \rightarrow (g^r, g^{rk})
\]

Rerandomization: \((g^r, g^{rk}) \rightarrow (g^{rr'}, g^{rr'k})\)

Reidentification: given \((u, v)\), check if \( u^k = v \)
A Rerandomizable & Reidentifiable Commitment

Let $g \in G$, $G$ is a group where DDH is hard

Com($k, r$) $\rightarrow$ ($g^r, g^{rk}$)

Rerandomization: ($g^r, g^{rk}$) $\rightarrow$ ($g^{rr'}, g^{rr'k}$)

Reidentification: given ($u, v$), check if $u^k = v$

Security follows from DDH: ($g^r, g^{rk}, g^{rr'}, g^{rr'k}$) vs ($g^r, g^{rk}, g^{rr'}, g^{rz}$)
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization & Reidentification
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization & Reidentification
4. Verification of shuffle

\[ \text{com}(u_1) \quad \text{com}(u_2) \quad \text{com}(u_3) \quad \text{com}(u_4) \]

Registration Part 1

\[ \text{Registration Part 2} \quad \text{Election} \]

\[ R \in [N] \]

\[ \text{com}(u_2)' \quad \text{com}(u_5)' \quad \text{com}(u_4)' \quad \text{com}(u_1)' \quad \text{com}(u_3)' \]

\[ \text{com}(u_2)' \quad \text{com}(u_5)' \quad \text{com}(u_4)' \quad \text{com}(u_1)' \quad \text{com}(u_3)' \]
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization & Reidentification
4. Verification of shuffle -- NIZK or other users check

Registration Part 1

\[
\text{com(u1)} \quad \text{com(u2)} \quad \text{com(u3)} \quad \text{com(u4)}
\]

Registration Part 2

\[
\text{com(u1)} \quad \text{com(u2)} \quad \text{com(u3)} \quad \text{com(u4)} \quad \text{com(u5)}
\]

Election

\[
R \in \mathbb{N}
\]

\[
\begin{array}{l}
\text{com(u2)'} \\
\text{com(u5)'} \\
\text{com(u4)'} \\
\text{com(u1)'} \\
\text{com(u3)'}
\end{array}
\]
SSLE from DDH

1. Commitments
2. Shuffling
3. Rerandomization & Reidentification
4. Verification of shuffle -- NIZK or other users check
5. Defend against duplication attacks

\[
\text{com}(u_1) \quad \text{com}(u_2) \quad \text{com}(u_3) \quad \text{com}(u_4)
\]

\[
\text{Registration Part 1}
\]

\[
\begin{array}{c}
\text{com}(u_1) \\
\text{com}(u_2) \\
\text{com}(u_3) \\
\text{com}(u_4) \\
\end{array}
\]

\[
\text{Registration Part 2}
\]

\[
\begin{array}{c}
\text{com}(u_1) \\
\text{com}(u_2) \\
\text{com}(u_3) \\
\text{com}(u_4) \\
\text{com}(u_5) \\
\end{array}
\]

\[
\text{Election } R \in [N]
\]

\[
\begin{array}{c}
\text{com}(u_2)' \\
\text{com}(u_5)' \\
\text{com}(u_4)' \\
\text{com}(u_1)' \\
\text{com}(u_3)'
\end{array}
\]
Duplication Attack

Duplication attack makes it possible for 2 different users to register with a commitment to the same value

Breaks uniqueness and unpredictability
Preventing Duplication Attacks

How to ensure that users never commit to the same value?

Idea: Derive a secret commitment value and a tag from a master secret

Sample random $k$

$$H(k) \rightarrow k_L, k_R$$

Post $\text{com}(k_L)$ and $k_R$

Registrations to the same secret detected by duplicate $k_R$

(H modeled as random oracle)
Saving Communication

Protocol thus far has required linear communication for each registration

\[
\text{com}(k_{1L}) \quad \text{com}(k_{2L}) \quad \text{com}(k_{3L}) \quad \text{com}(k_{4L}) \quad \text{com}(k_{5L})
\]

Registration Part 1

\[
\text{com}(k_{1L})' \quad \text{com}(k_{2L})' \quad \text{com}(k_{3L})' \quad \text{com}(k_{4L})' \quad \text{com}(k_{5L})'
\]

Registration Part 2

Election \( R \in [N] \)

\[
\text{com}(k_{1L})' \quad \text{com}(k_{2L})' \quad \text{com}(k_{3L})' \quad \text{com}(k_{4L})' \quad \text{com}(k_{5L})'
\]
Saving Communication

Protocol thus far has required linear communication for each registration

\[
\begin{align*}
\text{com}(k_{1L}) & \\
\text{com}(k_{2L}) & \\
\text{com}(k_{3L}) & \\
\text{com}(k_{4L}) & \\
\text{com}(k_{5L}) & \\
\end{align*}
\]

\[
\begin{align*}
\text{com}(k_{1L})' & \\
\text{com}(k_{2L})' & \\
\text{com}(k_{3L})' & \\
\text{com}(k_{4L})' & \\
\text{com}(k_{5L})' & \\
\end{align*}
\]

Communicating all this is expensive
Saving Communication

Communication/Security tradeoff: instead of shuffling new entry into the whole list, split the list into a number of buckets and only shuffle into one bucket.
Saving Communication

Communication/Security tradeoff: instead of shuffling new entry into the whole list, split the list into a number of buckets and only shuffle into one bucket.

\[
\begin{align*}
\text{com}(k_{1L}) \\
\text{com}(k_{2L}) \\
\text{com}(k_{3L}) \\
\vdots \\
\text{com}(k_{NL})
\end{align*}
\]

\[
\begin{align*}
\text{com}(k_{1L}) \\
\ldots \\
\text{com}(k_{\sqrt{N},L}) \\
\text{com}(k_{\sqrt{N+1},L}) \\
\ldots \\
\text{com}(k_{N-\sqrt{N+1},L}) \\
\text{com}(k_{N,L})
\end{align*}
\]
Saving Communication

Communication/Security tradeoff: instead of shuffling new entry into the whole list, split the list into a number of buckets and only shuffle into one bucket.

Larger buckets mean more unpredictability but also more communication

$\sqrt{N}$ sized buckets seems like a good tradeoff
Security

With a deterministic choice of buckets, we get the following theorem:

**Theorem 19.** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A,$ \textit{SSSLE} is a unique, fair, and $\frac{1}{\sqrt{N-c}}$-unpredictable SSLE scheme in the random oracle model.
Security

With a deterministic choice of buckets, we get the following theorem:

**Theorem 19.** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSLE is a unique, fair, and $\frac{1}{\sqrt{N-c}}$-unpredictable SSLE scheme in the random oracle model.

We can do better by randomizing the choice of buckets, so an adversary needs to corrupt $O(N)$ users to guess winner with constant probability.
Security

With a deterministic choice of buckets, we get the following theorem:

Theorem 19. Assuming that $\mathbb{G}$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSL is a unique, fair, and $\frac{1}{\sqrt{N-c}}$-unpredictable SSLE scheme in the random oracle model.

We can do better by randomizing the choice of buckets, so an adversary needs to corrupt $O(N)$ users to guess winner with constant probability

Theorem 20 (Informal). Assuming that $\mathbb{G}$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSL modified to assign buckets randomly at user registration time is a unique, fair, and $\frac{1}{\sqrt{N-c} \sqrt{2 \lambda (N-c)}}$-unpredictable SSLE scheme in the random oracle model.
Security

With a deterministic choice of buckets, we get the following theorem:

**Theorem 19.** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSLE is a unique, fair, and $\frac{1}{\sqrt{N-c}}$-unpredictable SSLE scheme in the random oracle model.

We can do better by randomizing the choice of buckets, so an adversary needs to corrupt $O(N)$ users to guess winner with constant probability

**Theorem 20 (Informal).** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSLE modified to assign buckets randomly at user registration time is a unique, fair, and $\frac{1}{\sqrt{N-c}} - \sqrt{\frac{2\lambda(N-c)}{\sqrt{N}}}$-unpredictable SSLE scheme in the random oracle model.

Open problem: we believe we can do better with a more clever shuffling/bucketing algorithm, e.g. by using something like a square shuffle [Hastad06]
With a deterministic choice of buckets, we get the following theorem:

**Theorem 19.** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSLE is a unique, fair, and $\frac{1}{\sqrt{N-c}}$-unpredictable SSLE scheme in the random oracle model.

We can do better by randomizing the choice of buckets, so an adversary needs to corrupt $O(N)$ users to guess winner with constant probability

**Theorem 20 (Informal).** Assuming that $G$ is a group in which the DDH problem is hard, then for any adversary $A$, SSSLE modified to assign buckets randomly at user registration time is a unique, fair, and

$$\frac{1}{\sqrt{N-c}} \cdot \frac{1}{\sqrt{2\lambda(N-c)}}$$-unpredictable SSLE scheme in the random oracle model.

Open problem: we believe we can do better with a more clever shuffling/bucketing algorithm, e.g. by using something like a square shuffle [Hastad06]

Open problem: constant communication per election (in a practical scheme)
SSLE from Obfuscation

Obfuscation \cite{BGI+01,GGH+13}

Obfuscator $iO(C)$ produces a new circuit $C'$ such that:

1. $C$ and $C'$ have the exact same behavior.
2. For any two circuits $C_0$, $C_1$ that have the exact same behavior, no adversary can distinguish between $iO(C_0)$ and $iO(C_1)$. 
SSLE from Obfuscation

**Obfuscation** [BGI+01, GGH+13]
Obfuscator $iO(C)$ produces a new circuit $C'$ such that:

1. $C$ and $C'$ have the exact same behavior
2. For any two circuits $C_0$, $C_1$ that have the exact same behavior, no adversary can distinguish between $iO(C_0)$ and $iO(C_1)$

**Puncturable PRF** [BW13, BGI14, KPTZ13]
PRF where you can generate a *punctured* key that allows you to evaluate the PRF everywhere except at that point.

Given the punctured key, the value of the PRF at the punctured point is still pseudorandom.
SSLE from Obfuscation

Plan:

1. Write a program that picks leader using secret key embedded in the program
2. Obfuscate program during trusted setup and distribute to everyone
3. Any participant just needs to post a public key to register for elections
4. Obfuscated program output should allow leader to prove she won
SSLE from Obfuscation

Program to obfuscate, first attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]
SSLE from Obfuscation

Program to obfuscate, first attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( w \leftarrow F(k, s) \)
SSLE from Obfuscation

Program to obfuscate, first attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( w \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \text{ if } i = w \mod n, b \leftarrow 0 \text{ otherwise} \)
4. Output \( b \)
SSLE from Obfuscation

Program to obfuscate, first attempt

\[ P((pk_0, ..., pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, ..., pk_{N-1} \)
2. \( w \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. Output \( b \)

✅ Elects one leader randomly based on secret key

❌ Anyone can learn the leader by trying each value of \( i \)
SSLE from Obfuscation

Program to obfuscate, second attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( w \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \text{ if } i = w \mod n, \ b \leftarrow 0 \text{ otherwise} \)
4. 

SSLE from Obfuscation

Program to obfuscate, second attempt

\[ P((pk_0, ..., pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, ..., pk_{N-1} \)
2. \((w, r) \leftarrow F(k, s)\)
3. \( b \leftarrow 1 \text{ if } i = w \mod n, b \leftarrow 0 \text{ otherwise} \)
4.
SSLE from Obfuscation

Program to obfuscate, second attempt

\[ P((pk_0, ..., pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, ..., pk_{N-1} \)
2. \( (w, r) \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( ct \leftarrow \text{Encrypt}(pk_r, b; r) \)
5. Output \( ct \)
SSLE from Obfuscation

Program to obfuscate, second attempt

\[ P((pk_0, ..., pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, ..., pk_{N-1} \)
2. \((w, r) \leftarrow F(k, s)\)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( ct \leftarrow Encrypt(pk_i, b; r) \)
5. Output \( ct \)

✅ Elects one leader randomly based on secret key

✅ Only user \( i \) can decrypt \( b_i \)

❌ Not clear how winner can prove that she won the election
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R) : \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( (w, r) \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. 
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( (w,r,r') \leftarrow F(k,s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4.
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( (w,r,r') \leftarrow F(k,s) \)
3. \( b \leftarrow 1 \text{ if } i = w \mod n, b \leftarrow 0 \text{ otherwise} \)
4. \( c \leftarrow \text{com}(b; r) \)
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( (w,r,r') \leftarrow F(k,s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( c \leftarrow \text{com}(b; r) \)
5. \( ct \leftarrow \text{Encrypt}(pk_i, r; r') \)
6. Output \( c, ct \)
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, \ pk_0, \ldots, pk_{N-1} \)
2. \( (w,r,r') \leftarrow F(k,s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( c \leftarrow \text{com}(b; r) \)
5. \( ct \leftarrow \text{Encrypt}(pk_i, r; r') \)
6. Output \( c, ct \)

✅ Elects one leader randomly based on secret key
✅ Only user \( i \) can decrypt \( b_i \)
✅ Prove leadership by revealing \( r \)
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \((w, r, r') \leftarrow F(k, s)\)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( c \leftarrow \text{com}(b; r) \)
5. \( ct \leftarrow \text{Encrypt}(pk'_i, r; r') \)
6. Output \( c, ct \)

✅ Elects one leader randomly based on secret key
✅ Only user \( i \) can decrypt \( b_i \)
✅ Prove leadership by revealing \( r \)

Why not encrypt?
SSLE from Obfuscation

Program to obfuscate, final attempt

\[
P((pk_0, \ldots, pk_{N-1}), i, N, R):
\]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \((w,r,r') \leftarrow F(k, s)\)
3. \(b \leftarrow 1\) if \(i = w \mod n\), \(b \leftarrow 0\) otherwise
4. \(c \leftarrow \text{com}(b; r)\)
5. \(ct \leftarrow \text{Encrypt}(pk_i, r; r')\)
6. Output \(c, ct\)

✅ Elects one leader randomly based on secret key
✅ Only user \(i\) can decrypt \(b_i\)
✅ Prove leadership by revealing \(r\)

Why not encrypt?
If the encryption does not commit, adversary could potentially find bad randomness that allows a non-winning ciphertext to decrypt to 1
SSLE from Obfuscation

Program to obfuscate, final attempt

\[ P((pk_0, \ldots, pk_{N-1}), i, N, R): \]

1. \( s \leftarrow R, pk_0, \ldots, pk_{N-1} \)
2. \( (w, r, r') \leftarrow F(k, s) \)
3. \( b \leftarrow 1 \) if \( i = w \mod n \), \( b \leftarrow 0 \) otherwise
4. \( c \leftarrow \text{com}(b; r) \)
5. \( ct \leftarrow \text{Encrypt}(pk_i, r; r') \)
6. Output \( c, ct \)

✅ Elects one leader randomly based on secret key
✅ Only user \( i \) can decrypt \( b_i \)
✅ Prove leadership by revealing \( r \)

See paper for proofs of uniqueness, selective fairness, selective unpredictability
SSLE from tFHE

Reminder: why can’t we use a generic MPC protocol for SSLE?

   Easy DoS opportunity if everyone has to come back for a second round
SSLE from tFHE

Reminder: why can’t we use a generic MPC protocol for SSLE?

Easy DoS opportunity if everyone has to come back for a second round

What if only a few people have to come back and it doesn’t matter which ones?
SSLE from tFHE

Reminder: why can’t we use a generic MPC protocol for SSLE?

   Easy DoS opportunity if everyone has to come back for a second round

What if only a few people have to come back *and it doesn’t matter which ones*?

Tools from threshold crypto can enable this!
SSLE from tFHE

**Threshold Encryption:**
Standard public-key encryption, but instead of one secret key, many users have shares of a secret key that produce *partial decryptions*, with \( t \) partial decryptions needed to produce a plaintext.
SSLE from tFHE

Threshold Encryption:
Standard public-key encryption, but instead of one secret key, many users have shares of a secret key that produce partial decryptions, with $t$ partial decryptions needed to produce a plaintext.

Fully Homomorphic Encryption (FHE):
Standard public-key encryption, but ciphertexts can be added together and multiplied. Expensive operation is multiplication, high multiplicative depth is especially costly.
SSLE from tFHE

**Threshold Encryption:**
Standard public-key encryption, but instead of one secret key, many users have *shares* of a secret key that produce *partial decryptions*, with *t* partial decryptions needed to produce a plaintext.

**Fully Homomorphic Encryption (FHE):**
Standard public-key encryption, but ciphertexts can be added together and multiplied. Expensive operation is multiplication, high multiplicative depth is especially costly.

**Threshold FHE (tFHE):**
Combine the two tools above.
SSLE from tFHE

Threshold Encryption:
Standard public-key encryption, but instead of one secret key, many users have shares of a secret key that produce partial decryptions, with \( t \) partial decryptions needed to produce a plaintext.

Fully Homomorphic Encryption (FHE):
Standard public-key encryption, but ciphertexts can be added together and multiplied. Expensive operation is multiplication, high multiplicative depth is especially costly.

Threshold FHE (tFHE):
Combine the two tools above.

Using these tools, we can only really hope for threshold unpredictability and fairness
SSLE from tFHE

Plan:
1. All participants get a tFHE decryption key
2. Define a computation that picks the leader
3. Evaluate computation under tFHE
4. Some subset of $t$ users post partial decryptions
5. Output of computation somehow secretly determines winner
SSLE from tFHE

Plan:

1. All participants get a tFHE decryption key
2. Define a computation that picks the leader
3. Evaluate computation under tFHE
4. Some subset of $t$ users post partial decryptions
5. Output of computation somehow secretly determines winner

Unlike the obfuscation case, everyone gets the same output.
SSLE from tFHE

Idea:

Each participant registers with a secret $k$

Output of computation is the secret of a randomly chosen participant

The participant knows her secret, but nobody else knows who owns it
SSLE from tFHE

Idea:

Each participant registers with a secret $k$

Output of computation is the secret of a randomly chosen participant

The participant knows her secret, but nobody else knows who owns it

Main remaining problems to solve:

1. Efficiently generating randomness inside the tFHE
2. Efficiently using the randomness to pick someone’s secret
SSLE from tFHE

Idea:
Each participant registers with a secret k
Output of computation is the secret of a randomly chosen participant
The participant knows her secret, but nobody else knows who owns it

Main remaining problems to solve:
1. Efficiently generating randomness inside the tFHE
2. Efficiently using the randomness to pick someone’s secret

See paper for other details
SSLE from tFHE

\[ k_1 \]
\[ k_2 \]
\[ k_3 \]
\[ k_4 \]
\[ k_5 \]
SSLE from tFHE

\[ k_1, k_2, k_3, k_4, k_5 \]

Random weight-1 vector

\[ 0 \]
\[ 0 \]
\[ 0 \]
\[ 1 \]
\[ 0 \]

\[ = k_4 \]
SSLE from tFHE

How can we efficiently generate a random weight-1 vector given some random bits inside the tFHE?

“efficiently” = low multiplicative depth
SSLE from tFHE

1. Start with $\log N$ random bits
SSLE from tFHE

1. Start with $\log N$ random bits

2. Split bits into length-2 vectors
   where $b \rightarrow (b, 1-b)$:
   a. $0 \rightarrow (0,1)$
   b. $1 \rightarrow (1,0)$
SSLE from tFHE

1. Start with $\log N$ random bits

2. Split bits into length-2 vectors where $b \rightarrow (b, 1-b)$:
   a. $0 \rightarrow (0,1)$
   b. $1 \rightarrow (1,0)$

3. Take outer product of adjacent vectors and flatten
   a. E.g. $(0,1) \bowtie (1,0) = (0,0,1,0)$
SSLE from tFHE

1. Start with $\log N$ random bits

2. Split bits into length-2 vectors where $b \rightarrow (b, 1-b)$:
   a. $0 \rightarrow (0,1)$
   b. $1 \rightarrow (1,0)$

3. Take outer product of adjacent vectors and flatten
   a. E.g. $(0,1) \bowtie (1,0) = (0,0,1,0)$

4. Repeat step 3 until only a single length-$N$ vector remains

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>(1,0)</td>
<td>(1,0)</td>
<td>(0,1)</td>
</tr>
<tr>
<td>(0,0,1,0)</td>
<td>(0,0,0,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,0,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,0)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SSLE from tFHE

1. Start with \( \log N \) random bits

2. Split bits into length-2 vectors where \( b \rightarrow (b, 1-b) \):
   a. \( 0 \rightarrow (0,1) \)
   b. \( 1 \rightarrow (1,0) \)

3. Take outer product of adjacent vectors and flatten
   a. E.g. \((0,1) \otimes (1,0) = (0,0,1,0)\)

4. Repeat step 3 until only a single length-\( N \) vector remains

Multiplicative depth: \( \log \log N \)
Single Secret Leader Election

Elect exactly 1 leader such that only the leader learns who she is and can prove it.

Our contributions:

Formalization of SSLE requirements and security definitions

Three constructions: from DDH, tFHE, and obfuscation
Single Secret Leader Election

Elect exactly 1 leader such that only the leader learns who she is and can prove it.

Our contributions:

- Formalization of SSLE requirements and security definitions
- Three constructions: from DDH, tFHE, and obfuscation


Contact: saba@cs.stanford.edu