ASIACRYPT 2023

# Unconditionally Secure Multiparty Computation for Symmetric Functions with Low Bottleneck Complexity

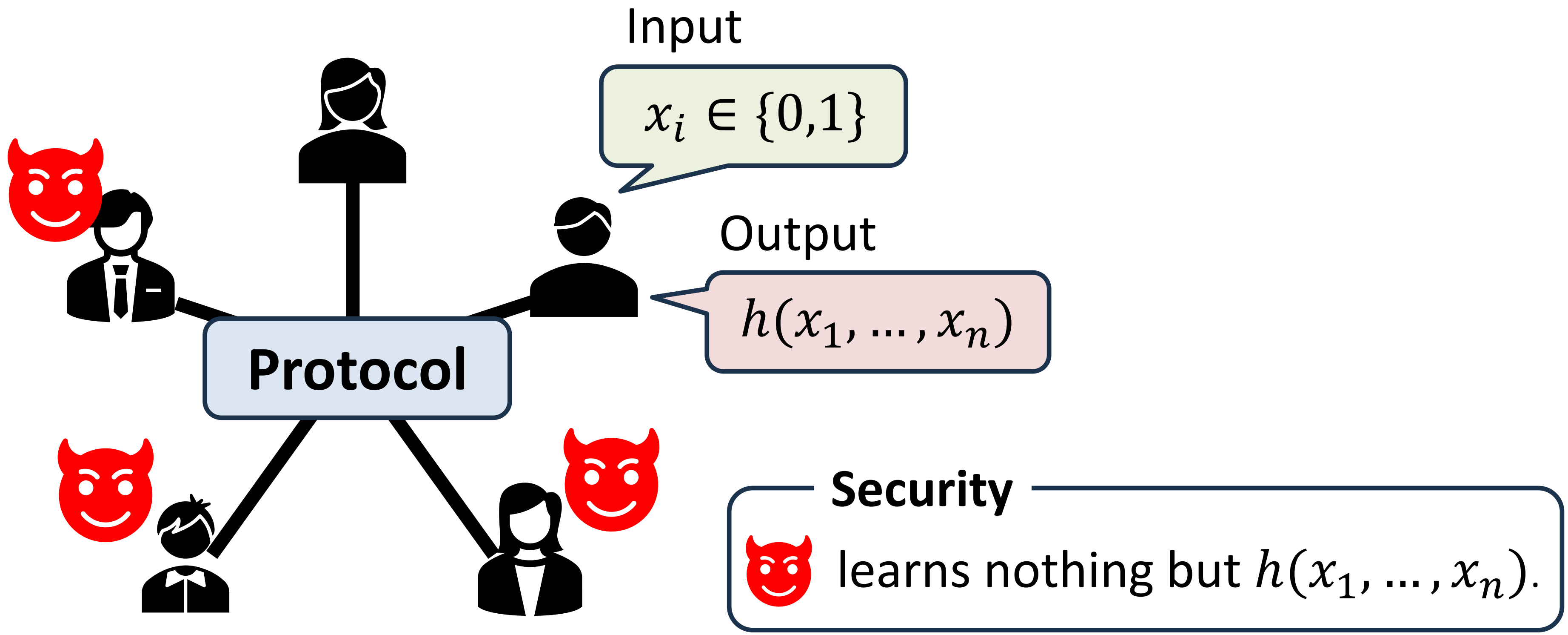
December 8, 2023

Reo Eriguchi

AIST, Japan

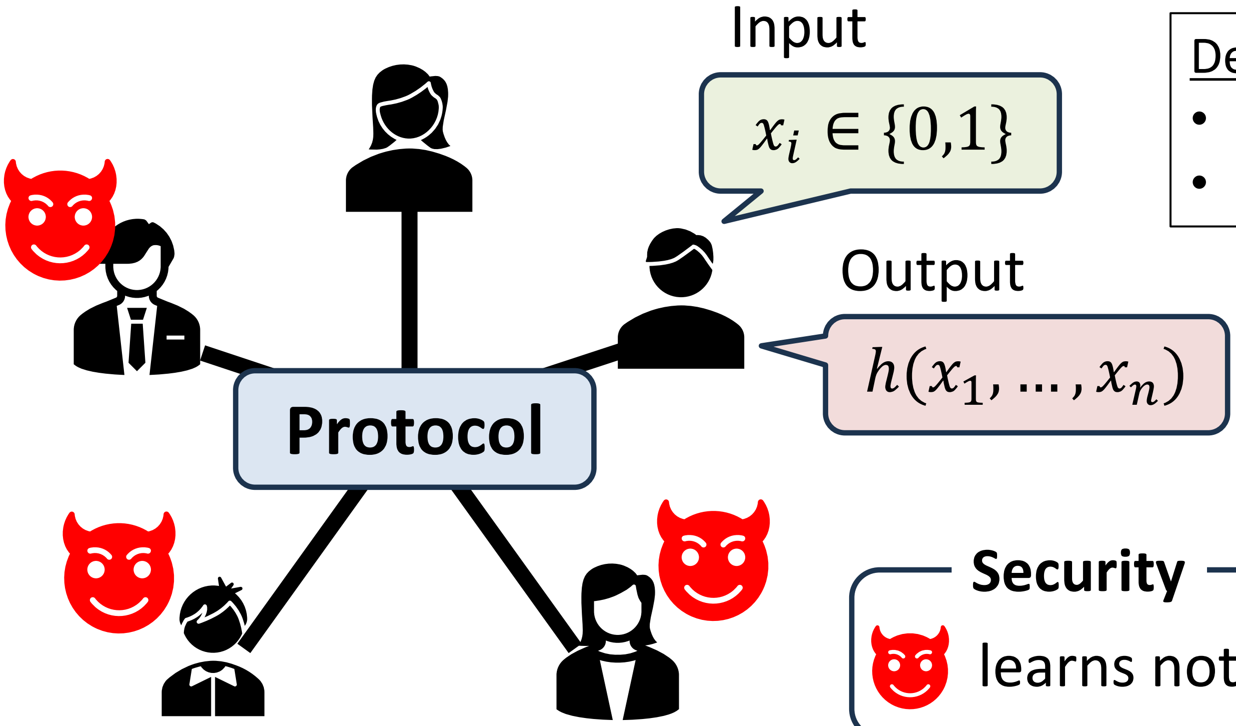
# Secure Multiparty Computation (MPC)

- MPC for a function $h : \{0,1\}^n \rightarrow \{0,1\}$



# Secure Multiparty Computation (MPC)

- MPC for a function $h : \{0,1\}^n \rightarrow \{0,1\}$




<u>Details on our setting</u>

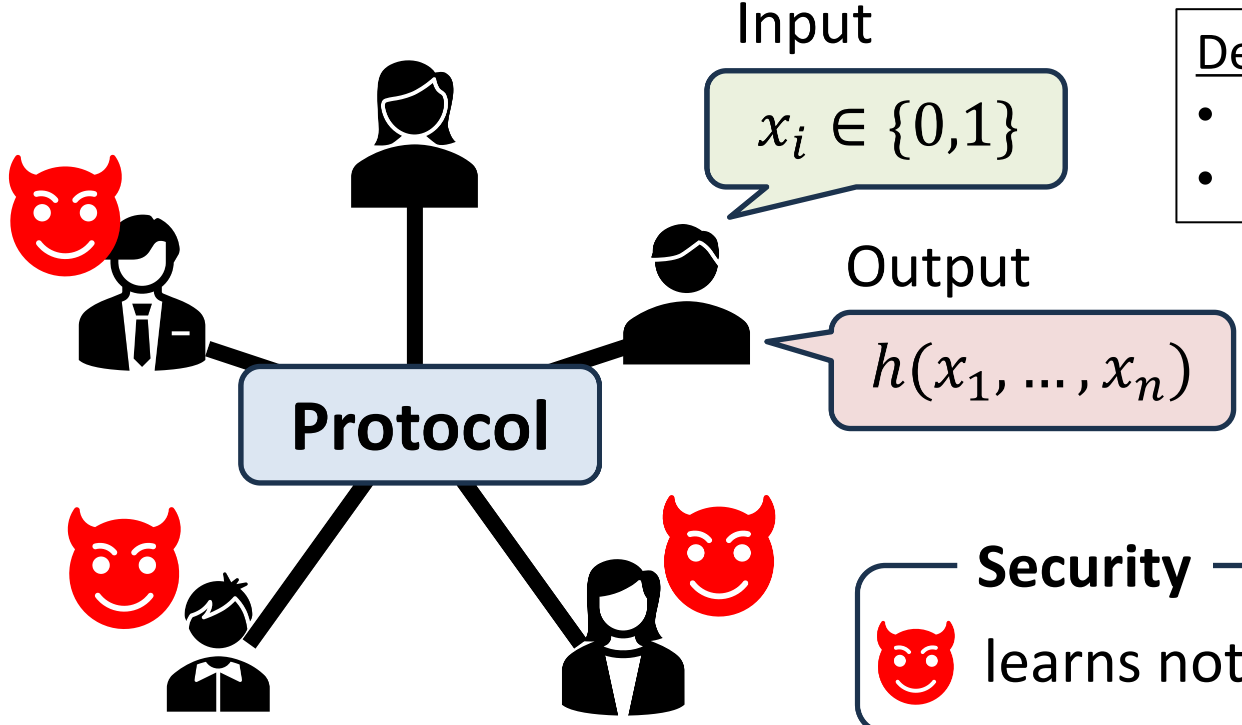
- Semi-honest adversaries
- Preprocessing model

**Security**

learns nothing but $h(x_1, \ldots, x_n)$.

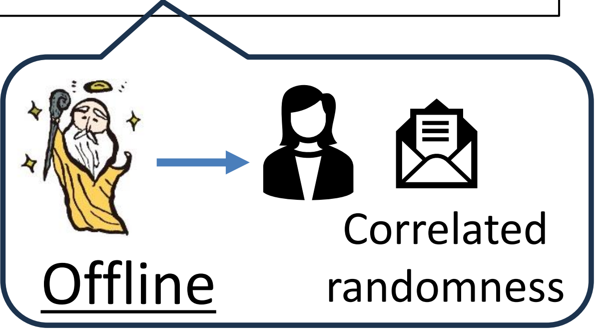
# Secure Multiparty Computation (MPC)

- MPC for a function $h : \{0,1\}^n \rightarrow \{0,1\}$




Details on our setting

- Semi-honest adversaries
- Preprocessing model



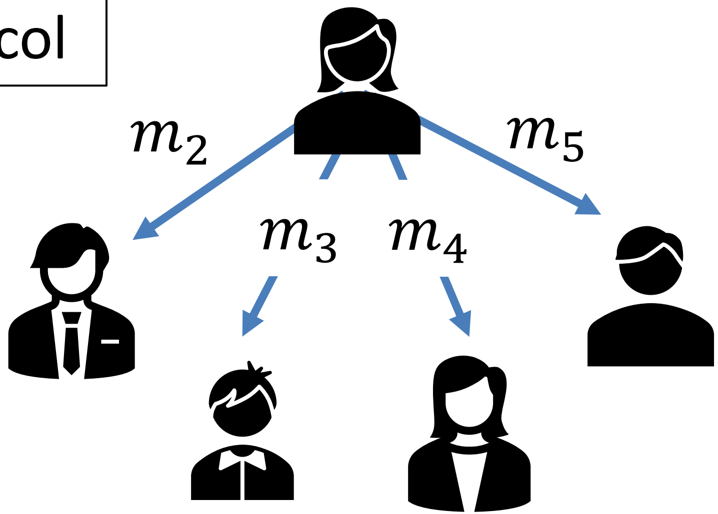

**Security**

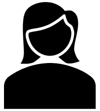
learns nothing but $h(x_1, \ldots, x_n)$.

# Bottleneck Complexity [BJPY18]

- Efficiency measure capturing the *load-balancing* aspect of protocols

Standard protocol




 bears $\Omega(n)$ communication

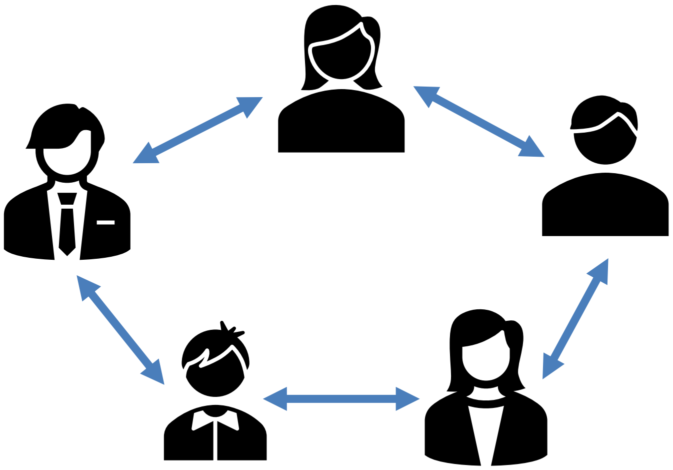
Bottleneck for efficiency if $n$ is large

[BJPY18] Boyle, Jain, Prabhakaran, Yu, : The Bottleneck Complexity of Secure Multiparty Computation. ICALP 2018

# Bottleneck Complexity [BJPY18]

- Efficiency measure capturing the *load-balancing* aspect of protocols

Protocol with low BC



Maximum per-party communication cost is possibly $o(n)$

$=$ *Bottleneck complexity*

Fit large-scale networks!

# Previous Results

- [BJPY18] showed that
  it is impossible to securely compute *all functions* with $o(n)$ BC.

# Previous Results

- [BJPY18] showed that
  it is impossible to securely compute *all functions* with $o(n)$ BC.

- Computationally secure MPC for *symmetric functions* with $o(n)$ BC
  $h\left(x_{\sigma(1)}, \dots, x_{\sigma(n)}\right) = h(x_1, \dots, x_n)$ for any permutation $\sigma$

# Previous Results

- [BJPY18] showed that
  it is impossible to securely compute *all functions* with $o(n)$ BC.

- Computationally secure MPC for *symmetric functions* with $o(n)$ BC
  $h\left(x_{\sigma(1)}, \dots, x_{\sigma(n)}\right) = h(x_1, \dots, x_n)$ for any permutation $\sigma$
  - Based on fully homomorphic encryption [BJPY18]
  - Based on one-way functions [ORS22]

[ORS22] Orlandi, Ravi, Scholl: On the bottleneck complexity of MPC with correlated randomness. PKC 2022

# Previous Results

- [BJPY18] showed that
  it is impossible to securely compute *all functions* with $o(n)$ BC.

- Computationally secure MPC for *symmetric functions* with $o(n)$ BC
  $h\left(x_{\sigma(1)}, \dots, x_{\sigma(n)}\right) = h(x_1, \dots, x_n)$ for any permutation $\sigma$
  - Based on fully homomorphic encryption [BJPY18]
  - Based on one-way functions [ORS22]

*Can we construct unconditionally secure MPC protocols for symmetric functions with $o(n)$ BC?*

[ORS22] Orlandi, Ravi, Scholl: On the bottleneck complexity of MPC with correlated randomness. PKC 2022

# Our Results

- Unconditionally secure protocols for symmetric functions such that:

| Protocol | Bottleneck complexity | Correlated randomness | Corruption |
|---|---|---|---|
| 1st protocol* | $O(\log n)$ | $O(n)$ | $n-1$ |
| 2nd protocol | $O(\sqrt{n})$ | $O(\sqrt{n})$ | $n-1$ |
| 3rd protocol | $O\left(n^{1/d}\log n\right)$ | $O\left(n^{1/d}\log n\right)$ | $< n/(d-1)$ |

* Independently discovered by [KOPR23]

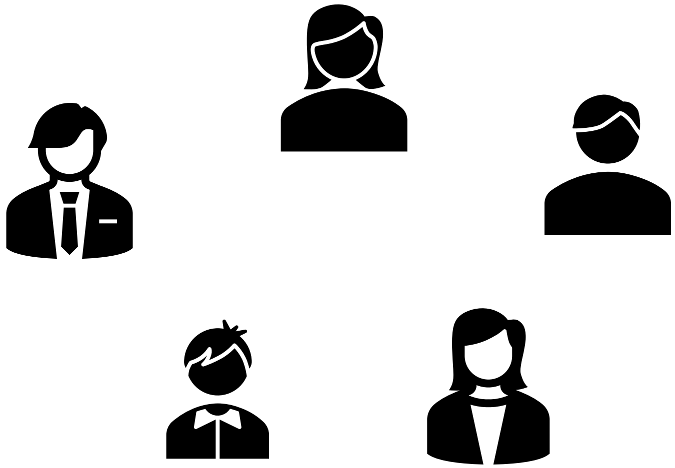
$d \geq 2$ is any constant

- More efficient protocols tailored to
  - AND function
  - Private set intersection

[KOPR23] Keller, Orlandi, Paskin-Cherniavsky, Ravi: MPC with low bottleneck-complexity: Information-theoretic security and more. ITC 2023

# Warm-up

- **Protocol for SUM** (over a group $\mathbb{G}$)
  - Input: $x_i$
  - Output: $s = \sum_i x_i$



# Warm-up

- **Protocol for SUM** (over a group $\mathbb{G}$)
  - Input: $x_i$
  - Output: $s = \sum_i x_i$

Offline

$(r_i)_{i\in[n]}$ :
additive shares of 0

Corr. rand.


CR: $O(\log|\mathbb{G}|)$

# Warm-up

- **Protocol for SUM** (over a group $\mathbb{G}$)
  - Input: $x_i$
  - Output: $s = \sum_i x_i$

Output

$$m_n = s + \sum_i r_i$$
$$= s$$

Corr. rand.

$r_i$

Input

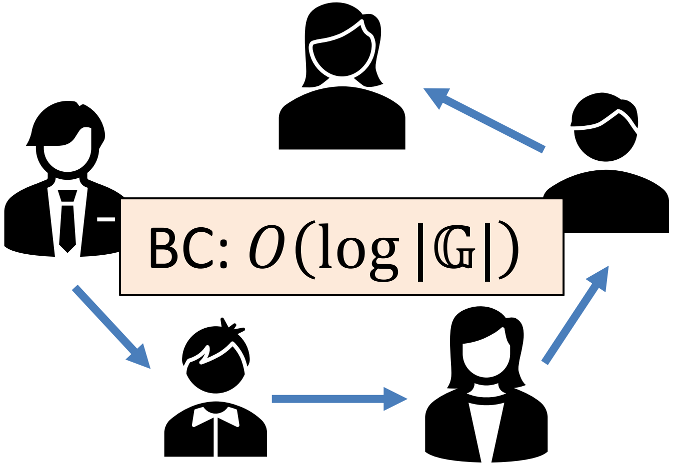
$x_i$



<u>Online</u>

Parties send

$$m_i \leftarrow m_{i-1} + x_i + r_i$$



# Warm-up

- **Protocol for SUM** (over a group $\mathbb{G}$)
  - Input: $x_i$
  - Output: $s = \sum_i x_i$

Output

$$m_n = s + \sum_i r_i$$
$$= s$$

Corr. rand.

$r_i$



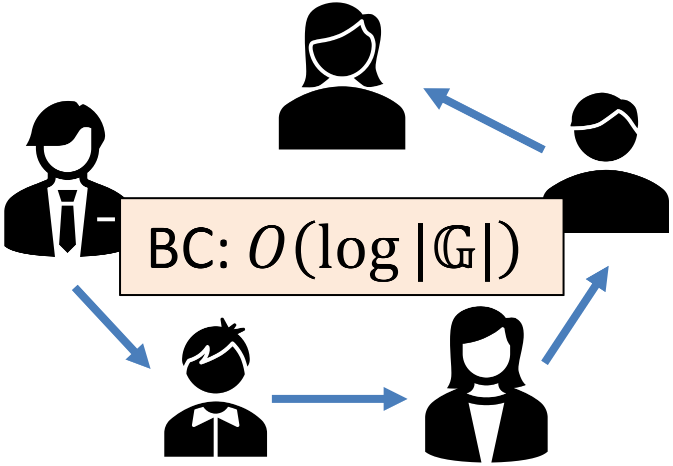
Input

$x_i$

<u>Online</u>

Parties send

$$m_i \leftarrow m_{i-1} + x_i + r_i$$




Opening secrets can be done with low BC

# Symmetric Function

- If $h$ is symmetric, $h(x_1, \dots, x_n)$ depends only on the number of 1's.

  ➡ There exists $f : \{0,1,\dots,n\} \rightarrow \{0,1\}$ such that

$$h(x_1, \dots, x_n) = f(s), \text{ where } s = \sum_i x_i$$

# One-Time Truth Table [IKM+13]

- Secure computation of $f(s)$ based on the truth table

$$\mathbf{T}_f = \big(f(0), \dots, f(n)\big)$$

$$[r], \begin{array}{cl} \begin{pmatrix} [0] \\ \vdots \\ \color{red}{[1]} \\ \vdots \\ [0] \\ \vdots \\ [0] \end{pmatrix} & \begin{array}{l} \\ \\ < r \\ \\ \\ \\ \\ \end{array} \end{array} \Rightarrow \; s - r \downarrow \begin{array}{cl} \begin{pmatrix} [0] \\ \vdots \\ [0] \\ \vdots \\ \color{red}{[1]} \\ \vdots \\ [0] \end{pmatrix} & \begin{array}{l} \\ \\ \\ \\ < s \\ \\ \\ \end{array} \end{array} \Rightarrow \; \mathbf{T}_f \cdot \begin{pmatrix} [0] \\ \vdots \\ [0] \\ \vdots \\ \color{red}{[1]} \\ \vdots \\ [0] \end{pmatrix} = [f(s)]$$

[IKM+13] Ishai, Kushilevitz, Meldgaard, Orlandi, Paskin-Cherniavsky: On the power of correlated randomness in secure computation. TCC 2013

# Our First Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s)$, where $s = \sum_i x_i$

<u>Offline</u>

$$r \leftarrow_{\$} \mathbb{Z}_{n+1} = \{0,1,\dots,n\}$$

$$\mathbf{e}_r = (\underset{\overset{\wedge}{0}}{0}, \dots, \underset{\overset{\wedge}{r}}{1}, \dots, \underset{\overset{\wedge}{n}}{0})^\top$$

Correlated randomness

$[r]_i$ $[\mathbf{e}_r]_i$

CR: $O(n)$

$[r]_i$, $[\mathbf{e}_r]_i = ([0]_i, \dots, [1]_i, \dots, [0]_i)^\top$ : additive shares

# Our First Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s)$, where $s = \sum_i x_i$

<u>Offline</u>

$$r \leftarrow_\$ \mathbb{Z}_{n+1} = \{0,1,\ldots,n\}$$

$$\mathbf{e}_r = (\underset{\hat{0}}{0}, \ldots, \underset{\hat{r}}{1}, \ldots, \underset{\hat{n}}{0})^\top$$

$[r]_i, [\mathbf{e}_r]_i = ([0]_i, \ldots, [1]_i, \ldots, [0]_i)^\top$: additive shares

Correlated randomness

$[r]_i \ [\mathbf{e}_r]_i$

CR: $O(n)$

# Our First Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s)$, where $s = \sum_i x_i$



Can be done with low BC

<u>Online</u>

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Correlated randomness

$[r]_i \;\; [\mathbf{e}_r]_i$

# Our First Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s)$, where $s = \sum_i x_i$



Online

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Can be done with low BC

(2) Permute $[\mathbf{e}_r]_i$ with shift $s - r$:

$$[\mathbf{e}_r]_i \to \left[\mathbf{e}_{r+(s-r)}\right]_i = [\mathbf{e}_s]_i$$

Local computation

Correlated randomness

$[r]_i \ \ [\mathbf{e}_r]_i$

# Our First Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s)$, where $s = \sum_i x_i$



Can be done with low BC

## Online

(1) Open $\sum_i (x_i - [r]_i) = s - r$

(2) Permute $[\mathbf{e}_r]_i$ with shift $s - r$:

$$[\mathbf{e}_r]_i \to \left[\mathbf{e}_{r+(s-r)}\right]_i = [\mathbf{e}_s]_i$$

(3) Multiply $[\mathbf{e}_s]_i$ by $\mathbf{T}_f = \left(f(0), \ldots, f(n)\right)$:

$$[f(s)]_i = \mathbf{T}_f \cdot [\mathbf{e}_s]_i$$

(4) Open $\sum_i [f(s)]_i = f(s)$

## Correlated randomness

$[r]_i \;\; [\mathbf{e}_r]_i$

Local computation

# Our First Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s)$, where $s = \sum_i x_i$



Online

Correlated randomness

$[r]_i \;\; [\mathbf{e}_r]_i$

Can be done with low BC

(1) Open $\sum_i (x_i - [r]_i) = s - r$

(2) Permute $[\mathbf{e}_r]_i$ with shift $s - r$:

$$[\mathbf{e}_r]_i \to \left[\mathbf{e}_{r+(s-r)}\right]_i = [\mathbf{e}_s]_i$$

(3) Multiply $[\mathbf{e}_s]_i$ by $\mathbf{T}_f = \big(f(0), \ldots, f(n)\big)$:

$$[f(s)]_i = \mathbf{T}_f \cdot [\mathbf{e}_s]_i$$

Local computation

(4) Open $\sum_i [f(s)]_i = f(s)$

$O(\log n)$ BC

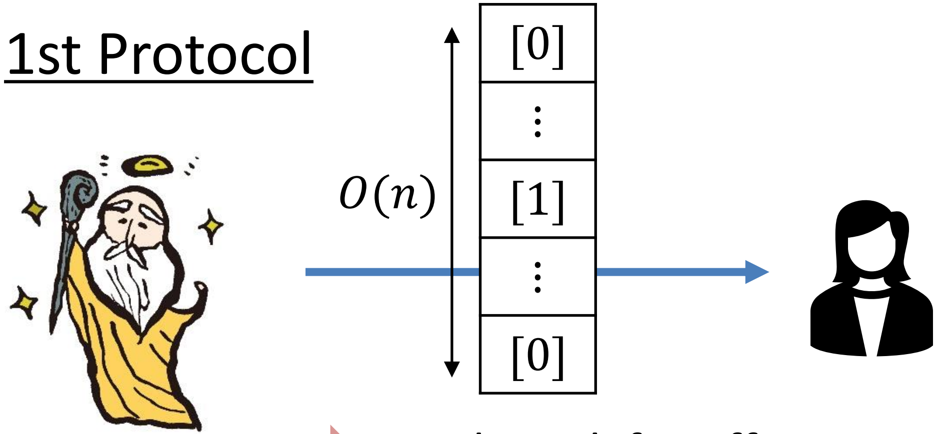
# Our Second Protocol

- 1st protocol has a large amount of correlated randomness.

| Protocol | Bottleneck complexity | Correlated randomness |
|---|---|---|
| 1st protocol | $O(\log n)$ | $O(n)$ |
| 2nd protocol | $O(\sqrt{n})$ | $O(\sqrt{n})$ |

The overall cost is *sublinear*

1st Protocol




Bottleneck for efficiency

# Reducing Correlated Randomness

$$f(s) = [f(0), \dots, f(n)] \cdot \begin{array}{c} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \end{array} \begin{array}{l} \\ \\ < s \\ \\ \\ \end{array}$$

CRT implies the correspondence
$\phi : \mathbb{Z}_{n+1} \ni s \longmapsto (s_1, s_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$

$$f(s) = \begin{array}{c} [0, \dots, 1, \dots, 0] \\ \hphantom{[0, \dots,}\hat{s}_1\hphantom{, \dots, 0]} \end{array} \cdot \underbrace{\overbrace{\boxed{\mathbf{M}_f}}^{q}}_{p \times q} \cdot \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ < s_2 \\ \\ \\ \end{array}$$

The product of $\sqrt{n}$-dim. matrix/vectors

$O(\max\{p, q\}) = O(\sqrt{n})$ correlated randomness

# Our Second Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s) = \mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$

# Our Second Protocol

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^\top \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

Offline

$r \leftarrow_{\$} \mathbb{Z}_{n+1}$

$\phi(r) = (r_1, r_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$

# Our Second Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^\top \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$

<u>Offline</u>

$r \leftarrow_\$ \mathbb{Z}_{n+1}$

$\phi(r) = (r_1, r_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$

$$\mathbf{e}_{r_1} = (\underset{\substack{\wedge \\ 0}}{0}, \dots, \underset{\substack{\wedge \\ r_1}}{1}, \dots, \underset{\substack{\wedge \\ p}}{0})^\top$$

$$\mathbf{e}_{r_2} = (\underset{\substack{\wedge \\ 0}}{0}, \dots, \underset{\substack{\wedge \\ r_2}}{1}, \dots, \underset{\substack{\wedge \\ q}}{0})^\top$$

# Our Second Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^\top \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$

<u>Offline</u>

$r \leftarrow_\$ \mathbb{Z}_{n+1}$

$\phi(r) = (r_1, r_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$

$\mathbf{e}_{r_1} = (\underset{\substack{\wedge\\0}}{0}, \dots, \underset{\substack{\wedge\\r_1}}{1}, \dots, \underset{\substack{\wedge\\p}}{0})^\top$

$\mathbf{e}_{r_2} = (\underset{\substack{\wedge\\0}}{0}, \dots, \underset{\substack{\wedge\\r_2}}{1}, \dots, \underset{\substack{\wedge\\q}}{0})^\top$

Correlated randomness

$[r]_i$

$[\mathbf{e}_{r_1}]_i$

$[\mathbf{e}_{r_2}]_i$

CR : $O(\max\{p, q\}) = O(\sqrt{n})$

# Our Second Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$



<u>Online</u>

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Low BC

Correlated randomness

$[r]_i$

$[\mathbf{e}_{r_1}]_i$

$[\mathbf{e}_{r_2}]_i$

# Our Second Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i, \ (s_1, s_2) = \phi(s)$



<u>Online</u>

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Low BC

(2) Decompose $\phi(s - r) = (y_1, y_2)$

Local computation

Correlated randomness

$[r]_i$

$[\mathbf{e}_{r_1}]_i$

$[\mathbf{e}_{r_2}]_i$

# Our Second Protocol

- Input: $(x_1, \dots, x_n)$
- Output: $h(x_1, \dots, x_n) = f(s) = \mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i, \; (s_1, s_2) = \phi(s)$



<u>Online</u>

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Low BC

(2) Decompose $\phi(s - r) = (y_1, y_2)$

(3) Permute $[\mathbf{e}_{r_b}]_i$ with shift $y_b$:

$$[\mathbf{e}_{r_b}]_i \rightarrow [\mathbf{e}_{r_b + y_b}]_i = [\mathbf{e}_{s_b}]_i$$

Local computation

Correlated randomness

$[r]_i$

$[\mathbf{e}_{r_1}]_i$

$[\mathbf{e}_{r_2}]_i$

# Our Second Protocol

- Input: $(x_1, \ldots, x_n)$
- Output: $h(x_1, \ldots, x_n) = f(s) = \mathbf{e}_{s_1}^\top \cdot \mathbf{M}_f \cdot \mathbf{e}_{s_2}$

$s = \sum_i x_i,\ (s_1, s_2) = \phi(s)$



Online

(1) Open $\sum_i (x_i - [r]_i) = s - r$

Low BC

(2) Decompose $\phi(s - r) = (y_1, y_2)$

(3) Permute $\left[\mathbf{e}_{r_b}\right]_i$ with shift $y_b$:

$$\left[\mathbf{e}_{r_b}\right]_i \to \left[\mathbf{e}_{r_b + y_b}\right]_i = \left[\mathbf{e}_{s_b}\right]_i$$

Local computation

(4) Multiply $\left[\mathbf{e}_{s_2}\right]_i$ with a *constant* matrix $\mathbf{M}_f$:

$$\left[\mathbf{e}_{s_2}\right]_i \to \left[\mathbf{M}_f \mathbf{e}_{s_2}\right]_i$$

Correlated randomness

$[r]_i$

$\left[\mathbf{e}_{r_1}\right]_i$

$\left[\mathbf{e}_{r_2}\right]_i$

# Our Second Protocol

- The remaining step is to securely obtain

$$\mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \mathbf{e}_{s_2} \text{ from } \left[\mathbf{e}_{s_1}\right]_i \text{ and } \left[\mathbf{M}_f \mathbf{e}_{s_2}\right]_i$$

- How do we securely compute the product of secrets?

# Our Second Protocol

- The remaining step is to securely obtain

$$\mathbf{e}_{s_1}^{\top} \cdot \mathbf{M}_f \mathbf{e}_{s_2} \text{ from } [\mathbf{e}_{s_1}]_i \text{ and } [\mathbf{M}_f \mathbf{e}_{s_2}]_i$$

- How do we securely compute the product of secrets?

$\Rightarrow$ Using Beaver triples

**Novel observation**

Standard multiplication protocol based on Beaver triples has ***low BC***!

$\Rightarrow$ Both BC and CR are *constant* per multiplication

# Multiplication based on Beaver Triples

- Input: $[x]_i, [y]_i$ $(x, y \in \{0,1\})$
- Output: $[xy]_i$

Correlated randomness

Offline

$$a, b \leftarrow_{\$} \{0,1\}$$
$$c \leftarrow ab$$

$[a]_i$

$[b]_i$

$[c]_i$

# Multiplication based on Beaver Triples

- Input: $[x]_i, [y]_i \ \ (x, y \in \{0,1\})$
- Output: $[xy]_i$



<u>Online</u>

Low BC

(1) Open $x' = x - a, y' = y - b$

Correlated randomness

$[a]_i$

$[b]_i$

$[c]_i$

# Multiplication based on Beaver Triples

- Input: $[x]_i, [y]_i \ \ (x, y \in \{0,1\})$
- Output: $[xy]_i$

<u>Online</u>

Low BC



(1) Open $x' = x - a, y' = y - b$

(2) Compute

$$[xy]_i = [x'y']_i + x'[b]_i + y'[a]_i + [c]_i$$

Local computation

Correlated randomness

$[a]_i$

$[b]_i$

$[c]_i$

# Multiplication based on Beaver Triples

- Input: $[x]_i, [y]_i \ \ (x, y \in \{0,1\})$
- Output: $[xy]_i$

<u>Online</u>

Low BC



(1) Open $x' = x - a, y' = y - b$

(2) Compute

$$[xy]_i = [x'y']_i + x'[b]_i + y'[a]_i + [c]_i$$

Local computation

Correlated randomness

$[a]_i$

$[b]_i$

$[c]_i$

➡ Both BC and CR are *constant* per multiplication

# Our Second Protocol

- $O(\sqrt{n})$ Beaver triples suffice for secure computation of $\mathbf{e}_{s_1}^\top \cdot \mathbf{M}_f \mathbf{e}_{s_2}$ from $\left[\mathbf{e}_{s_1}\right]_i$ and $\left[\mathbf{M}_f \mathbf{e}_{s_2}\right]_i$.

$\Rightarrow$ BC and CR of the protocol are $O(\sqrt{n})$

# Summary

- Bottleneck complexity captures load-balancing aspect of MPC.
- Previous protocols computing symmetric functions with $o(n)$ BC are *computationally secure*.
- We construct *unconditionally secure* protocols such that

| Protocol | Bottleneck complexity | Correlated randomness | Corruption |
|---|---|---|---|
| 1st protocol | $O(\log n)$ | $O(n)$ | $n-1$ |
| 2nd protocol | $O(\sqrt{n})$ | $O(\sqrt{n})$ | $n-1$ |
| 3rd protocol | $O\left(n^{1/d}\log n\right)$ | $O\left(n^{1/d}\log n\right)$ | $< n/(d-1)$ |

$d \geq 2$ is any constant

- More efficient protocols tailored to AND function and PSI.

# Future Work

- What is the optimal bottleneck complexity of computing symmetric functions?
  - E.g., is there a secure protocol such that both BC and CR are $O(\log n)$?

- Can we derive a lower bound on bottleneck complexity for *symmetric* functions?
  - [BJPY18] derived a non-trivial lower bound for *general* functions.

- Can we achieve malicious security unconditionally?
  - [BJPY18] showed a generic compiler based on heavy cryptographic primitives.

# Thank you!

Please see https://eprint.iacr.org/2023/662 for the full paper.