Chair of
Applied Cryptography

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Multi-Client Oblivious RAM with Poly-Logarithmic Communication

Sherman S. M. Chow[1], Katharina Fech[2], **Russell W. F. Lai**[2], Giulio Malavolta[3]

[1]The Chinese University of Hong Kong
[2]Friedrich-Alexander University Erlangen-Nuremberg
[3]Max Planck Institute for Security and Privacy

Oblivious Random Access Machine (ORAM) [Goldreich-Ostrovsky'96]

Logical Access

$(\texttt{Write}, \mathsf{addr}, \mathsf{val})$

$\downarrow$

$\boxed{\mathsf{ORAM.Compile}(\text{secret state}, \cdot)}$

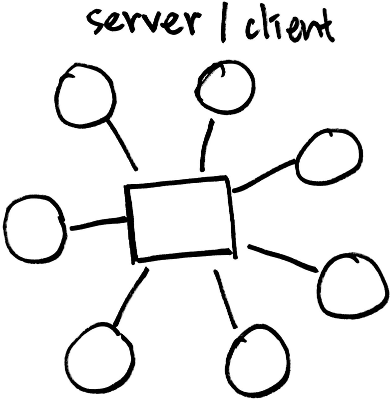
$\downarrow$

Physical Accesses

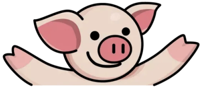
$(\texttt{Read}, \mathsf{addr}_1), (\texttt{Write}, \mathsf{addr}_1, \mathsf{val}_1), \ldots, (\texttt{Read}, \mathsf{addr}_\ell), (\texttt{Write}, \mathsf{addr}_\ell, \mathsf{val}_\ell)$

$\ell = \mathsf{polylog}(n)$ communication overhead (*e.g.*, PathORAM) for database size n

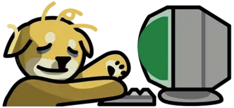
Network Setting?



Multi-Client ORAM (MCORAM) [Maffei et al.'17]



Data Owner

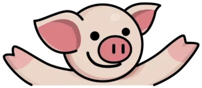


Server

Malicious Client I

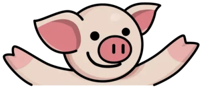
Honest Client II

Multi-Client ORAM (MCORAM)



addr	1	2	...	i	...	N
val	v_1	v_2	...	v_i	...	v_N
Readable by	I	II	...	I	...	I, II
Writable by	I	II	...	-	...	I

Multi-Client ORAM (MCORAM)

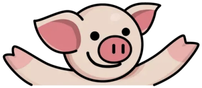


addr	1	2	...	i	...	N
val	v_1	v_2	...	v_i	...	v_N
Readable by	I	II	...	I	...	I, II
Writable by	I	II	...	-	...	I

Obliviousness

$(\texttt{Read}, 2), (\texttt{Write}, 2, v_2')$

Multi-Client ORAM (MCORAM)

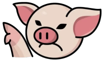


addr	1	2	...	i	...	N
val	v_1	v_2	...	v_i	...	v_N
Readable by	I	II	...	I	...	I, II
Writable by	I	II	...	-	...	I

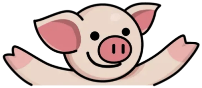
Obliviousness

$(\texttt{Read}, 2), (\texttt{Write}, 2, v_2')$

$(\texttt{Read}, 2)$



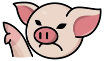
Multi-Client ORAM (MCORAM)



addr	1	2	...	i	...	N
val	v_1	v_2	...	v_i	...	v_N
Readable by	I	II	...	I	...	I, II
Writable by	I	II	...	-	...	I

Integrity

$(\texttt{Write}, 2, v_2')$



Multi-Client ORAM (MCORAM)

Syntax (Simplified)

$$
\begin{aligned}
(\mathsf{msk}, \mathsf{EDB}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DB}) \\
\mathsf{sk}_i &\leftarrow \mathsf{KGen}(\mathsf{msk}, \mathsf{policy}_i) \\
\langle v' | \mathsf{EDB}' \rangle &\leftarrow \mathsf{OAccess}\langle C(\mathsf{sk}_i, \mathsf{addr}, v) | S(\mathsf{EDB}) \rangle && \textit{\# } v = \bot \textit{ for read}
\end{aligned}
$$

Multi-Client ORAM (MCORAM)

Syntax (Simplified)

$$
\begin{aligned}
(\mathsf{msk}, \mathsf{EDB}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DB}) \\
\mathsf{sk}_i &\leftarrow \mathsf{KGen}(\mathsf{msk}, \mathsf{policy}_i) \\
\langle v' | \mathsf{EDB}' \rangle &\leftarrow \mathsf{OAccess}\langle C(\mathsf{sk}_i, \mathsf{addr}, v) | S(\mathsf{EDB}) \rangle \qquad \# \, v = \bot \text{ for read}
\end{aligned}
$$

Obliviousness

For malicious server S^* (colluding with corrupt clients),

$$
\begin{aligned}
&\mathsf{view}_{S^*}(\mathsf{OAccess}\langle C(\mathsf{sk}, \mathsf{addr}, v) | S^* \rangle) \\
\approx &\mathsf{view}_{S^*}(\mathsf{OAccess}\langle C(\mathsf{sk}', \mathsf{addr}', v') | S^* \rangle)
\end{aligned}
$$

unless they are trivially distinguishable.

Multi-Client ORAM (MCORAM)

Syntax (Simplified)

$$(\mathsf{msk}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DB})$$

$$\mathsf{sk}_i \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathsf{policy}_i)$$

$$\langle v' | \mathsf{EDB}' \rangle \leftarrow \mathsf{OAccess}\langle C(\mathsf{sk}_i, \mathsf{addr}, v) | S(\mathsf{EDB}) \rangle \qquad \textit{\# } v = \perp \textit{ for read}$$

Integrity

Let $\mathsf{legal}((C_1, \mathsf{addr}_1, v_1), \ldots, (C_\ell, \mathsf{addr}_\ell, v_\ell)) = ((C_{i_1}, \mathsf{addr}_{i_1}, v_{i_1}), \ldots, (C_{i_k}, \mathsf{addr}_{i_k}, v_{i_k}))$. Let

$$\mathsf{EDB}' = (\mathsf{OAccess}(\mathsf{sk}_i, \mathsf{addr}_i, v_i, \cdot))_{i=1}^{\ell} \circ \mathsf{EDB} \qquad \textit{\# All accesses}$$

$$\mathsf{DB}' = (\mathsf{Access}(\mathsf{addr}_{i_j}, v_{i_j}, \cdot))_{j=1}^{k} \circ \mathsf{DB} \qquad \textit{\# Subsequence of legal accesses}$$

Then EDB' encodes the same values as DB'.

Existing Schemes and Our Result

Scheme	Malicious Client	Server Computation	Client Computation	Communication
[Maffei *et al.*'15]	✘	$O(\log n)$	$O(\log n)$	$O(\log n)$
[Maffei *et al.*'17]	✓	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$

Existing Schemes and Our Result

Scheme	Malicious Client	Server Computation	Client Computation	Communication
[Maffei *et al.*'15]	✘	$O(\log n)$	$O(\log n)$	$O(\log n)$
[Maffei *et al.*'17]	✓	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$

Is multi-client ORAM with $\mathrm{polylog}(n)$ ***communication possible?***

Existing Schemes and Our Result

Scheme	Malicious Client	Server Computation	Client Computation	Communication
[Maffei *et al.*'15]	✘	$O(\log n)$	$O(\log n)$	$O(\log n)$
[Maffei *et al.*'17]	✓	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$
This work	✓	$O(n)$	$O(\log n)$	$O(\log n)$

Is multi-client ORAM with $\mathrm{polylog}(n)$ ***communication possible?***

This Work: Yes!

Existing Schemes and Our Result

Scheme	Malicious Client	Server Computation	Client Computation	Communication
[Maffei *et al.*'15]	✘	$O(\log n)$	$O(\log n)$	$O(\log n)$
[Maffei *et al.*'17]	✓	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$
This work	✓	$O(n)$	$O(\log n)$	$O(\log n)$

Is multi-client ORAM with $\mathrm{polylog}(n)$ ***communication possible?***

This Work: Yes!

Main Technique: Cross-key evaluation of FHE ciphertexts

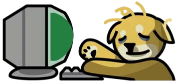
Fully Homomorphic Encryption (FHE)

Syntax

$$
\begin{aligned}
(\mathsf{ek}, \mathsf{dk}) &\leftarrow \mathsf{KGen}(1^\lambda) \\
\mathsf{ctxt} &\leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}) \\
\mathsf{msg} &\leftarrow \mathsf{Dec}(\mathsf{dk}, \mathsf{ctxt}) \\
\mathsf{ctxt}' &\leftarrow \mathsf{Eval}(\mathsf{ek}, f, (\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k))
\end{aligned}
$$

Warm-Up: Oblivious Read

Use private-information retrieval (PIR)! To read $\mathsf{addr}^* \in [n]$:



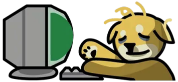
$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGen}(1^\lambda)$

$\overline{\mathsf{addr}}^* \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{addr}^*)$

$$\xrightarrow{\mathsf{ek}, \overline{\mathsf{addr}}^*}$$

Warm-Up: Oblivious Read

Use private-information retrieval (PIR)! To read $\text{addr}^* \in [n]$:



$(\text{ek}, \text{dk}) \leftarrow \text{KGen}(1^\lambda)$

$\overline{\text{addr}}^* \leftarrow \text{Enc}(\text{ek}, \text{addr}^*)$

$$\boxed{\begin{array}{l}\text{Read}_{v_1,\ldots,v_n}(\text{addr}) \\ \hline \textbf{return } v_{\text{addr}}\end{array}}$$

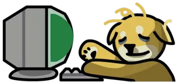
$$\xrightarrow{\text{ek}, \overline{\text{addr}}^*}$$

$\overline{v}_{\text{addr}^*} \leftarrow \text{Eval}(\text{ek}, \text{Read}_{v_1,\ldots,v_n}, \overline{\text{addr}}^*)$

$$\xleftarrow{\overline{v}_{\text{addr}^*}}$$

Warm-Up: Oblivious Read

Use private-information retrieval (PIR)! To read $\mathsf{addr}^* \in [n]$:



$\boxed{\begin{array}{l} \mathsf{Read}_{v_1,\ldots,v_n}(\mathsf{addr}) \\ \hline \textbf{return } v_{\mathsf{addr}} \end{array}}$

$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGen}(1^\lambda)$

$\overline{\mathsf{addr}}^* \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{addr}^*)$

$\xrightarrow{\mathsf{ek}, \overline{\mathsf{addr}}^*}$

$\overline{v}_{\mathsf{addr}^*} \leftarrow \mathsf{Eval}(\mathsf{ek}, \mathsf{Read}_{v_1,\ldots,v_n}, \overline{\mathsf{addr}}^*)$

$\xleftarrow{\overline{v}_{\mathsf{addr}^*}}$

$v_{\mathsf{addr}^*} = \mathsf{Dec}(\mathsf{dk}, \overline{v}_{\mathsf{addr}^*})$

$\textbf{return } v_{\mathsf{addr}^*}$

Read-Access Control

1. Instead of v_{addr}, store $\bar{v}_{\mathrm{addr}} \leftarrow \mathrm{Enc}(\mathrm{ek}_{\mathrm{addr}}, v_{\mathrm{addr}})$.
2. Reveal $\mathrm{dk}_{\mathrm{addr}}$ to clients with read-access to addr.

Challenge: Oblivious Write

Intuition

1. To write v^* to addr^*, client sends $(\overline{\mathsf{addr}}^*, \overline{v}^*) = \mathsf{Enc}(\mathsf{ek}_{\mathsf{addr}^*}, (\mathsf{addr}^*, v^*))$ to server.
2. Server homomorphically evaluates a function
   which updates $\overline{v}_{\mathsf{addr}}$ to $\overline{v}^*$ if $\mathsf{addr} = \mathsf{addr}^*$
   over each $\overline{v}_{\mathsf{addr}}$ and $(\overline{\mathsf{addr}}^*, \overline{v}^*)$.

Challenge: Oblivious Write

Intuition

1. To write v^* to addr^*, client sends $(\overline{\mathrm{addr}}^*, \bar{v}^*) = \mathrm{Enc}(\mathrm{ek}_{\mathrm{addr}^*}, (\mathrm{addr}^*, v^*))$ to server.
2. Server homomorphically evaluates a function
   which updates $\bar{v}_{\mathrm{addr}}$ to $\bar{v}^*$ if $\mathrm{addr} = \mathrm{addr}^*$
   over each $\bar{v}_{\mathrm{addr}}$ and $(\overline{\mathrm{addr}}^*, \bar{v}^*)$.

Problem

- When $\mathrm{addr} \neq \mathrm{addr}^*$, v_{addr} and (addr^*, v^*) are encrypted under $\mathrm{ek}_{\mathrm{addr}}$ and $\mathrm{ek}_{\mathrm{addr}^*}$ respectively, unclear how homomorphic evaluation works (*e.g.*, may overwrite v_{addr} with garbage).
- Revealing $\mathrm{ek}_{\mathrm{addr}^*}$ breaks obliviousness.

FHE Correctness

1. Decryption Correctness

For $(\mathsf{ek}, \mathsf{dk}) \in \mathsf{KGen}(1^\lambda)$,

$$\begin{tikzcd} \mathsf{msg} \arrow[d] & = & \mathsf{msg} \\ \boxed{\mathsf{Enc}(\mathsf{ek}, \cdot)} \arrow[r] & \mathsf{ctxt} \arrow[r] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[u] \end{tikzcd}$$

FHE Correctness

2. Evaluation Correctness

For $(\mathsf{ek}, \mathsf{dk}) \in \mathsf{KGen}(1^\lambda)$, for $\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k$ produced under ek,

$$
\begin{tikzcd}
(\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k) \arrow[r] \arrow[d] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & (x_1, \ldots, x_k) \arrow[r] & \boxed{f(\cdot)} \arrow[d] \\
\boxed{\mathsf{Eval}(\mathsf{ek}, f, \cdot)} \arrow[r] & \mathsf{ctxt}' \arrow[r] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & f(x_1, \ldots, x_k)
\end{tikzcd}
$$

FHE Correctness

2. Evaluation Correctness

For $(\mathsf{ek}, \mathsf{dk}) \in \mathsf{KGen}(1^\lambda)$, for $\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k$ produced under ek,

$$
\begin{tikzcd}
(\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k) \arrow[r] \arrow[d] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & (x_1, \ldots, x_k) \arrow[r] & \boxed{f(\cdot)} \arrow[d] \\
\boxed{\mathsf{Eval}(\mathsf{ek}, f, \cdot)} \arrow[r] & \mathsf{ctxt}' \arrow[r] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & f(x_1, \ldots, x_k)
\end{tikzcd}
$$

Unclear what happens if $\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k$ are evaluated/decrypted under the “wrong” ek/dk.

New: FHE Strong Correctness

1. Decryption Correctness

Same as before.

New: FHE Strong Correctness

1. Decryption Correctness

Same as before.

2. Well-Defined Decryption

For any dk, $\mathsf{Dec}(\mathsf{dk}, \cdot)$ is well-defined even for ctxt produced under $\mathsf{ek}' \neq \mathsf{ek}$ (although decryption might result in garbage).

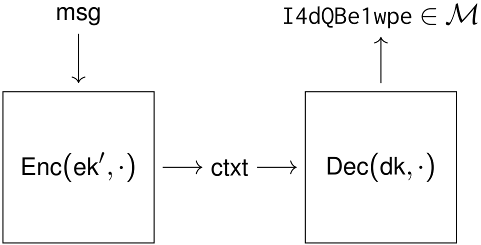
New: FHE Strong Correctness

1. Decryption Correctness

Same as before.

2. Well-Defined Decryption

For any dk, $\mathsf{Dec}(\mathsf{dk}, \cdot)$ is well-defined even for ctxt produced under $\mathsf{ek}' \neq \mathsf{ek}$ (although decryption might result in garbage).



New: FHE Strong Correctness

3. Strong Evaluation Correctness

For $(\mathsf{ek}, \mathsf{dk}) \in \mathsf{KGen}(1^\lambda)$, for **any** $\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k$ **in the ciphertext space**,

$$\begin{tikzcd}
(\mathsf{ctxt}_1, \ldots, \mathsf{ctxt}_k) \arrow[r] \arrow[d] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & (x_1, \ldots, x_k) \arrow[r] & \boxed{f(\cdot)} \arrow[d] \\
\boxed{\mathsf{Eval}(\mathsf{ek}, f, \cdot)} \arrow[r] & \mathsf{ctxt}' \arrow[r] & \boxed{\mathsf{Dec}(\mathsf{dk}, \cdot)} \arrow[r] & f(x_1, \ldots, x_k)
\end{tikzcd}$$

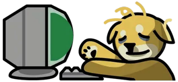
From Correctness to Strong Correctness

We construct a generic compiler $\mathsf{FHE} \mapsto \mathsf{FHE}'$:

FHE				FHE'
Well-Defined Decryption	+	(Standard) Correctness	$\implies$	Strong Correctness
Circular Security	+	Key Privacy/CPA-Security	$\implies$	Key Privacy/CPA-Security

Solution: Oblivious Write

To write v^* to addr^*:



$$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{addr}^*}, (\mathsf{addr}^*, v^*))$$

$$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{ek}_{\mathsf{addr}^*}, (\mathsf{addr}^*, v^*, \sigma))$$

$$\xrightarrow{\mathsf{ctxt}}$$

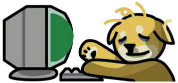
Solution: Oblivious Write

To write v^* to addr^*:

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{addr}^*}, (\mathsf{addr}^*, v^*))$

$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{ek}_{\mathsf{addr}^*}, (\mathsf{addr}^*, v^*, \sigma))$

$$\xrightarrow{\mathsf{ctxt}}$$



$\mathsf{Write}_{\mathsf{vk}_{\mathsf{addr}}}(v, (\mathsf{addr}^*, v^*, \sigma))$

if $\mathsf{Vf}(\mathsf{vk}_{\mathsf{addr}}, (\mathsf{addr}^*, v^*), \sigma) = 1$

  return v^*

else return v

foreach $\mathsf{addr} \in [n]$ **do**

  $\bar{v}'_{\mathsf{addr}} \leftarrow \mathsf{Eval}(\mathsf{ek}_{\mathsf{addr}}, \mathsf{Write}_{\mathsf{vk}_{\mathsf{addr}}}, (\bar{v}_{\mathsf{addr}}, \mathsf{ctxt}))$

return $(\bar{v}'_1, \ldots, \bar{v}'_n)$

Why it works?

Case 1: $\text{addr} = \text{addr}^*$ (Update)

- The condition $\text{Vf}(\text{vk}_{\text{addr}}, (\text{addr}^*, v^*), \sigma) = 1$ is satisfied.
- The homomorphic evaluation results in an encryption of v^* under $\text{ek}_{\text{addr}} = \text{ek}_{\text{addr}^*}$.

$\text{Write}_{\text{vk}_{\text{addr}}}(v, (\text{addr}^*, v^*, \sigma))$

if $\text{Vf}(\text{vk}_{\text{addr}}, (\text{addr}^*, v^*), \sigma) = 1$

  return v^*

else return v

Why it works?

Case 1: $\mathsf{addr} = \mathsf{addr}^*$ (Update)

- The condition $\mathsf{Vf}(\mathsf{vk}_{\mathsf{addr}}, (\mathsf{addr}^*, v^*), \sigma) = 1$ is satisfied.
- The homomorphic evaluation results in an encryption of v^* under $\mathsf{ek}_{\mathsf{addr}} = \mathsf{ek}_{\mathsf{addr}^*}$.

Case 2: $\mathsf{addr} \neq \mathsf{addr}^*$ (Rerandomize)

- Let $(\mathsf{addr}^\dagger, v^\dagger, \sigma^\dagger) = \mathsf{Dec}(\mathsf{dk}_{\mathsf{addr}}, \mathsf{ctxt})$ (most likely garbage).
- The condition $\mathsf{Vf}(\mathsf{vk}_{\mathsf{addr}}, (\mathsf{addr}^\dagger, v^\dagger), \sigma^\dagger) = 1$ is not satisfied w.h.p. (otherwise we can extract a forgery under $\mathsf{vk}_{\mathsf{addr}}$).
- By strong correctness, the homomorphic evaluation results in a (fresh) encryption of v under $\mathsf{ek}_{\mathsf{addr}}$.

$\mathsf{Write}_{\mathsf{vk}_{\mathsf{addr}}}(v, (\mathsf{addr}^*, v^*, \sigma))$

if $\mathsf{Vf}(\mathsf{vk}_{\mathsf{addr}}, (\mathsf{addr}^*, v^*), \sigma) = 1$
  return v^*
else return v

How about no FHE?

Issues

- Homomorphically evaluating “if signature verifies, then ...” circuit is costly
- FHE is only known from lattice-based assumptions

How about no FHE?

Issues

- Homomorphically evaluating “if signature verifies, then ...” circuit is costly
- FHE is only known from lattice-based assumptions

Solution: Multi-server setting

(t, m)-Distributed Point Functions (DPF)

Point functions

$$P_{\mathrm{addr}^*,\delta^*} : [n] \to \{0,1\}$$

$$\mathrm{addr} \mapsto \begin{cases} \delta^* & \mathrm{addr} = \mathrm{addr}^* \\ 0 & \mathrm{addr} \neq \mathrm{addr}^* \end{cases}$$

(t, m)-Distributed Point Function

$$(f_1, \dots, f_m) \leftarrow \mathrm{Share}(P_{\mathrm{addr}^*,\delta^*})$$

$$y_j \leftarrow \mathrm{Eval}(f_j, \mathrm{addr})$$

$$P_{\mathrm{addr}^*}(\mathrm{addr}) \leftarrow \mathrm{Dec}(y_1, \dots, y_m)$$

- t-Security: $P_{\mathrm{addr}^*,\delta^*}$ hidden given any t-subset of $\{f_1, \dots, f_m\}$
- Additive Decoding: $\mathrm{Dec}(y_1, \dots, y_m) = y_1 + \ldots + y_m$
  $\implies$ Allows linearly homomorphic evaluation

(t, m^2)-Multi-Server ORAM from (t, m)-DPF ([Bunn et al.'20]: $m = 2$)

Additive-secret-share $\mathsf{DB} \in \mathbb{F}^n$ as $(\mathsf{DB}_1, \ldots, \mathsf{DB}_m) \in \mathbb{F}^{n \times m}$ and arrange them in an m-by-m matrix:

$$\begin{matrix} \mathsf{DB}_1 & \mathsf{DB}_1 & \ldots & \mathsf{DB}_1 \\ \mathsf{DB}_2 & \mathsf{DB}_2 & \ldots & \mathsf{DB}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{DB}_m & \mathsf{DB}_m & \ldots & \mathsf{DB}_m \end{matrix}$$

Server (i, j) gets the (i, j)-th entry, *i.e.*, DB_i.

(t, m^2)-Multi-Server ORAM from (t, m)-DPF ([Bunn et al.'20]: $m = 2$)

Additive-secret-share $\mathsf{DB} \in \mathbb{F}^n$ as $(\mathsf{DB}_1, \ldots, \mathsf{DB}_m) \in \mathbb{F}^{n \times m}$ and arrange them in an m-by-m matrix:

$$\begin{matrix} \mathsf{DB}_1 & \mathsf{DB}_1 & \ldots & \mathsf{DB}_1 \\ \mathsf{DB}_2 & \mathsf{DB}_2 & \ldots & \mathsf{DB}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{DB}_m & \mathsf{DB}_m & \ldots & \mathsf{DB}_m \end{matrix}$$

Server (i, j) gets the (i, j)-th entry, *i.e.*, DB_i.

Reading addr*

- $(f_1, \ldots, f_m) \leftarrow \mathsf{Share}(P_{\mathsf{addr}^*,1})$
- Send f_j to server (i, j) for $i \in [n]$ *# Column j gets f_j*
- Server (i, j) computes $y_{i,j} \leftarrow \langle (\mathsf{Eval}(f_j, \mathsf{addr}))_{\mathsf{addr}=1}^{n}, \mathsf{DB}_i \rangle$
  # (j-th share of i-th share of $\mathsf{DB}[\mathsf{addr}^*]$)
- Client recovers $\mathsf{DB}[\mathsf{addr}^*] = \sum_{i,j} y_{i,j}$

(t, m^2)-Multi-Server ORAM from (t, m)-DPF ([Bunn et al.'20]: $m = 2$)

Additive-secret-share $\mathsf{DB} \in \mathbb{F}^n$ as $(\mathsf{DB}_1, \ldots, \mathsf{DB}_m) \in \mathbb{F}^{n \times m}$ and arrange them in an m-by-m matrix:

$$\begin{matrix} \mathsf{DB}_1 & \mathsf{DB}_1 & \ldots & \mathsf{DB}_1 \\ \mathsf{DB}_2 & \mathsf{DB}_2 & \ldots & \mathsf{DB}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{DB}_m & \mathsf{DB}_m & \ldots & \mathsf{DB}_m \end{matrix}$$

Server (i, j) gets the (i, j)-th entry, *i.e.*, DB_i.

Writing $(\mathsf{addr}^*, \delta^*)$, $\delta^* =$ change of $\mathsf{DB}[\mathsf{addr}^*]$

- $(f_1, \ldots, f_m) \leftarrow \mathsf{Share}(P_{\mathsf{addr}^*, \delta^*})$
- Send f_i to server (i, j) for $i \in [n]$ *# Row i gets f_i*
- Server (i, j) computes $\mathsf{DB}'_i \leftarrow (\mathsf{Eval}(f_i, \mathsf{addr}))_{\mathsf{addr}=1}^{n} + \mathsf{DB}_i$
  # i-th share of $\mathsf{DB}[\mathsf{addr}^*]$ shifted by i-th share of δ^*
  # Servers in the same row i compute the same DB'_i

Multi-Server MCORAM from Multi-Server ORAM

- Stateless scheme $\Longrightarrow$ Trivial extension to multi-client setting

Multi-Server MCORAM from Multi-Server ORAM

- Stateless scheme $\Longrightarrow$ Trivial extension to multi-client setting
- Read-access control by encryption (as in single-server MCORAM)
- Write-access control by zero-knowledge proofs (only achieves “accoutable integrity”)

Multi-Server MCORAM from Multi-Server ORAM

- Stateless scheme $\implies$ Trivial extension to multi-client setting
- Read-access control by encryption (as in single-server MCORAM)
- Write-access control by zero-knowledge proofs (only achieves “accoutable integrity”)
- Communication dominated by share size $|f_j|$

DPF Constructions (from Non-Lattice Assumptions)

Scheme	(t, m)	Share Size $\lvert f_j \rvert$
[Gilboa-Ishai'14, Boyle *et al.*'15,16]	$(1, 2)$	$\mathrm{polylog}(n)$
[Bunn *et al.*'20]	$(2, 3)$	$O(\sqrt{n})$

DPF Constructions (from Non-Lattice Assumptions)

Scheme	(t,m)	Share Size $\lvert f_j \rvert$
[Gilboa-Ishai'14, Boyle *et al.*'15,16]	$(1,2)$	$\mathrm{polylog}(n)$
[Bunn *et al.*'20]	$(2,3)$	$O(\sqrt{n})$
This work	$(2,3)$, $(3,4)$	$\mathrm{polylog}(n)$

- Observation: Scheme of Boyle *et al.* has NC1 evaluation circuit
- Approach: Emulate server using $(1,2)$-homomorphic secret sharing for NC1 circuits [Boyle *et al.*'16]
- Consequence: $(1,4)$-, $(2,9)$- and $(3,16)$-multi-server MCORAM with polylog communication

Also in the paper...

- Simulation-based definition of MCORAM
- Secret-key compression using constrained PRF

Conclusion

There exist multi-client ORAM with poly-logarithmic communication!

Conclusion

There exist multi-client ORAM with poly-logarithmic communication!

Conclusion

There exist multi-client ORAM with poly-logarithmic communication!

Russell W. F. Lai
Friedrich-Alexander University Erlangen-Nuremberg
russell.lai@cs.fau.de
russell-lai.hk