from circuits to RAM programs in malicious-2PC

Abstract: Secure 2-party computation (2PC) is becoming practical in some domains. However, most approaches are limited by the fact that the desired functionality must be represented as a boolean circuit. In response, the random-access machine (RAM) model has recently been investigated as a promising alternative to circuits.

In this talk, I will discuss some pitfalls of basing malicious-secure 2PC on the RAM model rather than circuits. I will then describe two new protocols for malicious-secure 2PC of RAM programs, whose performance relative to the semi-honest model matches the state of the art for circuit-based 2PC techniques. For malicious security with statistical security parameter \(2^{-s}\), our protocol without preprocessing has overhead \(s\) compared to the semi-honest model; our protocol with preprocessing has overhead \(\sim 2s/\log T\), where \(T\) is the running time of the RAM program.

▶ Arash Afshar @ University of Calgary
▶ Zhangxiang Hu @ Oregon State University
▶ Payman Mohassel @ University of Calgary
▶ Mike Rosulek @ Oregon State University
secure 2pc

\[ f(x; y) \]
secure 2pc
secure 2pc

\[ f(x, y) \]

\[ f(x, y) \]
secure 2pc

\[ f(x, y) \]
“to securely evaluate $f$, first express $f$ as a boolean circuit, then ...”
“to securely evaluate $f$, first express $f$ as a boolean circuit, then ...”

AES ✓

AES S-box (need 208 of these)
“to securely evaluate $f$, first express $f$ as a boolean circuit, then ...”

AES ✓  stable marriage ✗

AES S-box (need 208 of these)
“to securely evaluate \( f \), first express \( f \) as a boolean circuit, then ...”

AES ✓

stable marriage ✗

binary search ✗

size of circuit = \( \Omega(\text{size of inputs}) \)

AES S-box (need 208 of these)
why we like (boolean) circuits

Garbled circuit technique [Yao86, BellareHoangRogaway12]
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

A_0
A_1

B_0
B_1

C_0
C_1

pick two random labels for each wire
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

randomly associate labels with true/false
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

give “encrypted truth table” for each gate
why we like (boolean) circuits

Garbled circuit technique [Yao86, BellareHoangRogaway12]

give “encrypted truth table” for each gate
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

A₀: true
A₁: false
B₀: false
B₁: true

Enc_{A₁,B₀}(C₀)
Enc_{A₁,B₁}(C₀)
Enc_{A₀,B₀}(C₀)
Enc_{A₀,B₁}(C₁)

give “encrypted truth table” for each gate
why we like (boolean) circuits

Garbled circuit technique [Yao86,BellareHoangRogaway12]

Informal security proof:

- Wire label leaks no information about logical value
- Receiver only learns one label for each wire (induction)
RAM programs

small internal state

CPU

memory

Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state

[GoldreichOstrosvky96]

Can make any RAM program oblivious, polylog overhead in runtime & memory

[ShiChanStefanovLi11, ...]

Must still "touch" all of memory, in initialization phase

Our results only need "metadata-obliviousness" (R vs W, address)
RAM programs

small internal state

CPU

Memory

read, addr_1

data_1

[GoldreichOstrosvky96]

Can make any RAM program oblivious, polylog overhead in runtime & memory

[ShiChanStefanovLi11, .....

Must still “touch” all of memory, in initialization phase

Our results only need “metadata-obliviousness” (R vs W, address)
RAM programs

```
cpu

read, addr_1

data_1

read, addr_2

data_2

memory
```

small internal state

Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state

[GoldreichOstrosvky96]

Can make any RAM program oblivious, polylog overhead in runtime & memory

[ShiChanStefanovLi11, ...]

Must still “touch” all of memory, in initialization phase

Our results only need “metadata-obliviousness” (R vs W, address)
RAM programs

small internal state

CPU

memory

read, addr_1

data_1

read, addr_2

data_2

write, addr_3, data_3

ok

Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state

[GoldreichOstrosvky96]

Can make any RAM program oblivious, polylog overhead in runtime & memory

[ShiChanStefanovLi11, .....

Must still “touch” all of memory, in initialization phase

Our results only need “metadata-obliviousness” (R vs W, address)
Oblivious RAM (ORAM) = memory access pattern leaks nothing about inputs/outputs/state [GoldreichOstrovsky96]

- Can make any RAM program oblivious, polylog overhead in runtime & memory [ShiChanStefanovLi11, .....]
- Must still “touch” all of memory, in initialization phase
- Our results only need “metadata-obliviousness” (R vs W, address)
semi-honest RAM-2PC [GKKKMRV12]

Alice

Bob

memory

initialize memory; secret share initial CPU state

initialize memory; secret share initial CPU state
semi-honest RAM-2PC [GKKKMRV12]

secure 2pc of augmented CPU circuit
semi-honest RAM-2PC \cite{GKKKMRV12}

ORAM $\Rightarrow$ safe to let Bob handle all memory access
semi-honest RAM-2PC [GKKKMRV12]

Alice

Bob

example: CPU wants to read
semi-honest RAM-2PC [GKKKMRV12]

example: CPU wants to read

example: CPU wants to write
semi-honest RAM-2PC [GKKKMRV12]

example: CPU wants to write
semi-honest RAM-2PC [GKKM09]

Alice

Bob

write, addr, data

memory[addr] := data

data

example: CPU wants to write
semi-honest RAM-2PC [GKKKMRV12]

example: CPU wants to write
malicious-security pitfalls
malicious-security pitfalls

\[ st_A \quad \oplus \quad st_B \]

read data

cpu

semi-honest
malicious 2pc
malicious-security pitfalls

Integrity of state
malicious-security pitfalls

$st_A$  $st_B$

$\oplus$

junk

$\text{CPU}$

semi-honest malicious 2pc

Integrity of state and memory!
use a MAC?
use a MAC?

This will work, but ...

\[ \text{crypto circuitry inside garbled circuit} \]
\[ \text{many inputs (MAC tags & keys) to garbled circuit} \]
use a MAC?

This will work, but ...

\[st_A\]
\[st_B\]
\[\oplus\]

Ver? Ver? Ver?

cpu

data'

Mac Mac Mac
use a MAC?

This will work, but ...

- Crypto circuitry inside garbled circuit
- Many inputs (MAC tags & keys) to garbled circuit
a better idea... 

State & memory information should be:

1. Kept private
2. Protected from tampering
a better idea...

State & memory information should be:

1. Kept private
2. Protected from tampering
a better idea...

State & memory information should be:

1. Kept private ✓
2. Protected from tampering ✓

Key Idea:

Directly reuse garbled values for state & memory!
a better idea...

State & memory information should be:

1. Kept private ✓
2. Protected from tampering ✓

Key Idea

Directly reuse garbled values for state & memory!
overview
overview

Goals:

▶ malicious security

▶ no extra overhead inside garbled circuits

▶ efficiency matching state-of-the-art for circuit-based 2PC
overview

Goals:

- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

Results:

<table>
<thead>
<tr>
<th>style</th>
<th>cost</th>
<th>technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>streaming</td>
<td>$s \times$ semi-honest</td>
<td>blind cut-and-choose, forge-and-lose</td>
</tr>
<tr>
<td>online/offline</td>
<td>$\sim 2s / \log T \times$ semi-honest</td>
<td>batched cut-and-choose, LEGO</td>
</tr>
</tbody>
</table>

for security $2^{-s}$

$T = \text{ORAM running time}$
overview

Goals:

▶ malicious security
▶ no extra overhead inside garbled circuits
▶ efficiency matching state-of-the-art for circuit-based 2PC

Results:

<table>
<thead>
<tr>
<th>style</th>
<th>cost</th>
<th>technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>streaming</td>
<td>( s \times \text{semi-honest} )</td>
<td>blind cut-and-choose, forge-and-lose</td>
</tr>
<tr>
<td>online/offline</td>
<td>( \sim 2s / \log T \times \text{semi-honest} )</td>
<td>batched cut-and-choose, LEGO</td>
</tr>
</tbody>
</table>

for security \( 2^{-s} \)

\( T = \text{ORAM running time} \)

Theme:

▶ Re-use wire labels between evaluations of garbled CPU circuit
reusing wire labels: overview

**Diagram:**
- **garbled CPU**
  - state in → data in
  - state out → data out
  - mem access

**Notes:**
- Quiz: Isn't this the same as making a monolithic garbled circuit for unrolled RAM computation?
- Connections between "data in/out" determined at runtime!
- Later inputs can depend on prior outputs
- Note: CPU need not encrypt data!
reusing wire labels: overview

$t - 1$

garbled CPU -> state out -> data out

state in -> data in

$t$

garbled CPU -> state out -> data out

state in -> data in

$t + 1$

garbled CPU

mem access

connections between "data in/out" determined at runtime!

later inputs can depend on prior outputs

Note: CPU need not encrypt data!
reusing wire labels: overview

$t - 1$

garbled CPU

state out

mem access

data out

state

t

garbled CPU

state out

data out

mem access

state in

data in

t + 1

garbled CPU

state in

data in

Quiz: isn’t this the same as making a monolithic garbled circuit for unrolled RAM computation?

connections between “data in/out” determined at runtime!
later inputs can depend on prior outputs

Note: CPU need not encrypt data!
reusing wire labels: overview

$t - 1$

garbled CPU

state

data out

mem access

data in

$t$

garbled CPU

state

data out

mem access

data in

$t + 1$

garbled CPU

Quiz: isn’t this the same as making a monolithic garbled circuit for unrolled RAM computation?

▶ connections between “data in/out” determined at runtime!

▶ later inputs can depend on prior outputs

Note: CPU need not encrypt data!
reusing wire labels: overview

$t - 1$

garbled CPU

state

data out

$\downarrow$

$\downarrow$

$\downarrow$

$t$

garbled CPU

state

data in

$\downarrow$

$\downarrow$

$\downarrow$

$t + 1$

garbled CPU

state

data in

$\downarrow$

$\downarrow$

$\downarrow$

Quiz: isn't this the same as making a monolithic garbled circuit for unrolled RAM computation?

$\Rightarrow$

Connections between "data in/out" determined at runtime!

$\Rightarrow$

Later inputs can depend on prior outputs

Note: CPU need not encrypt data!
reusing wire labels: overview

Quiz: isn't this the same as making a monolithic garbled circuit for unrolled RAM computation?

Connections between "data in/out" determined at runtime!

Later inputs can depend on prior outputs

Note: CPU need not encrypt data!
reusing wire labels: overview

Quiz: isn’t this the same as making a monolithic garbled circuit for unrolled RAM computation?
Quiz: isn’t this the same as making a monolithic garbled circuit for unrolled RAM computation?

- connections between “data in/out” determined at runtime!
- later inputs can depend on prior outputs
reusing wire labels: overview

Quiz: isn’t this the same as making a monolithic garbled circuit for unrolled RAM computation?

- connections between “data in/out” determined at runtime!
- later inputs can depend on prior outputs

Note: CPU need not encrypt data!
how to do cut-and-choose?

generate many garbled circuits
how to do cut-and-choose?

open & check some fraction of them; abort if any are bad

GC GC GC GC GC GC GC GC GC
how to do cut-and-choose?

evaluate remaining circuits; take majority output
how to do cut-and-choose?

what about in the RAM setting?

t - 1

\[ \begin{array}{cccccccc}
\text{GC} & \text{GC} & \text{GC} & \text{GC} & \text{GC} & \text{GC} & \text{GC} & \text{GC} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
\end{array} \]

t
how to do cut-and-choose?

\[
t - 1 \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \\
\text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \\
\]

\[
t \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \quad \text{GC} \\
\]

generate many garbled circuits
open & check some fraction of them; abort if any are bad
evaluate remaining circuits; take majority output

what about in the RAM setting?
check circuit
cannot share wire labels! (secrets revealed)
eval circuit must share wire labels!
can't predict check/eval when generating garbled circuits!
how to do cut-and-choose?

check circuit **cannot** share wire labels! (secrets revealed)
how to do cut-and-choose?

$\text{eval circuit } \textbf{must} \text{ share wire labels!}$

$t - 1$ \hspace{1cm} $t$

$\text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC}$

$\text{GC}$

$\text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC} \hspace{0.5cm} \text{GC}$
how to do cut-and-choose?

can’t predict check/eval when generating garbled circuits!

t – 1  

[GC]  [GC]  [GC]  [GC]  [GC]  [GC]  [GC]  [GC]  [GC]


t  

[??]  [??]  [??]  [??]  [??]  [??]  [??]  [??]  [??]  [??]  [??]
our approach

Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]

establish many threads of computation
**our approach**

Blind cut-and-choose \([\text{KamaraMohasselRiva12,KreuterShelatShen12,Mood+14}]\)

<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
<th>#10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(eval)</td>
<td>(check)</td>
<td>(eval)</td>
<td>(eval)</td>
<td>(check)</td>
<td>(check)</td>
<td>(check)</td>
<td>(eval)</td>
<td>(check)</td>
<td>(eval)</td>
</tr>
</tbody>
</table>

receiver *secretly* sets each thread to “check” or “eval”
our approach

Blind cut-and-choose [KamaraMohasselRiva12, KreuterShelatShen12, Mood+14]

<table>
<thead>
<tr>
<th>t</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
<th>#10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
<td>GC</td>
</tr>
</tbody>
</table>

sender generates garbled circuits, reusing wire labels within each thread
our approach

Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]

sender generates garbled circuits, reusing wire labels within each thread
our approach

Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]

sender generates garbled circuits, reusing wire labels within each thread
## Our Approach

**Blind cut-and-choose** \[\text{KamaraMohasselRiva12, KreuterShelatShen12, Mood+14}\]

<table>
<thead>
<tr>
<th>Thread</th>
<th>Operation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>eval</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>check</td>
<td>✓</td>
</tr>
<tr>
<td>#3</td>
<td>eval</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>eval</td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>check</td>
<td>✓</td>
</tr>
<tr>
<td>#6</td>
<td>check</td>
<td>✓</td>
</tr>
<tr>
<td>#7</td>
<td>check</td>
<td>✓</td>
</tr>
<tr>
<td>#8</td>
<td>eval</td>
<td></td>
</tr>
<tr>
<td>#9</td>
<td>eval</td>
<td></td>
</tr>
<tr>
<td>#10</td>
<td>eval</td>
<td></td>
</tr>
</tbody>
</table>

- **Check-threads**: receiver gets only enough to check
- **Eval-threads**: receiver gets only enough to eval on sender's input

Establish many threads of computation. The receiver secretly sets each thread to "check" or "eval". The sender generates garbled circuits, reusing wire labels within each thread.
our approach

Blind cut-and-choose [KamaraMohasselRiva12,KreuterShelatShen12,Mood+14]

eval-threads: receiver gets only enough to eval on sender’s input
overview of protocol #1

Cost of protocol = (# of threads) \times (cost of semi-honest)

- with traditional cut-and-choose: \sim 3s threads for security $2^{-s}$
- with [Lindell13] cheating-recovery trick: only $s$ threads
- we show how to perform [Lindell13] trick only once at the end; communication independent of RAM running time!
overview of protocol #1

Cost of protocol = (# of threads) \times (cost of semi-honest)

- with traditional cut-and-choose: \sim 3s threads for security 2^{-s}
- with [Lindell13] cheating-recovery trick: only s threads
- we show how to perform [Lindell13] trick only once at the end; communication independent of RAM running time!

Preprocessing, streaming?

- need to remember wire labels of previous circuits!
- can’t pre-process garbled circuits (wire labels have runtime dependence)
preprocessing: batched cut-and-choose

Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]
preprocessing: batched cut-and-choose

Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

generate a lot of garbled circuits
preprocessing: batched cut-and-choose

Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

open and check some fraction of them
Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
preprocessing: batched cut-and-choose

Want to do 2PC of same circuit $N$ times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
Want to do 2PC of same circuit $N$ times?  
[HuangKatzKolesnikovKumaresanMalozemoff14, LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
Want to do 2PC of same circuit $N$ times?

[ HuangKatzKolesnikovKumaresanMalozemoff14, LindellRiva14 ]

buckets of size $O(s/ \log N)$ give security $2^{-s}$
preprocessing for RAM-2PC?

Pros:

▶ RAM CPU circuit evaluated over and over!
▶ Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
▶ Pre-processing already inherent for ORAM
preprocessing for RAM-2PC?

Pros:

▶ RAM CPU circuit evaluated over and over!
▶ Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
▶ Pre-processing already inherent for ORAM

Cons:

▶ Wire-label dependence determined at runtime!
▶ Cannot re-use wire labels if all circuits garbled beforehand
preprocessing for RAM-2PC?

Pros:

- RAM CPU circuit evaluated over and over!
- Batched cut-and-choose would reduce number of garbled circuits needed (in online phase)
- Pre-processing already inherent for ORAM

Cons:

- Wire-label dependence determined at runtime!
- Cannot re-use wire labels if all circuits garbled beforehand

_If only we had a way to “connect wires on the fly” in existing garbled circuits!_
the LEGO approach!

Garble **individual gates** and connect them later

[NielsenOrlandi09,FrederiksenJakobsenNielsenNordholdOrlandi13]

- We extend the technique to circuits
- Some careful modifications are necessary
our “LEGO RAM” approach

LEGO set 6062 “Battering Ram”
xor-homomorphic commitment

\[ A \rightarrow \text{commit}(A) \]
xor-homomorphic commitment

\[\text{commit}(A) \quad \text{commit}(B) \quad \text{commit}(C)\]
xor-homomorphic commitment

\[ \text{commit}(A) \] \[ \text{commit}(B) \] \[ \text{commit}(C) \]
xor-homomorphic commitment

\[
\text{open } A
\]
xor-homomorphic commitment

\[ A \quad B \quad C \]

open \( B \oplus C \)
garbled circuit, wire labels

Each wire has a secret “parity bit”

\[
\begin{array}{c}
A_0 \\
A_1 \\
0
\end{array}
\]

\[- - - \text{encodes false}
\[- - - \text{encodes true}\]
garbled circuit, wire labels

Each wire has a secret “parity bit”

\[
\begin{array}{c}
A_0 \\
A_1 \\
1 \\
\end{array}
\]

encodes false

encodes true
soldering

connect wires of garbled circuits "on the fly"

commit: $A_0$

commit: $A_1$

commit: $0$

$\Delta_0 = A_0 B_1$

$\Delta_1 = A_1 B_0$
soldering

connect wires of garbled circuits “on the fly”

commit: $A_0$

$A_1$

0

commit: $B_0$

$B_1$

1
soldering

connect wires of garbled circuits “on the fly”

A_0
A_1
0

open:

B_0
B_1
1

xor = 1: must flip select bits
connect wires of garbled circuits “on the fly”

\[ \Delta_0 = A_0 \oplus B_1 \]

xor = 1: must flip select bits

open:
soldering

connect wires of garbled circuits “on the fly”

open:

xor = 1: must flip select bits

\[ \Delta_0 = A_0 \oplus B_1 \]

\[ \Delta_1 = A_1 \oplus B_0 \]
soldering

connect wires of garbled circuits “on the fly”

xor = 1: must flip select bits

\[ \Delta_0 = A_0 \oplus B_1 \]

\[ \Delta_1 = A_1 \oplus B_0 \]
connect wires of garbled circuits “on the fly”

xor = 1: must flip select bits

\[ \Delta_0 = A_0 \oplus B_1 \]

\[ \Delta_1 = A_1 \oplus B_0 \]
connect wires of garbled circuits “on the fly”

xor = 1: must flip select bits

\[ \Delta_0 = A_0 \oplus B_1 \]

\[ \Delta_1 = A_1 \oplus B_0 \]
putting it all together
putting it all together

generate lots of garbled CPU circuits + homomorphic commitments
putting it all together

open and check some fraction of them
putting it all together

each timestep, select random “bucket” of circuits to evaluate
putting it all together

solder input/output wires together by opening commitments
putting it all together

CPU
CPU
CPU
CPU
CPU

(evaluate by taking majority wire labels of each computation path)
Putting it all together

- St
- Read data

Solder input wires from previous garbled circuits
overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

Online cost of protocol = \( \text{bucket size} \times \text{cost of semi-honest} \)

- bucket size = \( O \left( \frac{s}{\log T} \right) \) where \( T \) = RAM running time

- concretely, for \( s = 40 \):
  - \( T = 5,000 \) bucket size = 7
  - \( T = 50,000 \) bucket size = 5

Other cool features:
- oblivious RAM = (original RAM) + (ORAM construction steps)
- pre-process different circuits separately: smaller bucket size for (common) ORAM steps
overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

Online cost of protocol = (bucket size) × (cost of semi-honest)

- bucket size = $O(s/ \log T)$ where $T =$ RAM running time
- concretely, for $s = 40$:
  - $T = 5,000 \Rightarrow$ bucket size = 7
  - $T = 500,000 \Rightarrow$ bucket size = 5
overview of protocol #2

Two phase protocol:

- offline phase = circuit generation & batch cut-and-choose
- online phase = soldering & evaluation

Online cost of protocol = (bucket size) $\times$ (cost of semi-honest)

- bucket size = $O\left(\frac{s}{\log T}\right)$ where $T$ = RAM running time
- concretely, for $s = 40$:
  - $T = 5,000 \Rightarrow$ bucket size = 7
  - $T = 500,000 \Rightarrow$ bucket size = 5

Other cool features:

- oblivious RAM = (original RAM) + (ORAM construction steps)
- pre-process different circuits separately: smaller bucket size for (common) ORAM steps
lies and omissions

Lots of other standard tools from circuit-2PC:

- Input consistency checks
- Output authenticity checks
- Preventing selective aborts
- Cheating recovery techniques

RAM stuff I didn’t mention:

- Elephant in the room: ORAM initialization!
- Getting inputs into the RAM
- ORAM needs randomness
- Safe to run many RAM invocations with same memory
summary

Goals:
- malicious security
- no extra overhead inside garbled circuits
- efficiency matching state-of-the-art for circuit-based 2PC

Results:

<table>
<thead>
<tr>
<th>style</th>
<th>cost</th>
<th>technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>streaming</td>
<td>$s \times$ semi-honest</td>
<td>blind cut-and-choose, forge-and-lose</td>
</tr>
<tr>
<td>online/offline</td>
<td>$\sim 2s / \log T \times$ semi-honest</td>
<td>batched cut-and-choose, LEGO</td>
</tr>
</tbody>
</table>

Theme:
- Leverage existing security properties of wire labels in garbled circuits!
the end!

Mike Rosulek: rosulekm@eecs.oregonstate.edu

*From Circuits to RAM Programs in Malicious-2PC*
Arash Afshar, Zhangxiang Hu, Payman Mohassel, Mike Rosulek
appearing on eprint soon