Outsourcing RAM Computation

Daniel Wichs
Northeastern University

Mainly based on joint works with:
Craig Gentry, Shai Halevi, Mariana Raykova
Problem Overview

- Weak client wants to leverage resources of a powerful server to compute $P(x)$ without revealing $x$.

- Efficiency Requirements:
  - Client does much less work than computing $P(x)$
  - Server does about as much work as computing $P(x)$
Use FHE! Done?

- Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,...]

- But FHE works over *circuits* rather than *RAM programs*.

I’m very efficient!
Circuits vs. RAM

• Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,...]

• But FHE works over \textit{circuits} rather than \textit{RAM programs}.
  – RAM complexity $\mathcal{T} \Rightarrow$ circuit or TM complexity $\mathcal{T}^{\#2}$
  – For programs with initial “data in memory”, efficiency gap can be exponential (e.g., Google search).

• Could use ORAM, but then client does all the work.
Goals

- **Client’s work:** $O(|x| + |y|)$
- **Server’s work:** $O(\text{RAM run-time of } P)$.

- May allow client **pre-processing** of $P$.
  - Client does **one-time computation** in $O(\text{RAM run-time of } P)$.
  - Later, outsource many executions of $P$. Amortized efficiency.
• **Basic scenario:** client wants to run independent executions of $P$ on inputs $x \downarrow 1, x \downarrow 2, x \downarrow 3, ...$

• **Persistent Memory Data:**
  – Client initially outsources large private ‘memory data’ $D$.
  – Program executions $P^{\uparrow D}(x \downarrow i)$ can read/write to $D$. 
Goals

- Non-interactive solution: “reusable garbled RAM”.
Garbled Computation

Persistent Memory Data:

Garble Data: $D \rightarrow D$

Can execute many programs with read/write access to data.

Garbled RAM

[LO13, GHLORW14, GLOS15]

Garble RAM: $P \rightarrow P$
Garble input: $x \rightarrow x$

Size of $P$, run-time $P(x)$ is $O(\text{RAM run-time } P)$.

Reusable Garbled RAM

[GHRW14, CHJV14,...]

Can garble many inputs per program.
Efficiently outsource RAM comp.
Outsourcing via Reusable G-RAM

• Client garbles program $P \rightarrow P$ \hspace{1cm} [ data $D \rightarrow D$ ].
  – Pre-processing $= O(\text{run-time } P)$

• Client repeatedly garbles inputs $x \downarrow i \rightarrow x \downarrow i$.

• Server evaluates $P$ on $x \downarrow i$ to get $y \downarrow i$ [ using $D$ ]
  – Evaluation time $= O(\text{run-time } P)$
• **Client learns** $y \downarrow i$. Server sends it back (+1 round = optimal).

• **Output privacy**: set $y \downarrow i = \text{encryption of real output}$. Server sends back $y \downarrow i$.

• **Verifiability**: $y \downarrow i$ includes (one-time) MAC of real output.
Garbled RAM

PART I

• Overview of [LO13].
• Circularity issue, fixes.

PART II

Reusable Garbled RAM

Combine:
• Non-reusable garbled RAM.
• Obfuscation.
PART I

One-Time Garbled RAM
Garbled RAM Definition

without persistent data

Client

secret: $k$

$\text{GProg}(P, k) \rightarrow P$

$\text{GInput}(x, k) \rightarrow x$

Server

$\text{Eval}(P, x) \rightarrow y$
Garbled RAM Definition

with persistent data

\[
\begin{align*}
\text{Client} & \quad \text{secret: } k \\
\text{GData} & (D,k) \rightarrow D \\
\text{GProg} & (P,k,i) \rightarrow P \downarrow i \quad \approx O(\text{run-time}) \\
\text{GInput} & (x \downarrow i, k, i) \rightarrow x \downarrow i
\end{align*}
\]

Server

\[
\begin{align*}
\text{Eva} & I \uparrow D \ (P \downarrow i, x \downarrow i) \rightarrow y \downarrow i \\
& \approx O(\text{run-time})
\end{align*}
\]

- **Security:** server only learns \(y \downarrow 1, y \downarrow 2, \ldots\) (even data access pattern is hidden!)
Weak vs. Full Security

Weak security: May reveal data $D$, and *data-access pattern* of computations.
- Locations of memory accessed in each step.
- Values read and written to memory.

• Compiler: weak $\Rightarrow$ full security:
  - Use oblivious RAM [GO96,...] to encode/access memory.
Overview of [Lu-Ostrovsky 13]

For now, read-only computation.
Memory
Data \( D = [D[1], D[2], D[3], \ldots] \)

Read location: i

CPU
Step 1

\( \text{read bit} \)

CPU
Step 2

\( \text{state} \)

\( \text{state} \)

\( \ldots \)

GProg:

Read location: $i$

GInp

CPU

Step 1

garbled circuit

CPU

Step 2

garbled circuit

state

state

garbled

garbled

read bit

\ldots
GData: \[
\begin{array}{c}
F(k_1,D[1]) \\
F(k_2,D[2]) \\
F(k_3,D[3]) \\
\vdots
\end{array}
\]

(F(k)(...)) is a PRF

GProg:

Read location: i

CPU

Step 1

garbled circuit

read bit

CPU

Step 2

garbled circuit

state

garbled

garbled

state

…
GData:

\[ F\downarrow k \left( 1, D[1] \right), \quad F\downarrow k \left( 2, D[2] \right), \quad F\downarrow k \left( 3, D[3] \right), \ldots \]

GProg:

Read location: \( i \)

\[ c\downarrow 0 = Enc \left( F\downarrow k \left( i, 0 \right), \text{label} \downarrow 0 \right), \quad c\downarrow 1 = Enc \left( F\downarrow k \left( i, 1 \right), \text{label} \downarrow 1 \right) \]

GInp:

CPU

Step 1

state

garbled circuit

PRF Key: \( k \)

garbled

CPU

Step 2

state

garbled circuit

PRF Key: \( k \)

garbled

\( F\downarrow k \left( \ldots \right) \) is a PRF
Let’s try to prove security...

State

garbled circuit

PRF Key: $k$

Step 1

$\text{CPU}$

Read location: $i$

$c_{\downarrow 0} = Enc(F_{\downarrow k}(i,0), label_{\downarrow 0}), \ c_{\downarrow 1} = Enc(F_{\downarrow k}(i,1), label_{\downarrow 1})$

$\text{read bit}$

State

garbled circuit

PRF Key: $k$

Step 2

$\text{CPU}$

$\text{...}$
Use security of 1\textsuperscript{st} garbled circuit only learn output

\[ c_0 = \text{Enc}(F \downarrow k(i,0), \text{label}_0) \]
\[ = \text{Enc}(F \downarrow k(i,1), \text{label}_1) \]
Use security of 1st garbled circuit only learn output (assume D[i]=1)

\[ c_{\downarrow 0} = \text{Enc}(F_{\downarrow k}(i,0), \text{label}_{\downarrow 0}) \]

\[ \text{label}_{\downarrow 1} \]

CPU

Step 2

garbled circuit

PRF Key: k

read bit

labels
garbled state

state

...
Use security of 2\textsuperscript{nd} garbled circuit

don’t learn
\textit{label}↓0  for read bit

don’t learn PRF key \textit{k}

Use security of Encryption/PRF

\[ c↓0 = \text{Enc}(F↓k(i,0), \text{label}↓0) \]

\textit{label}↓1

read bit

labels

\textit{state}

garbled state

CPU

Step 2

garbled circuit

PRF Key: \textit{k}

...
Circularity* Problem!

* May appear rectangular
So is it secure?

• Perhaps, but...
  – No proof.
  – No “simple” circularity assumption on one primitive.
Can we fix it? Yes!

- Fix 1: Using identity-based encryption (IBE).
  [GHLORW14]

- Fix 2: Evolving key, “key revocation” (OWF).
  [GHLORW14], [GLOS15]
PART II

Reusable Garbled RAM
Main Results

1-time Garbled RAM

+ 

Reusable Garbled Circuits (obfuscation)

Reusable Garbled RAM
Reusable Garbled RAM Definition

without persistent data

Client

secret: k

View can be simulated given $y \downarrow 1, y \downarrow 2, \ldots$

$GProg(P, k) \rightarrow P$

$\{ GInput(x \downarrow i, k) \rightarrow x \downarrow i : i=1,\ldots \}$

Server

$Eval(P, x \downarrow i) \rightarrow y \downarrow i$
Reusable Garbled RAM Definition

with persistent data

View can be simulated given $y \downarrow 1, y \downarrow 2, \ldots$

$GData(D, k) \rightarrow D$

$GProg(P, k) \rightarrow P$

$GInput(x \downarrow i, k, i) \rightarrow x \downarrow i$

$Eval^D(P, x \downarrow i) \rightarrow y \downarrow i$

Client

secret: $k$

Server
• **Construct reusable garbled RAM** by combining:
  - one-time garbled RAM \((\text{GProg1, GInput1, GEval1})\)
  - reusable garbled circuits

\[
\begin{align*}
P \downarrow \text{one}, x \downarrow \text{one} \\
\text{C}[P] \{ \\
\text{GProg1}(P; k) \rightarrow P \downarrow \text{one} \\
\text{GInput1}(x, k) \rightarrow x \downarrow \text{one} \\
\} \\
x, k
\end{align*}
\]

**Reusable Gprog** \(P \rightarrow P \downarrow \text{reuse}\)
Reusable circuit-garbling of \(C[P]\)

**Reusable Ginput** \(x \rightarrow x \downarrow\)
Choose fresh one-time key \(k\)
garble input \((x, k)\) for \(C[P]\)

• Size of \(C[P] = \text{(RAM run-time of } P)\)
• \(|\text{input}| = O(|x|)\)
• \(|\text{output}| = \text{(RAM run-time of } P)\)
• **Construct** *reusable garbled RAM* by combining:
  
  – one-time garbled RAM \((G\text{Prog}1, G\text{Input}1, GE\text{val}1)\)
  
  – reusable garbled circuits

**Problem:** In reusable garbled circuits of \([GKPVZ13]\), size of garbled input always exceeds size of circuit output.

**Unfortunately:** This is inherent. Cannot do better if want simulation security.

\[
\begin{align*}
P \downarrow \text{one}, \ x \downarrow \text{one} \\
\mathcal{C}[P] \{ \\
G\text{Prog}1(P, k) &\rightarrow P \downarrow \text{one} \\
G\text{Input}1(x, k) &\rightarrow x \downarrow \text{one} \\
\} \\
x,k
\end{align*}
\]

- Size of \(\mathcal{C}[P] = (\text{RAM run-time of } P)\)
- \(|\text{input}| = O(|x|)\)
- \(|\text{output}| = (\text{RAM run-time of } P)\)
• Construct reusable garbled RAM by combining:
  – one-time garbled RAM \((G\text{Prog}1, G\text{Input}1, G\text{Eval}1)\)
  – reusable garbled circuits

• Solution:
  – Show that we do not need simulation-security for reusable garbled-circuits. A weaker notion suffices.
  – Construct reusable garbled-circuits with weaker security notion but better efficiency needed in construction. (using indistinguishability obfuscation)
• **Theorem:** Get reusable garbled RAM where:
  – Garble, evaluate program: $O(\text{RAM run-time } P)$.
  – Garble input = $O(\text{input } + \text{output size})$.
assuming “ind. obfuscation” + stat. sound NIZK.

• **Theorem:** Get reusable garbled RAM with persistent memory where:
  – garble data = $O(\text{data size})$
  – garble program = $O(\text{description size } P )$
  – garble input = $O(\text{input } + \text{output size})$
  – evaluate = $O(\text{RAM run-time } P)$
assuming “strong differing-inputs obfuscation” (heuristic).

• **New:** can be done from ind. obf. [CHJV14] ([BGT14,KLW14,LP14])
Thank You!

Don’t turn me into a circuit!