Better 2-round adaptive MPC

Ran Canetti, Oxana Poburinnaya

TAU and BU

BU
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate communication (without knowing $x_1, \ldots, x_n$)
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate communication (without knowing $x_1, \ldots, x_n$)
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate communication (without knowing $x_1, \ldots, x_n$)
Adaptive Security of MPC

Adaptive corruptions:

- adversary can decide who to corrupt adaptively during the execution

Simulator:

1. simulate communication (without knowing $x_1, \ldots, x_n$)
2. simulate $r_i$ of corrupted parties, consistent with communication and $x_i$
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:

Example: encryption

\[ c = \text{Enc}(m; r) \]
Adaptive Security of MPC

Adaptive corruptions:
adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate fake ciphertext \( c \) (without knowing \( m \))

Example: encryption
\[ c = \text{Enc}(m; r) \]
Adaptive Security of MPC

Adaptive corruptions:
- adversary can decide who to corrupt adaptively during the execution

Simulator:
1. simulate fake ciphertext c (without knowing m)
2. upon corruption, learn m and provide consistent r, sk

Example: encryption
- \( c = \text{Enc}(m; r) \)
Full Adaptive Security

Full adaptive security:
● No erasures
Full Adaptive Security

Full adaptive security:
- No erasures
- Security even when all parties are corrupted
Full Adaptive Security

**Full adaptive security:**
- No erasures
- Security even when all parties are corrupted
Full Adaptive Security

Full adaptive security:
- No erasures
- Security even when all parties are corrupted

Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15. Before: number of rounds ~ depth of the circuit (e.g. CLOS02)
Full Adaptive Security

**Full adaptive security:**
- No erasures
- Security even when all parties are corrupted

Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15. Before: number of rounds $\sim$ depth of the circuit (e.g. CLOS02)

**Full adaptive security for randomized functionalities:**
- Randomness of the computation remains hidden even when all parties are corrupted
Full Adaptive Security

Full adaptive security:
- No erasures
- Security even when all parties are corrupted

Fully adaptively secure, constant rounds protocols appeared only recently: CGP15, DKR15, GP15. Before: number of rounds $\sim$ depth of the circuit (e.g. CLOS02)

Full adaptive security for randomized functionalities:
- Randomness of the computation remains hidden even when all parties are corrupted

Example: $F$ internally chooses random primes $p, q$, and outputs $N = pq$. Most protocols (e.g. CLOS02) reveal $p, q$, when all parties are corrupted.
<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser,</td>
<td>2</td>
<td>2</td>
<td>OWF subexp iO</td>
</tr>
<tr>
<td>Poburinnaya’15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dachman-Soled, Katz,</td>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
</tr>
<tr>
<td>Rao’15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garg, Polychroniadou’</td>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Only 3 fully adaptively secure protocols with constant rounds - but with a CRS*. Only one of them is 2 round MPC.

*need a CRS even for HBC case!
### Full Adaptive Security

<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser, Poburinnaya’15</td>
<td>2</td>
<td>2</td>
<td>OWF subexp iO</td>
</tr>
<tr>
<td>Dachman-Soled, Katz, Rao’15</td>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
</tr>
<tr>
<td>Garg, Polychroniadou’15</td>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
</tr>
</tbody>
</table>

**Q1:** can we build 2 round MPC with **global (non-programmable) CRS?**
## Full Adaptive Security

<table>
<thead>
<tr>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
<th>global CRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>OWF subexp. iO</td>
<td>+</td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
<td>+</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
<td>- (even in HBC case)</td>
</tr>
</tbody>
</table>

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?
**Full Adaptive Security**

<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
<th>global CRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser,</td>
<td>2</td>
<td>2</td>
<td>OWF subexp iO</td>
<td>+</td>
</tr>
<tr>
<td>Poburinnaya’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dachman-Soled, Katz,</td>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
<td>+</td>
</tr>
<tr>
<td>Rao’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garg, Polychroniadou’15</td>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(even in HBC case)</td>
</tr>
</tbody>
</table>

Q1: can we build 2 round MPC with **global (non-programmable)** CRS?

Q2: can we compute **all randomized functionalities** (even not adaptively well formed, e.g. N = pq)?
# Full Adaptive Security

<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
<th>global CRS</th>
<th>randomized functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser,</td>
<td>2</td>
<td>2</td>
<td>OWF subexp iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Poburinnaya’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dachman-Soled, Katz,</td>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Rao’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garg, Polychroniadou’15</td>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(even in HBC case)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Q1:** can we build 2 round MPC with **global (non-programmable)** CRS?

**Q2:** can we compute **all randomized functionalities** (even not adaptively well formed, e.g. $N = pq$)?
### Full Adaptive Security

<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
<th>global CRS</th>
<th>randomized functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser, Poburinnaya’15</td>
<td>2</td>
<td>2</td>
<td>OWF, subexp iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Dachman-Soled, Katz, Rao’15</td>
<td>n</td>
<td>4</td>
<td>OWF, iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Garg, Polychroniadou’15</td>
<td>n</td>
<td>2</td>
<td>TDP, subexp. iO</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Q1:** can we build 2 round MPC with **global (non-programmable) CRS?**

**Q2:** can we compute **all randomized functionalities** (even not adaptively well formed, e.g. $N = pq$)?

**Q3:** can we build 2 round MPC from **weaker assumptions?** (e.g. remove the need for subexp. iO)
<table>
<thead>
<tr>
<th></th>
<th># of parties</th>
<th># of rounds</th>
<th>assumptions</th>
<th>global CRS</th>
<th>randomized functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canetti, Goldwasser,</td>
<td>2</td>
<td>2</td>
<td>OWF subexp iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Poburinnaya’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dachman-Soled, Katz,</td>
<td>n</td>
<td>4</td>
<td>OWF iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Rao’15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garg, Polychroniadou’15</td>
<td>n</td>
<td>2</td>
<td>TDP subexp. iO</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(even in HBC case)</td>
</tr>
<tr>
<td>This work</td>
<td>n</td>
<td>2</td>
<td>injective OWF iO</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(comp. close)</td>
</tr>
</tbody>
</table>

Q1: can we build 2 round MPC with global (non-programmable) CRS?

Q2: can we compute all randomized functionalities (even not adaptively well formed, e.g. \( N = pq \))? 

Q3: can we build 2 round MPC from weaker assumptions? (e.g. remove the need for subexp. iO)
Our results:

Part I:

**Theorem** (informal): Assuming indistinguishability obfuscation for circuits and injective one way functions, there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:
- the CRS is global;
- even randomized functionalities can be computed.
Our results:

Part I:

**Theorem** (informal): Assuming indistinguishability obfuscation for circuits and injective one way functions, there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:

- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.
Our results:

Part I:

**Theorem** (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions, there exists **2-round, fully-adaptively-secure, RAM-efficient semi-honest** MPC protocol where:
- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.

Part II:

**Theorem** (informal):
Assuming iO for circuits and TDPs, there exists **RAM-efficient statistically sound NIZK**.
Our results:

Part I:

Theorem (informal):
Assuming indistinguishability obfuscation for circuits and injective one way functions, there exists 2-round, fully-adaptively-secure, RAM-efficient semi-honest MPC protocol where:
- the CRS is global;
- even randomized functionalities can be computed.

The first two-round fully adaptive MPC without subexp. iO assumption;
The first two-round fully adaptive MPC with global CRS.

Part II:

Theorem (informal):
Assuming iO for circuits and TDPs, there exists RAM-efficient statistically sound NIZK.

Theorem (GP15, our work):
Assuming subexp. iO for circuits and RAM-efficient statistically sound NIZK, there exists 2-round, fully-adaptively-secure, RAM-efficient byzantine MPC protocol.
Part I: HBC protocol with global CRS
First attempt

\[ x_i = \text{Enc}_{PK}(x_i) \]
First attempt

\[ x_i = \text{Enc}_{PK}(x_i) \]

- decrypt each using SK
- output \( f(x_1, \ldots, x_n) \)
First attempt

\[ x_i = \text{Enc}_{PK}(x_i) \]

\[ x_1, x_2, \ldots, x_n \]

- decrypt each using SK
- output \( f(x_1, \ldots, x_n) \)

\[ y = f(x_1, x_2, \ldots, x_n) \]
First attempt

\[ x_i = \text{Enc}_{PK}(x_i) \]

\[ x'_1, x'_2, ..., x'_n \]

- decrypt each using SK
- output \( f(x_1, ..., x_n) \)

\[ y' = f(x'_1, x'_2, ..., x'_n) \]
Second attempt

\[ x_i = \text{Commit}(x_i; r_i) \]

\[ x_{i;r_i} = \text{Enc}_{PK}(x_i || r_i) \]

opening of comm

PK
Second attempt

\[ x_i = \text{Commit}(x_i; r_i) \]

\[ x_i r_i = \text{Enc}_{PK}(x_i || r_i) \]

- decrypt each using SK
- verify each
- output \( f(x_1, \ldots, x_n) \)

opening of comm
Second attempt

\[ x_i = \text{Commit}(x_i; r_i) \]
\[ x_{ri} = \text{Enc}_{PK}(x_i || r_i) \]

- decrypt each using SK
- verify each
- output \( f(x_1, ..., x_n) \)

- opening of comm

\[ y = f(x_1, x_2, ..., x_n) \]
Second attempt

\[ x_i = \text{Commit}(x_i; r_i) \]
\[ x_i r_i = \text{Enc}_{PK}(x_i || r_i) \]

- decrypt each using SK
- verify each
- output \( f(x_1, \ldots, x_n) \)

\[ y = f(x_1, x_2', \ldots, x_n) \]
Our protocol

\[
\begin{align*}
    x_i & = \text{Commit}(x_i; r_i) \\
    x_{i}r_{i} & = \text{Enc}_{PK}(x_i||r_i||\square \square \ldots \square )
\end{align*}
\]
Our protocol

\[ x_i = \text{Commit}(x_i; r_i) \]

\[ x_i r_i = \text{Enc}_{PK}(x_i || r_i || \text{...}) \]

- decrypt each using SK
- check that are the same in each
- verify each
- output \( f(x_1, \ldots, x_n) \)

\[ y = f(x_1, x_2, \ldots, x_n) \]
Our protocol

\[ x_i = \text{Commit}(x_i; r_i) \]

\[ x_i \oplus r_i \]

\[ = \text{Enc}_{PK}(x_i \| r_i \| ...) \]

- decrypt each using SK
- check that \( ... \) are the same in each
- verify each
- output \( f(x_1, ..., x_n) \)
Our protocol

\[ x_i = \text{Commit}(x_i; r_i) \]

\[ x_i \cdot r_i = \text{Enc}_{PK}(x_i || r_i || \ldots) \]

- decrypt each using SK
- check that are the same in each
- verify each
- output \( f(x_1, \ldots, x_n) \)

The adversary cannot mix and match encryptions.
Required primitives

- decrypt each \(-\) using SK
- check that \(-\) are the same in each \(-\)
- verify each \(-\)
- output \(f(x_1, \ldots, x_n)\)
Required primitives

Commitments

**Problem:**
equivocal commitments require **local** CRS

- decrypt each using SK
- check that are the same in each
- verify each
- output \( f(x_1, \ldots, x_n) \)
Required primitives

Commitments

Problem:
evocal commitments require local CRS

Solution:
semi-honest commitments (no CRS)
Com(0) = (r, prg(s)); Com(1) = (prg(s), r)

Property:
honestly generated is statistically binding.

- decrypt each using SK
- check that are the same in each
- verify each
- output f(x₁, …, xₙ)
Required primitives

**Encryption**

- **Problem:**
  cannot use security of encryption since SK is in the program

- decrypt each using SK
- check that are the same in each
- verify each
- output $f(x_1, \ldots, x_n)$
Required primitives

**Encryption**

**Problem:**
cannot use security of encryption since SK is in the program

- decrypt each \( x \) using SK
- check that \( x \) are the same in each
- verify each
- output \( f(x_1, \ldots, x_n) \)
**Required primitives**

- decrypt each $x_i$ using SK
- check that $x_i$ are the same in each
- verify each
- output $f(x_1, \ldots, x_n)$

**Encryption Problem:**
cannot use security of encryption since SK is in the program
**Required primitives**

**Problem:**
cannot use security of encryption since SK is in the program

---

**Encryption**

- decrypt each $x_i$ using SK
- check that $x_i$ are the same in each
- verify each
- output $f(x_1, \ldots, x_n)$

---

**Problem:**
cannot use security of encryption since SK is in the program

---

**Diagram:**

- Challenger
- PK
- m
- $c = \text{Enc}(m)$ or simulated $c$, SK
- GM
- PK, SK
Required primitives

Encryption

Problem:
cannot use security of encryption since SK is in the program

- decrypt each $x_i$ using SK
- check that $x_i$ are the same in each
- verify each
- output $f(x_1, \ldots, x_n)$

Encryption Problem:
cannot use security of encryption since SK is in the program

$\text{PK}$  $m$  $\text{GM}$
$\text{Challenger}$
$c = \text{Enc}(m) \text{ or simulated } c, \text{ SK}[c]$
Required primitives

Encryption

Problem:
cannot use security of encryption since SK is in the program

Solution:
Puncturable randomized encryption (PRE) (from iO and injective OWFs)

Property:
simulation-secure even when almost all SK is known

Encryption Problem:
cannot use security of encryption since SK is in the program

Solution:
Puncturable randomized encryption (PRE) (from iO and injective OWFs)

Property:
simulation-secure even when almost all SK is known
Required primitives

Encryption

**Problem:**
cannot use security of encryption since SK is in the program

**Solution:**
Puncturable randomized encryption (PRE) (from iO and injective OWFs)

**Property:**
simulation-secure even when almost all SK is known*

*: Simulation-secure analog of Sahai-Waters PDE

- decrypt each using SK
- check that are the same in each
- verify each
- output $f(x_1, \ldots, x_n)$
Achieving globality and full adaptive security

Simulation: not global
Achieving globality and full adaptive security

Simulation: not global

Solution: Modify the protocol to choose PK, during the execution.
Achieving globality and full adaptive security

Simulation: not global

Solution: Modify the protocol to choose PK, during the execution.
Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
\[ F(x_1, \ldots, x_n; r) = \text{garbled } f, \text{garbled } x_1, \ldots, x_n. \]
How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
\[ F(x_1, \ldots, x_n; r) = \text{garbled } f, \text{garbled } x_1, \ldots, x_n. \]

Any MPC protocol + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol
How to make the protocol RAM-efficient

Ishai-Kushilevitz paradigm:
use MPC to evaluate garbling:
\[ F(x_1, \ldots, x_n; r) = \text{garbled } f, \text{garbled } x_1, \ldots, x_n. \]

Any MPC protocol + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol

Only works for n-1 corruptions!
How to make the protocol RAM-efficient

**Ishai-Kushilevitz paradigm:** use MPC to evaluate garbling:
\[ F(x_1, \ldots, x_n; r) = \text{garbled } f, \text{garbled } x_1, \ldots, x_n. \]

Any MPC protocol + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol

Only works for n-1 corruptions!
For full adaptive security:

Any **randomness-hiding** MPC protocol + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol
How to make the protocol RAM-efficient: **two ways**

Our MPC protocol (which is randomness-hiding) + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol
How to make the protocol RAM-efficient: two ways

Our MPC protocol (which is randomness-hiding) + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol

or

Our MPC protocol + iO for RAM = RAM-efficient protocol

(requires subexp. iO)
Part II: Byzantine protocol and NIZK for RAM
Part II: Byzantine protocol and NIZK for RAM

Any **randomness-hiding** MPC protocol + RAM-efficient garbling (e.g. CH’16) = RAM-efficient protocol

GP’15 doesn’t compute randomness-hiding functionalities, i.e. IK02 approach doesn’t work.
Malicious case

Observation: GP’15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$. In all NIZK proofs so far: the work of verifier $\sim$ circuit size of $f$. 
Malicious case

Observation: GP’15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$.
In all NIZK proofs so far: the work of verifier $\sim$ circuit size of $f$.

**Theorem** (Garg-Polychroniadou’15):
Assuming iO for RAM, one way functions, and **NIZK proofs for RAM**, there exists **2-round, fully-adaptively-secure, RAM-efficient** MPC protocol against malicious adversaries.
Malicious case

Observation: GP’15 works with circuits only because of NIZK proof of the statement $f(x_1, \ldots, x_n) = y$. In all NIZK proofs so far: the work of verifier $\sim$ circuit size of $f$.

**Theorem** (Garg-Polychroniadou’15):
Assuming iO for RAM, one way functions, and **NIZK proofs for RAM**, there exists **2-round, fully-adaptively-secure, RAM-efficient** MPC protocol against malicious adversaries.

**Theorem** (Our work):
Assuming garbling scheme for RAM and NIZK proofs for circuits, there exists **statistically sound NIZK proof system** for RAM.
Defs: NIZK, Garbling

NIZK proof system:
Let language L be defined by relation $R(x; w)$
Prove($x$, $w$) $\rightarrow \pi$
Verify($x$, $\pi$) $\rightarrow$ accept / reject
Defs: NIZK, Garbling

**NIZK proof system:**
Let language L be defined by relation R(x; w)
Prove(x, w) → π
Verify(x, π ) → accept / reject

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of P only depends on |R|_{RAM}
- |π| only depends on |R|_{RAM}
- work of V depends on RAM complexity of R

*: everything also depends on |x|, |w|.
Defs: NIZK, Garbling

**NIZK proof system:**
Let language $L$ be defined by relation $R(x; w)$
Prove$(x, w) \rightarrow \pi$
Verify$(x, \pi) \rightarrow$ accept / reject

**Garbling scheme:**
KeyGen$(r) \rightarrow k$
GarbleProg$(k, f) \rightarrow$
GarbleInput$(k, x) \rightarrow$

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of $P$ only depends on $|R|_{\text{RAM}}$
- $|\pi|$ only depends on $|R|_{\text{RAM}}$
- work of $V$ depends on RAM complexity of $R$

*: everything also depends on $|x|, |w|$. 
NIZK proof system:
Let language $L$ be defined by relation $R(x; w)$
$\text{Prove}(x, w) \rightarrow \pi$
$\text{Verify}(x, \pi) \rightarrow \text{accept / reject}$

Garbling scheme:
$\text{KeyGen}(r) \rightarrow k$
$\text{GarbleProg}(k, f) \rightarrow f$
$\text{GarbleInput}(k, x) \rightarrow x$

Completeness;
Statistical soundness;
Zero-knowledge;
RAM-efficient*:
- work of $P$ only depends on $|R|_{\text{RAM}}$
- $|\pi|$ only depends on $|R|_{\text{RAM}}$
- work of $V$ depends on RAM complexity of $R$

Correctness: can compute $f(x)$
Security: garbled values only reveal $f(x)$
RAM-efficient*:
- work of the garbler only depends on $|f|_{\text{RAM}}$
- size of garbled values depends on $|f|_{\text{RAM}}$
- work of the evaluator depends on RAM complexity of $f$

*: everything also depends on $|x|$, $|w|$.

*: everything also depends on $|x|$.
Def: NIZK, Garbling

**NIZK proof system:**
Let language L be defined by relation R(x; w)
Prove(x, w) → ρ
Verify(x, ρ) → accept / reject

**Garbling scheme:**
KeyGen(r) → k
GarbleProg(k, f) → 
GarbleInput(k, x) → 

**Completeness;**
**Statistical soundness;**
**Zero-knowledge;**
**RAM-efficient**:  
- work of P only depends on |R|_{RAM}  
- |ρ| only depends on |R|_{RAM}  
- work of V depends on RAM complexity of R

**Correctness:** can compute f(x)
**Security:** garbled values only reveal f(x)
**RAM-efficient**:  
- work of the garbler only depends on |f|_{RAM}  
- size of garbled values depends on |f|_{RAM}  
- work of the evaluator depends on RAM complexity of f

Exists under iO for circuits + OWFs  
(Canetti-Holmgren’16)

*: everything also depends on |x|, |w|.
NIZK + Garbled RAM $\rightarrow$ NIZK for RAM

Attempt 1

Convince that $\exists w$ such that $R(x; w) = 1$

Prover

$x \in L$
$w$

Verifier

$x \in L$
NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that $\exists w$ such that $R(x; w) = 1$

KeyGen(r) → k
GarbleProg(k, R) → $R(\ast, \ast)$
GarbleInput(k, (xw)) → x, w
NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that \( \exists w \) such that \( R(x; w) = 1 \)

Proof \( \pi = R(*,*), x, w \)

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

Prover

x ∈ L
w

Verifier

x ∈ L

Accept if \( \text{Eval}(R(*,*), x, w) = 1 \)
NIZK + Garbled RAM → NIZK for RAM

Attempt 1

Convince that $\exists w$ such that $R(x; w) = 1$

Proof $\pi = R(*,*) \quad x, w$

KeyGen(r) → k
GarbleProg(k, R) → $R(*,*)$
GarbleInput(k, (xw)) → $x, w$

Verifier doesn’t learn anything about $w$

Accept if $\text{Eval}(R(*,*)\ x, w) = 1$

Prover

$x \in L$

Verifier

$x \in L$

$w$
NIZK + Garbled RAM → NIZK for RAM

Attempt 1

KeyGen(r) → k
GarbleProg(k, R) → R(*,*)
GarbleInput(k, (xw)) → x, w

Proof \( \pi = R(*,*) x, w \)

Convince that \( \exists w \) such that \( R(x; w) = 1 \)

- Verifier doesn’t learn anything about \( w \)
- Malicious prover can garble all-one function

Accept if \( \text{Eval}(R(*,*) x, w) = 1 \)
NIZK + Garbled RAM $\rightarrow$ NIZK for RAM

Attempt 2

Prover

$x \in L$

$w$

Verifier

$x \in L$

Convince that $\exists w$ such that $R(x; w) = 1$

NIZK proof: “garbling done correctly, for correct $R$ and $x$”

KeyGen($r$) $\rightarrow$ $k$

GarbleProg($k$, $R$) $\rightarrow$ $R(*,*)$

GarbleInput($k$, ($xw$)) $\rightarrow$ $R(*,*)$

$x, w$

$x, w$

Accept if $\text{Eval}(R(*,*) (x, w)) = 1$

and if NIZK verifies.
NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that \( \exists w \) such that \( R(x; w) = 1 \)

Prover

\( x \in L \)

\( w \)

Verifier

\( x \in L \)

KeyGen(r) \( \rightarrow k \)

GarbleProg(k, R) \( \rightarrow R(*,*) \)

GarbleInput(k, (xw)) \( \rightarrow x, w \)

NIZK proof: “garbling done correctly, for correct \( R \) and \( x \)”

Accept if \( \text{Eval}(R(*,*)_{x,w}) = 1 \)

and if NIZK verifies.

- Verifier doesn’t learn anything about \( w \)
- Correctness of garbling guaranteed by NIZK: idea works for perfectly correct garbling scheme for RAM
NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that \( \exists w \) such that \( R(x; w) = 1 \)

NIZK proof: “garbling done correctly, for correct \( R \) and \( x \)”

-Verifier doesn’t learn anything about \( w \)
-Correctness of garbling guaranteed by NIZK: idea works for perfectly correct garbling scheme for RAM
-Problem: don’t have perfectly correct garbling scheme for RAM

KeyGen(r) → k
GarbleProg(k, R) → \( R(*,*) \)
GarbleInput(k, (xw)) → x, w

Accept if \( \text{Eval}(R(*,*), x, w) = 1 \) and if NIZK verifies.
NIZK + Garbled RAM → NIZK for RAM

Attempt 2

Convince that $\exists w$ such that $R(x; w) = 1$

KeyGen($r$) $\rightarrow$ $k$
GarbleProg($k$, $R$) $\rightarrow$ $R(*,*)$
GarbleInput($k$, $(xw)$) $\rightarrow$ $x, w$

NIZK proof: “garbling done correctly, for correct $R$ and $x$”

Accept if $\text{Eval}(R(*,*), x, w) = 1$
and if NIZK verifies.

What might go wrong?
- Can verify that garbling was done correctly for some $r$
- cannot verify that $r$ was chosen at random
NIZK + Garbled RAM $\rightarrow$ NIZK for RAM

Attempt 2

Convince that $\exists w$ such that $R(x; w) = 1$

NIZK proof: “garbling done correctly, for correct $R$ and $x$”

KeyGen($r$) $\rightarrow k$
GarbleProg($k$, $R$) $\rightarrow R(*,*)$
GarbleInput($k$, (xw)) $\rightarrow x, w$

Accept if $\text{Eval}(R(*,*)|x, w) = 1$ and if NIZK verifies.

What might go wrong?
Consider garbling which is incorrect for one bad key $k'$:
- For $k \neq k'$ the evaluation is always correct
- for $k'$ GarbleProg always outputs all-one function.
NIZK + Garbled RAM → NIZK for RAM

Attempt 2

KeyGen(r') → k'
GarbleProg(k', R) →
GarbleInput(k', x, 0) →

output 1
x, 0

Malicious Prover
x ∉ L

NIZK proof: “garbling done correctly, for correct R and x”

Verifier accepts

x ∈ L

Accept if Eval(R(*,*) x, w) = 1

and if NIZK verifies.

What might go wrong?
Consider garbling which is incorrect for one bad key k':
- For k ≠ k' the evaluation is always correct
- for k' GarbleProg always outputs all-one function.
NIZK + Garbled RAM → NIZK for RAM

**Attempt 2**

**Convince that \( \exists w \) such that \( R(x;w) = 1 \)

**NIZK proof:** “garbling done correctly, for correct \( R \) and \( x \)”

**Verifier accepts** \( x \in L \)

Accept if \( \text{Eval}(R(*,*), x, w) = 1 \)

and if NIZK verifies.

**Crucial observation:**
the garbling scheme of CH15 is **perfectly correct with abort**, i.e.: for any key \( k \) evaluation of garbled program on garbled input wither gives correct output, or \( \bot \).
Summary: two round adaptively secure protocols

**Semi-honest case:**
- global CRS
- RAM-efficient
- computes randomized functionalities
- from iO and injective OWFs (no subexp iO)

**Malicious case** (GP15 + our RAM efficient NIZK):
- RAM-efficient
- from subexp iO and TDP
Open questions

Fully adaptive constant round HBC protocol without a CRS?
Fully adaptive constant round malicious protocol without subexp iO?
Questions?