An Optimal Distributed Discrete Log Protocol with Applications to Homomorphic Secret Sharing

Itai Dinur\textsuperscript{1}, Nathan Keller\textsuperscript{2} and Ohad Klein\textsuperscript{2}

\textsuperscript{1} Department of Computer Science, Ben-Gurion University, Israel
\textsuperscript{2} Department of Mathematics, Bar-Ilan University, Israel

August 22, 2018
The Spaceships Problem

Spaceships land on adjacent cells of random numbers array.
Cannot communicate.
Allowed to read $T$ cells.
Must eventually stop.
Goal: Stop on the same cell with high probability.
Do not know who is on the left.

Main Problem
How can the spaceships maximize their meeting probability?
What is this highest probability (depending on $T$)?

Ohad Klein (BIU)
The Spaceships Problem

Spaceships land on adjacent cells of random numbers array. Cannot communicate. Allowed to read $T$ cells. Must eventually stop. Goal: Stop on the same cell with high probability.

Main Problem
How can the spaceships maximize their meeting probability? What is this highest probability (depending on $T$)?

Ohad Klein (BIU)
Spaceships land on adjacent cells of random numbers array.
The Spaceships Problem

- Spaceships land on **adjacent** cells of random numbers array.
- **Cannot communicate**.
- Allowed to read $T$ cells.

**Goal:** Stop on the same cell with high probability.

**Main Problem**

How can the spaceships maximize their meeting probability? What is this highest probability (depending on $T$)?

Ohad Klein (BIU)
The Spaceships Problem

- Spaceships land on **adjacent** cells of random numbers array.
- **Cannot communicate.**
- Allowed to read $T$ cells.
- Must eventually stop.
- Goal: Stop on the **same cell** with high probability.

0 5 4 0 2 4 6 1 6 8 7 5
The Spaceships Problem

- Spaceships land on **adjacent** cells of random numbers array.
- **Cannot communicate.**
- Allowed to read $T$ cells.
- Must eventually stop.
- Goal: Stop on the **same cell** with high probability.
- Do not know who is on the left.
The Spaceships Problem

- Spaceships land on **adjacent** cells of random numbers array.
- **Cannot communicate.**
- Allowed to read $T$ cells.
- Must eventually stop.
- Goal: Stop on the **same cell** with high probability.
- Do not know who is on the left.

### Main Problem

- How can the spaceships maximize their meeting probability?
- What is this highest probability (depending on $T$)?
Algorithm: Basic

1: function \textsc{Basic}(array, start, T) \\
2: \hspace{1em} return \text{arg \, min}_{i \in [start, start+T)} \{ array[i] \};
Algorithm: Basic

1: function Basic(array, start, T)
2:     return arg min_{i \in [start, start + T)} \{ array[i] \};

Analysis

Alice & Bob fail to synchronize iff minimum in on one of the ends.
Probability = 2/(T + 1).
Homomorphic Secret Sharing

- **Homomorphic Secret Sharing** – introduced by Boyle, Gilboa, Ishai [BGI] (CRYPTO’16) as a more practical alternative to FHE (Fully-Homomorphic-Encryption).

- Suppose we wish to securely compute in the cloud.

- HSS enables to distribute the evaluation of a **public function** $f$ on a **secret input** $x$ among two servers, each receiving a secret share $y$ or $z$, so that $f(x)$ can easily be recovered from $f'(y)$ and $f'(z)$.

  Each of $y$ and $z$ computationally hides $x$.

  'Share' and 'Join' are cheap.
Homomorphic Secret Sharing

- **Homomorphic Secret Sharing** – introduced by Boyle, Gilboa, Ishai [BGI] (CRYPTO’16) as a more practical alternative to FHE (Fully-Homomorphic-Encryption).

- Suppose we wish to securely compute in the cloud.

HSS enables to distribute the evaluation of a **public function** \( f \) on a **secret input** \( x \) among two servers, each receiving a secret share \( y \) or \( z \), so that \( f(x) \) can easily be recovered from \( f'(y) \) and \( f'(z) \).
  - Each of \( y \) and \( z \) computationally hides \( x \).

  ![Diagram](https://via.placeholder.com/150)

- ‘Share’ and ‘Join’ are cheap.
BGI constructed a group based HSS protocol.

- Security Relies **only on DDH** (Decisional-Diffie-Hellman) Hardness assumption.
- Low communication complexity.
- Applicable only for functions $f$ inside the class of ‘branching programs’.

Applications:

- **PIR**: Private information retrieval (with branching program predicate).
- **SMPC**: Secure multi-party computation in sublinear communication (leveled circuits).

Requires an algorithm solving the Distributed-Discrete-Log problem (**DDLOG**).
BGI constructed a group based HSS protocol.
- Security Relies **only on DDH** (Decisional-Diffie-Hellman) Hardness assumption.
- **Low communication complexity**.
- **Applicable only for** functions $f$ inside the class of ‘**branching programs**’.

**Applications**
- **PIR**: Private information retrieval (with branching program predicate).
- **SMPC**: Secure multi-party computation in sublinear communication (leveled circuits).
BGI constructed a group based HSS protocol.
- Security Relies only on DDH (Decisional-Diffie-Hellman) Hardness assumption.
- Low communication complexity.
- Applicable only for functions $f$ inside the class of ‘branching programs’.

Applications
- **PIR**: Private information retrieval (with branching program predicate).
- **SMPC**: Secure multi-party computation in sublinear communication (leveled circuits).

Requires an algorithm solving the Distributed-Discrete-Log problem (**DDLOG**).
Let $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$ be a secret input.

We wish to compute $f(x) \in \mathbb{Z}$, where $f$ contains instructions of the form:

1) $v_i \leftarrow v_j \pm v_k$ for variables $v_i, v_j, v_k \in \mathbb{Z}$.
2) $v_i \leftarrow v_j \cdot x_k$ where $x_k$ is an input bit.
3) $v_i \leftarrow x_k$. 

Share: Let $G$ be a cryptographic group generated by $g$. Choose random $y_i, z_i$ with $x_i = y_i + z_i$, and share $y_i, z_i$ respectively. Also, publish $g^{x_i}$. (Actually, use something similar to El-Gamal.)

Evaluation of $f(x)'$ almost identical to evaluation of $f(x)$.

We maintain that at any time $t$, $v_t(x) = v_t(y) + v_t(z)$. 

Trivial for instructions 1 & 3. What about 2?
Overview of BGI’s HSS Protocol (1)

- Let \( x = (x_1, \ldots, x_n) \in \{0, 1\}^n \) be a secret input.

- We wish to compute \( f(x) \in \mathbb{Z} \), where \( f \) contains instructions of the form:
  1) \( v_i \leftarrow v_j \pm v_k \) for variables \( v_i, v_j, v_k \in \mathbb{Z} \).
  2) \( v_i \leftarrow v_j \cdot x_k \) where \( x_k \) is an input bit.
  3) \( v_i \leftarrow x_k \).

- **Share \( x \):**
  - Let \( G \) be a cryptographic group generated by \( g \).
  - Choose random \( y_i, z_i \) with \( x_i = y_i + z_i \), and share \( y_i, z_i \) respectively.
  - Also, publish \( g^{x_i} \). (Actually, use something similar to El-Gamal.)
Overview of BGI’s HSS Protocol (1)

- Let \( x = (x_1, \ldots, x_n) \in \{0, 1\}^n \) be a secret input.

- We wish to compute \( f(x) \in \mathbb{Z} \), where \( f \) contains instructions of the form:
  1) \( v_i \leftarrow v_j \pm v_k \) for variables \( v_i, v_j, v_k \in \mathbb{Z} \).
  2) \( v_i \leftarrow v_j \cdot x_k \) where \( x_k \) is an input bit.
  3) \( v_i \leftarrow x_k \).

**SHARE \( x \):**
- Let \( G \) be a cryptographic group generated by \( g \).
- Choose random \( y_i, z_i \) with \( x_i = y_i + z_i \), and **share** \( y_i, z_i \) respectively.
- Also, **publish** \( g^{x_i} \). (Actually, use something similar to El-Gamal.)

**Evaluation of \( f' \) almost identical to evaluation of \( f \).**
- We maintain that at any time \( t \), \( v_i^t(x) = v_i^t(y) + v_i^t(z) \).
  - Trivial for instructions 1 & 3. What about 2?
Overview of BGI’s HSS Protocol (2)

- How would we implement \( v_i \leftarrow v_j \cdot x_k \)?
- We only have \( v_j \) and \( g^{x_k} \)!
Overview of BGI’s HSS Protocol (2)

- How would we implement $v_i \leftarrow v_j \cdot x_k$?
- We only have $v_j$ and $g^{x_k}$!
- We can compute $g^{v_i} = g^{v_j \cdot x_k}$, but we do not have $v_i$.
- We seek for an efficient probabilistic algorithm $A : G \rightarrow \mathbb{Z}/|G|$ satisfying:

**Simplified DDLOG problem**

Let $u, v \in \{0, 1\}$, then

$$\Pr [A(g^u) + A(g^v) \neq u + v] \leq \delta,$$

for a minimal $\delta > 0$, depending on the complexity of $A$. 
Overview of BGI’s HSS Protocol (2)

- How would we implement $v_i \leftarrow v_j \cdot x_k$?
- We only have $v_j$ and $g^{x_k}$!
- We can compute $g^{v_i} = g^{v_j x_k}$, but we do not have $v_i$.
- We seek for an efficient probabilistic algorithm $A: G \rightarrow \mathbb{Z}/|G|$ satisfying:

**Simplified DDLOG problem**

Let $u, v \in \{0, 1\}$, then

$$\text{Pr} [A(g^u) + A(g^v) \neq u + v] \leq \delta,$$

for a minimal $\delta > 0$, depending on the complexity of $A$.

- Can binary expand $v_j$ and assume $v_j \in \{0, 1\}$.
- Degenerate formulation due to usage $t \mapsto g^t$, instead of El-Gamal.
Let $G$ be a cyclic cryptographic group, with a generator $g$. Find probabilistic algorithms $A, B : G \to \mathbb{Z}/|G|$, so that

$$\forall x \in \mathbb{Z} : \Pr[A(g^{x+1}) - B(g^x) \neq 1] \leq \delta,$$

for a minimal $\delta > 0$, depending on the time complexity of $A, B$. 

Apply a PRF on $g^t$ to randomize.

BGI used 'Basic' algorithm, achieving $\delta = 1/T$ for running time $O(T)$. 

Ohad Klein (BIU)
Let $G$ be a cyclic cryptographic group, with a generator $g$. Find probabilistic algorithms $A, B : G \rightarrow \mathbb{Z}/|G|$, so that

$$\forall x \in \mathbb{Z} : \Pr[A(g^{x+1}) - B(g^x) \neq 1] \leq \delta,$$

for a minimal $\delta > 0$, depending on the time complexity of $A, B$.

- BGI used ‘Basic’ algorithm, achieving $\delta = 1/T$ for running time $O(T)$.

- Apply a PRF on $g^t$ to randomize.
Results

(Optimal) Spaceships Algorithm

There is an algorithm enabling Alice and Bob to synchronize except for probability $O(1/T^2)$, where $T$ is the number of array-queries.

Corollary
If Alice and Bob landed with initial distance $M$ of each other, probability of failure would be $O(M/T^2)$.

Spaceships Optimality
This is optimal! (No algorithm can achieve $o(1/T^2)$ error.)

DDLOG optimality
Algorithm is optimal in groups satisfying the DLI-Hardness assumption.
**Results**

**(Optimal) Spaceships Algorithm**

There is an algorithm enabling Alice and Bob to synchronize except for probability $O(1/T^2)$, where $T$ is the number of array-queries.

**Corollary**

If Alice and Bob landed with initial distance $M$ of each other, probability of failure would be $O(M/T^2)$. 
(Optimal) Spaceships Algorithm

There is an algorithm enabling Alice and Bob to synchronize except for probability $O(1/T^2)$, where $T$ is the number of array-queries.

Corollary

If Alice and Bob landed with initial distance $M$ of each other, probability of failure would be $O(M/T^2)$.

Spaceships Optimality

This is optimal! (No algorithm can achieve $o(1/T^2)$ error.)
Results

(Optimal) Spaceships Algorithm

There is an algorithm enabling Alice and Bob to synchronize except for probability $O(1/T^2)$, where $T$ is the number of array-queries.

Corollary

If Alice and Bob landed with initial distance $M$ of each other, probability of failure would be $O(M/T^2)$.

Spaceships Optimality

This is optimal! (No algorithm can achieve $o(1/T^2)$ error.)

DDLOG optimality

Algorithm is optimal in groups satisfying the DLI-Hardness assumption.
Optimal Algorithm

- Algorithm is composed of several ‘jumping’ phases.
- The ‘Jump’ algorithm is a variant of Pollard’s kangaroo algorithm.
Algorithm is composed of several ‘jumping’ phases.

The ‘Jump’ algorithm is a variant of Pollard’s kangaroo algorithm.

We view a 2-staged algorithm, achieving $O(1/T^{3/2})$ error, with $T$ queries.
2 staged algorithm

- Phase 1 is just ‘Basic’ algorithm with $T/2$ steps.

![Diagram showing two stages of a synchronization algorithm](image)

$L = 1$
2 staged algorithm

- Phase 1 is just ‘Basic’ algorithm with $T/2$ steps.
- Phase 2 performs $T/2$ jumps of size $\sim U(1, \sqrt{T})$ depending on current location.
- Phase 2 starts from minimum of phase 1; stops at its own minimum.
Analysis of algorithm

- In each stage, both Alice and Bob perform a *Jump*.
- In first stage, fail to synchronize with probability $O\left(\frac{1}{T}\right)$.
- On second stage, parties make $T/2$ steps of size $\leq \sqrt{T} = \sqrt{T}$, their random walks are expected to meet after $O\left(\sqrt{T}\right)$ steps.
- Thus, walks share $T - O\left(\sqrt{T}\right)$ steps $\Rightarrow$ fail with prob. $O\left(\frac{\sqrt{T}}{T}\right)$.

$\Rightarrow$ After second stage, synchronized except for probability $O\left(\frac{T - 3/2}{2}\right)$!
Analysis of algorithm

In each stage, both Alice and Bob perform a *Jump*.
In first stage, fail to synchronize with probability $O\left(\frac{1}{T}\right)$.
After synchronizing, parties remain synchronized forever!
Parties failed on first stage $\Rightarrow$ their distance is $O(T)$.
Analysis of algorithm

In each stage, both Alice and Bob perform a *Jump*.

In first stage, fail to synchronize with probability $O(1/T)$.

After synchronizing, parties remain synchronized forever!

Parties failed on first stage $\Rightarrow$ their distance is $O(T)$.

On second stage, parties make $T/2$ steps of size $\leq L = \sqrt{T}$, their random walks are expected to meet after $O(\sqrt{T})$ steps.

Thus, walks share $T - O(\sqrt{T})$ steps $\Rightarrow$ fail with prob. $O(\sqrt{T}/T)$. 
In each stage, both Alice and Bob perform a *Jump*.

In first stage, fail to synchronize with probability $O\left(\frac{1}{T}\right)$.

After synchronizing, parties remain synchronized forever!

Parties failed on first stage $\Rightarrow$ their distance is $O(T)$.

On second stage, parties make $T/2$ steps of size $L = \sqrt{T}$, their random walks are expected to meet after $O(\sqrt{T})$ steps.

Thus, walks share $T - O(\sqrt{T})$ steps $\Rightarrow$ fail with prob. $O(\sqrt{T}/T)$.

After second stage, synchronized except for probability $O(T^{-3/2})$!
Analysis of algorithm

Why would two random walks, starting $O(T)$ apart, with step-size $O(\sqrt{T})$, collide after $O(\sqrt{T})$ steps?

In general, if distance $\sim D$, and step size $\sim L$, collision after $\sim \frac{D}{L} + L$ steps.

Choose $L = \sqrt{D}$ to minimize collision time.
Analysis of algorithm

- Why would two random walks, starting $O(T)$ apart, with step-size $O(\sqrt{T})$, collide after $O(\sqrt{T})$ steps?

- In general, if distance $\sim D$, and step size $\sim L$, collision after $\sim D/L + L$ steps.
  - Choose $L = \sqrt{D}$ to minimize collision time.
Summary

In case Alice and Bob start with distance 1 apart, they can meet except for probability $O(1/T^2)$, by reading $T$ cells.

- What if their initial distance is $M$?
In case Alice and Bob start with distance 1 apart, they can meet except for probability $O(1/T^2)$, by reading $T$ cells.

- What if their initial distance is $M$?

By using the same algorithm, they meet except for probability $O(M/T^2)$.

$$\Pr[f(A) \neq f(B)] \leq \Pr[f(A) \neq f(C)] + \cdots + \Pr[f(J) \neq f(B)]$$
Lower Bound on DDLOG (assuming DLI)

Discrete-Log-in-Interval Hardness assumption

Many concrete (families of) cryptographic groups $\mathbb{G}$ with generator $g$, satisfy:
No algorithm can find $x$, given $g^x$, in time $o(\sqrt{R})$, assuming $x \sim U(0, R)$.

Spaceships algorithm solves DLI in $O(\sqrt{R})$.
Land Alice & Bob on the array $arr[t] = g^t$.
Put Alice at $g^0$, and Bob at $g^x$.
Let them perform $O(2\sqrt{R})$ queries.
Finds DLOG except for prob. $x/2 < 1/2$.
Hence optimal!
Solution for the spaceships problem can truly be proved to be optimal up to constant factors.
Lower Bound on DDLOG (assuming DLI)

Discrete-Log-in-Interval Hardness assumption

Many concrete (families of) cryptographic groups $\mathbb{G}$ with generator $g$, satisfy:
No algorithm can find $x$, given $g^x$, in time $o(\sqrt{R})$, assuming $x \sim U(0, R)$.

- Spaceships algorithm solves DLI in $O(\sqrt{R})$.
  - Land Alice & Bob on the array $arr[t] = g^t$.
  - Put Alice at $g^0$, and Bob at $g^x$.
  - Let them perform $O(2\sqrt{R})$ queries.
  - Finds DLOG except for prob. $x/(2\sqrt{R})^2 < 1/2$.

- Hence optimal!

- Solution for the spaceships problem can truly be proved to be optimal up to constant factors.

Ohad Klein (BIU)  How to Synchronize Efficiently?  Aug. 22, 2018  0xF / 0x17
Summary

- The Distributed Discrete Log Problem.
- Application to HSS.
- An optimal algorithm solving DDLOG. (Improving $O(1/T)$ to $O(1/T^2)$)
- A formal analysis of the algorithm is in the paper.
- A matching lower bound assuming DLI hardness assumption.

Techniques

- Iterated variant of Pollard’s Kangaroo algorithm.
- Analysis of algorithm with Martingales.
- Lower bounds with Discrete Fourier Analysis.
Thank You!
Algorithm: Optimal

1: function Optimal(arr, start, T)
2:   s = Jump(arr, start, T, 1);
3:   s = Jump(arr, s, T, T^{1/2});
4:   s = Jump(arr, s, T, T^{3/4});
5:   s = Jump(arr, s, T, T^{7/8});
6: ... ...
7: return s;

Algorithm: Jump

1: function Jump(arr, s, T, L)
2:   i, min = s, s;
3:   repeat T times
4:     if arr[i] < arr[min] then
5:       min = i;
6:     i += 1 + Hash(arr[i]) mod L;
7:   return min;