Protecting Yao from Malicious Attacks

Mike Rosulek

crypt@b-it 2018
Yao’s Protocol Recap

garble $f(\cdot, y)$

$\langle f(x, y) \rangle$  $\triangleright$ Full security against malicious receiver

$\langle g(x) \rangle$  $\triangleright$ Malicious sender can construct bad garbled circuit (essentially the only thing that can go wrong with Yao)
Yao’s Protocol Recap

\[ \text{Yao's Protocol Recap} \]

\[ \begin{array}{c}
\text{garbled } f(\cdot, y) \\
\end{array} \]

\[ \begin{array}{c}
\text{garbled } x \\
\end{array} \]

\[ \text{OT} \]

\[ \begin{array}{c}
x \quad x \quad y \\
\end{array} \]
Yao’s Protocol Recap

- Full security against malicious receiver
- Malicious sender can construct bad garbled circuit (essentially the only thing that can go wrong with Yao)
Yao’s Protocol Recap

\[ \text{garbled } f(\cdot, y) \]

\[ \text{garbled circuit} \]

\[ \text{garbled } f(x, y) \]

\[ x \xrightarrow{\text{OT}} y \]

\[ x \xrightarrow{\text{garbled } x} \]

\[ \text{Full security against malicious receiver} \]

\[ \text{Malicious sender can construct bad garbled circuit} \]

(essentially the only thing that can go wrong with Yao)
Yao’s Protocol Recap

\[ f(x, y) \]

\[ f(\cdot, y) \]

\[ g(\cdot) \]

\[ g(x) \]

\[ f(\cdot, y) \]

\[ f(x, y) \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]

\[ X \]

\[ Y \]
Yao’s Protocol Recap

- Full security against malicious receiver
- Malicious sender can construct bad garbled circuit
Yao’s Protocol Recap

- Full security against malicious receiver
- Malicious sender can construct bad garbled circuit
  - (essentially the **only** thing that can go wrong with Yao)
Roadmap

1. Cut-and-choose:
   - Concepts & mechanisms: reducing replication factor
   - Security pitfalls & challenges

2. Dual execution: security minus 1 bit of leakage

3. Batch setting: economies of scale for repeated computations
Essence of cut-and-choose

How can you be sure that a garbled circuit was generated correctly?
Opening a garbled circuit

\[ \text{seeing all input labels} \Rightarrow \text{can check correctness of garbled gates} \]

\[ \text{This circuit no longer provides any privacy to computation!} \]

\[ \text{Can open/check a garbled circuit or use it for evaluation, not both!} \]
Opening a garbled circuit

Seeing **all input labels** ⇒ can check correctness of garbled gates

- (Better yet, give a seed to PRG that determines all input labels)
Opening a garbled circuit

Seeing all input labels ⇒ can check correctness of garbled gates

▶ (Better yet, give a seed to PRG that determines all input labels)
▶ This circuit no longer provides any privacy to computation!
▶ Can open/check a garbled circuit or use it for evaluation, not both!
Essence of cut-and-choose

Cut-and-choose approach:

1. Prepare for several independent instances of Yao’s protocol
Essence of cut-and-choose

Cut-and-choose approach:

1. Prepare for several **independent** instances of Yao’s protocol
2. **Open/check** some *random* subset of the garbled circuits
   - Abort if any garbled circuits are bad!
3. Evaluate the remaining ones normally
   - If *all* opened circuits are good, the other circuits “probably” good too
Essence of cut-and-choose

Cut-and-choose approach:
1. Prepare for several independent instances of Yao’s protocol
2. Open/check some random subset of the garbled circuits
   ▶ Abort if any garbled circuits are bad!
3. Evaluate the remaining ones normally
   ▶ If all opened circuits are good, the other circuits “probably” good too

Questions:
▶ How many instances are needed? (replication factor) How many should be opened?
▶ How to actually do this without introducing new security flaws?
All-but-one [AumannLindell07]

Garble $n$ copies
All-but-one [AumannLindell07]

Garble $n$ copies; open random $n - 1$; evaluate 1
All-but-one  \cite{AumannLindell07}

Garble \( n \) copies; open random \( n - 1 \); evaluate 1

Adversary wins \( \Leftrightarrow \) \begin{align*}
\text{all opened circuits are good} \\
\text{unopened circuit is bad}
\end{align*}
All-but-one [AumannLindell07]

Garble $n$ copies; open random $n - 1$; evaluate 1

Adversary wins $\iff$

\[
\begin{cases}
\text{all opened circuits are good} \\
\text{unopened circuit is bad}
\end{cases}
\]

$\iff$ Adv exactly predicts cut-choose challenge
All-but-one \cite{AumannLindell07}

Garble $n$ copies; open random $n - 1$; evaluate 1

Adversary \textit{wins} $\iff$
\begin{align*}
\text{all opened circuits are good} \\
\text{unopened circuit is bad}
\end{align*}

$\iff$ Adv exactly predicts cut-choose challenge

Adversary can win with probability $1/n$ (too high!)
Majority cut [LindellPinkas07]

Garble $n$ copies
Majority cut \cite{LindellPinkas07}

Garble $n$ copies; open some random subset, evaluate others
Majority cut \cite{LindellPinkas07}

Garble $n$ copies; open some random subset, evaluate others

**Questions:**
- Evaluate *several circuits* $\Rightarrow$ what if some of them disagree?
- How many circuits? How many to open?
Majority cut [LindellPinkas07]

Suppose evaluated circuits disagree
Majority cut  \[\text{[LindellPinkas07]}\]

Suppose evaluated circuits disagree

- Garbler **must** be cheating \(\Rightarrow\) Evaluator should abort!
Majority cut [LindellPinkas07]

Suppose evaluated circuits disagree

- **Garbler** must be cheating ⇒ Evaluator should abort!

**THIS IS INSECURE!**
Majority cut [LindellPinkas07]

wrong output ⇔ first bit of $x$ is 1

Suppose evaluated circuits disagree
- Garbler **must** be cheating ⇒ Evaluator should abort!

**THIS IS INSECURE!**
- Ability to *detect* cheating can depend on private input!
Majority cut [LindellPinkas07]

Suppose evaluated circuits disagree

- Garbler **must** be cheating \(\Rightarrow\) Evaluator should abort!

**THIS IS INSECURE!**

- Ability to *detect* cheating can depend on private input!
- Need another way to deal with disagreeing outputs!
Majority cut [LindellPinkas07]

Idea: Accept the **majority** output of evaluated circuits.
Idea: Accept the **majority** output of evaluated circuits.

Adversary **wins** ⇔

- **all** opened circuits are **good**
- **majority** of unopened circuits are **bad**
Majority cut  

Idea: Accept the majority output of evaluated circuits.

Adversary wins ⇔ \{ all opened circuits are good \}  \lor \{ majority of unopened circuits are bad \}

[ShelatShen11]: To ensure \( \Pr[\text{Adv wins}] < 2^{-s} \):

- Generate ~ 3.12 \( s \) circuits (replication factor)
- Open random subset of =60% of circuits
- For \( s = 40 \): generate 125 circuits and check 75
Majority cut pitfalls

Even with correct garbled circuits, computation can still go wrong!
Even with **correct garbled circuits**, computation can still go wrong!

(either party could use inconsistent inputs!)
Evaluator input consistency

How to enforce input consistency for evaluator?
Evaluator input consistency

How to enforce input consistency for evaluator?

- **Easy**: use one OT for all evaluation circuits!
Garbler input consistency

How to enforce **input consistency** for garbler?

**Idea:** [ShelatShen13] compute the function \((x, y) \mapsto f(x, y) \parallel H(y)\)

- Evaluator checks that \(H(y)\) same for majority of circuits
- \(H\) should be **collision-resistant**
- \(H\) should **hide** \(y\) (include additional randomness in \(y\) if needed)
Garbler input consistency

How to enforce **input consistency** for garbler?

**Idea:** [ShelatShen13] compute the function \((x, y) \mapsto f(x, y) \parallel H(y)\)

- Evaluator checks that \(H(y)\) same for majority of circuits
- \(H\) should be **collision-resistant**
- \(H\) should hide \(y\) (include additional randomness in \(y\) if needed)

Can arrange for \(y\) to be **committed** before \(H\) is chosen

- Can use simple 2-universal function \(H\)
- Example: \(H(y) = \) multiplication by random (public) 0/1-matrix
  \(\Rightarrow\) computation of \(H\) free using Free-XOR garbling
Majority cut pitfalls

Even with correct garbled circuits, computation can still go wrong!
Majority cut pitfalls

Even with **correct garbled circuits**, computation can still go wrong!

![Diagram showing garbled circuits and selective failure attack](image)
Majority cut pitfalls

Even with correct garbled circuits, computation can still go wrong!

Selective failure attack: Garbler sends bad input wire labels

... conditioned on receiver’s OT choice bits (her private input!)

- E.g.: junk wire label $\Leftrightarrow$ first bit of $x$ is 1
Selective failure prevention

How to avoid **selective failure** attack?

**Idea:** [LindellPinkas07,ShelatShen13] Make OT choice bits less sensitive

- Evaluate the function \((x_1, \ldots, x_k), y \mapsto f(x_1 \oplus \cdots \oplus x_k, y)\)
- Each input bit is secret-shared into *k* OT choice bits \(\Rightarrow k\)-wise independence!
Selective failure prevention

How to avoid selective failure attack?

Idea: [LindellPinkas07,ShelatShen13] Make OT choice bits less sensitive

▶ Evaluate the function \((x_1, \ldots, x_k, y) \mapsto f(x_1 \oplus \cdots \oplus x_k, y)\)

▶ Each input bit is secret-shared into \(k\) OT choice bits ⇒ \(k\)-wise independence!

Analysis:

▶ Garbler “poisons” < \(k\) OTs ⇒ evaluator failure probability independent of \(x\)

▶ Garbler “poisons” ≥ \(k\) OTs ⇒ evaluator failure probability ≥ 1 – 2\(^{-k}\)
Cheating punishment [Lindell13]

$\text{if } p \text{ is proof of cheating, then output garbler's input (else output } \perp)\text{.}$

malicious secure computation $p_y \text{ compute } f(x, y) / Q_uestion:\text{ Can we get security if only one evaluated circuit is good?}$

Idea: $\text{[Lindell13]}$

$\Rightarrow$ Contradictory output wire labels are proof of cheating

$\Rightarrow$ However, evaluator cannot reveal whether she has such proof!

$\Rightarrow$ Let her privately exchange cheating proof for garbler’s input!
Cheating punishment \cite{Lindell13}

**Question:** Can we get security if only one evaluated circuit is good?
Cheating punishment [Lindell13]

Question: Can we get security if only one evaluated circuit is good?

Idea: [Lindell13]

- Contradictory output wire labels are proof of cheating
- However, evaluator cannot reveal whether she has such proof!
Cheating punishment [Lindell13]

Question: Can we get security if only one evaluated circuit is good?

Idea: [Lindell13]

- Contradictory output wire labels are proof of cheating
- However, evaluator cannot reveal whether she has such proof!
- Let her privately exchange cheating proof for garbler’s input!
Cheating punishment: details

- Auxiliary secure computation uses majority-cut-and-choose
- Auxiliary computation depends only on input length of $f$
- Many many many optimizations to make aux computation small
- Must ensure same input $y$ to both main & aux computations
- Evaluator can learn $f(x, y)$ in two ways, but can’t reveal which!
Cheating punishment: analysis

With just one good evaluation circuit:

Case 1: All evaluation circuits agree on output

Case 2: Evaluation circuits disagree on output
Cheating punishment: analysis

If $p$ is proof of cheating, then output garbler’s input (else output $\perp$).

Malicious secure computation

With just one good evaluation circuit:

Case 1: All evaluation circuits agree on output

$\Rightarrow$ output agrees with good circuit $\Rightarrow$ output is correct

Case 2: Evaluation circuits disagree on output
Cheating punishment: analysis

With just one good evaluation circuit:

**Case 1:** All evaluation circuits agree on output
⇒ output agrees with good circuit ⇒ output is correct

**Case 2:** Evaluation circuits disagree on output
⇒ evaluator gets proof of cheating ⇒ evaluator gets correct $f(x, y)$
Cheating punishment: analysis

Adversary wins $\iff \begin{cases} \text{all opened circuits are good} \\ \text{all unopened circuits are bad} \ (\text{and agree}) \end{cases}$
Cheating punishment: analysis

Adversary wins \(\Leftrightarrow\) \(\begin{cases} \text{all opened circuits are good} \\ \text{all unopened circuits are bad (and agree)} \end{cases}\)

Suppose each circuit is checked with independent probability 1/2
Cheating punishment: analysis

if $p$ is proof of cheating, then output garbler’s input (else output $\bot$)

malicious secure computation

Adversary wins $\Leftrightarrow \{\begin{align*}
\text{all opened circuits are good} \\
\text{all unopened circuits are bad (and agree)}
\end{align*}\} \Leftrightarrow$ Adv exactly predicts cut-choose challenge

Suppose each circuit is checked with independent probability $1/2$

- With only $s$ circuits, $Pr[\text{Adv wins}] \leq 2^{-s}$ (vs. $> 3s$ circuits)
Roadmap

1. Cut-and-choose:
   - Concepts & mechanisms: reducing replication factor
   - Security pitfalls & challenges

2. Dual execution: security minus 1 bit of leakage

3. Batch setting: economies of scale for repeated computations
dual execution protocol [MohasselFranklin06]

Yao’s protocol is secure against malicious receiver
dual execution protocol \cite{MohasselFranklin06}

Yao’s protocol is secure against malicious receiver

⇒ run it in both directions!
dual execution protocol [MohasselFranklin06]

Define a common garbled encoding:

\[
\begin{align*}
\llbracket f(x, y) \rrbracket_A &= \llbracket z \rrbracket_A \\
\llbracket f(x, y) \rrbracket_B &= \llbracket z \rrbracket_B 
\end{align*}
\]

Malicious Bob can't predict \( \llbracket z \rrbracket_A, B \) for \( z, f(x, y) \) (authenticity)

Malicious Bob learns whether \( g(x) = f(x, y) \): 1 bit of leakage on \( x \)

Malicious Bob can't make Alice accept incorrect output!
dual execution protocol \cite{MohasselFranklin06}

Define a \textbf{common} garbled encoding: \([z]_{A,B} \overset{\text{def}}{=} [z]_A \oplus [z]_B\)

- Malicious Bob can't predict \([z]_{A,B}\) for \(z \neq f(x, y)\) (\textbf{authenticity})
Define a common garbled encoding: $[z]_{A,B} \overset{\text{def}}{=} [z]_A \oplus [z]_B$

Malicious Bob can’t predict $[z]_{A,B}$ for for $z \neq f(x, y)$ (authenticity)

Malicious Bob learns whether $g(x) = f(x, y)$: 1 bit of leakage on $x$
Define a **common** garbled encoding: $\llbracket z \rrbracket_{A,B} \overset{\text{def}}{=} \llbracket z \rrbracket_A \oplus \llbracket z \rrbracket_B$

- Malicious Bob can’t predict $\llbracket z \rrbracket_{A,B}$ for for $z \neq f(x, y)$ (authenticity)
- Malicious Bob learns whether $g(x) = f(x, y)$: **1 bit of leakage** on $x$
- Malicious Bob can’t make Alice accept incorrect output!
reducing leakage [KolesnikovMohasselRivaRosulek15]
reducing leakage [KolesnikovMohasselRivaRosulek15]

Main idea:
- Run s copies of Yao’s protocol in each direction
Main idea:

- Run $s$ copies of Yao’s protocol in each direction
- Cut and choose: check each garbled circuit with probability 1/2.
Main idea:

- Run $s$ copies of Yao's protocol in each direction
- Cut and choose: check each garbled circuit with probability $1/2$.
- Garbled circuits in same direction have same output encoding
reducing leakage [KolesnikovMohasselRivaRosulek15]

Main idea:

- Run $s$ copies of Yao’s protocol in each direction
- Cut and choose: check each garbled circuit with probability $1/2$.
- Garbled circuits in same direction have same output encoding
- What to do when Alice gets disagreeing outputs?
reconciliation technique

\[
\begin{array}{c}
\left[ z^* \right]_B \\
\end{array}
\begin{array}{c}
\left[ z^* \right]_A \\
\end{array}
\]

- Honest parties can compute common \( \left[ z^* \right]_{A,B} \overset{\text{def}}{=} \left[ z^* \right]_B \oplus \left[ z^* \right]_A \).
reconciliation technique

\[ [z_1]_B, [z_2]_B, \ldots \]

\[ S_A = \left\{ [z_i]_{A,B} \right\}_i \]

- Honest parties can compute common \( [z^*]_{A,B} \overset{\text{def}}{=} [z^*]_B \oplus [z^*]_A \)
- If disagreeing outputs, compute set of candidates
reconciliation technique

\[ [z_1]_B, [z_2]_B, \ldots \]
\[ S_A = \{[z_i]_{A,B}\}_i \]

\[ [z^*]_A \]
\[ S_A \cap S_B \]

- Honest parties can compute common \([z^*]_{A,B} \overset{\text{def}}{=} [z^*]_B \oplus [z^*]_A\)
- If disagreeing outputs, compute set of candidates
- Do private set intersection on the sets!
  \(\Rightarrow\) PSI output identifies the “correct” \(z_i\)
s instances of Yao in each direction, check random subset
protocol summary

\[
\begin{align*}
\{ [z_i]_A \} & \times \{ [z'_i]_A \} \quad \text{s instances of Yao in each direction, check random subset} \\
\{ [z_i]_B \} & \times \{ [z'_i]_B \} \quad \text{Compute set of reconciliation values}
\end{align*}
\]
protocol summary

\[ S_A = \{\left[ z_i \right]_{A,B} \}_{i} \]

\[ S_B = \{\left[ z_i' \right]_{A,B} \}_{i} \]

- \( s \) instances of Yao in each direction, check random subset
- Compute set of reconciliation values
- Private set intersection to identify correct output
protocol analysis

\[
\begin{align*}
S_A &= \left\{ [z_i]_{A,B} \right\}_i \\
S_B &= \left\{ [z'_i]_{A,B} \right\}_i
\end{align*}
\]

\[
\begin{align*}
S_A \cap S_B
\end{align*}
\]

Bob's only "useful" PSI input is \([z^*]_{A,B}\)
protocol analysis: corrupt Bob

\[ [ z_1 ]_B, [ z_2 ]_B, \ldots \]

\$x\$

\[ S_A = \{ [ z_i ]_{A,B} \}_i \]

\$z^* = \text{correct output}$

\[ [ z^* ]_A \]

$\text{PSI output leaks nothing!}$

$\text{Bob's only "useful" PSI input is } [ z^* ]_{A,B}$

$\forall$ all evaluation circuits bad $\Rightarrow$ PSI output leaks just 1 bit
protocol analysis: corrupt Bob

Bob's only "useful" PSI input is $[[z^*]]_{A,B}$
protocol analysis: corrupt Bob

Bob’s only “useful” PSI input is $[[z^*]]_{A,B}$

Just one good evaluation circuit $\Rightarrow$
protocol analysis: corrupt Bob

Bob’s only “useful” PSI input is $[z^*]_{A,B}$

Just one good evaluation circuit $\Rightarrow$ PSI output leaks nothing!
protocol analysis: corrupt Bob

- Bob’s only “useful” PSI input is $[z^*]_{A,B}$
- **Just one** good evaluation circuit $\Rightarrow$ PSI output leaks nothing!
- **All** evaluation circuits bad $\Rightarrow$
protocol analysis: corrupt Bob

Bob’s only “useful” PSI input is $[[z^*_1, z^*_2, \ldots, z^*_n]]_{A, B}$

- **Just one** good evaluation circuit $\Rightarrow$ PSI output leaks nothing!
- **All** evaluation circuits bad $\Rightarrow$ PSI output leaks just 1 bit
“dual-ex+PSI” summary

$s$ garbled circuits in each direction (can be done simultaneously)

Adversary cannot violate output correctness

Adversary learns a single bit with probability $2^{-s}$ — only when:

- All opened circuits are correct
- All evaluated circuits are incorrect

Example: only 10 circuits for 0.1% chance of single-bit leakage

- all other security properties hold with overwhelming probability
Roadmap

1. **Cut-and-choose:**
   - Concepts & mechanisms: reducing replication factor
   - Security pitfalls & challenges

2. **Dual execution:** security minus 1 bit of leakage

3. **Batch setting:** economies of scale for repeated computations
online/offline setting

Want to do 2PC of same circuit \( N \) times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]
online/offline setting

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

[HuangKatzKolesnikovKumaresanMalozemoff14,$\text{LindellRiva14}$]

generate a lot of garbled circuits
online/offline setting

Want to do 2PC of same circuit \( N \) times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

open and check some fraction of them
online/offline setting

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

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
online/offline setting

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

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
online/offline setting

Want to do 2PC of same circuit \( N \) times?

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
online/offline setting

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

[HuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

pick a random “bucket” of available circuits and evaluate them
online/offline setting

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

[ChuangKatzKolesnikovKumaresanMalozemoff14,LindellRiva14]

Adversary wins $\iff$

\[
\begin{cases}
\text{all opened circuits are good} \\
\text{some bucket has all/maj bad circuits}
\end{cases}
\]

- for security $1/2^s$, need $2 + O(s/\log N)$ circuits per execution
- example: $N = 1024$, $s = 40 \implies$ only 4 circuits per execution
Cut-and-choose Perspective

**Big Idea:** Generate many garbled circuits; check some, evaluate others

- Traditional approach (majority evaluation): 125 circuits
- Cheating punishment technique: 40 circuits
- Willing to tolerate $\Pr[\text{leak 1 bit}] = 0.001$: 10 circuits (each direction)
- Willing to tolerate 1 bit of leakage: 2 circuits (1 in each direction)
- Evaluating same circuit many times: 3 or 4 circuits per evaluation
Cut-and-choose Perspective

**Big Idea:** Generate many garbled circuits; check some, evaluate others

- Traditional approach (majority evaluation): **125 circuits**
- Cheating punishment technique: **40 circuits**
- Willing to tolerate $\Pr[\text{leak 1 bit}] = 0.001$: **10 circuits** (each direction)
- Willing to tolerate 1 bit of leakage: **2 circuits** (1 in each direction)
- Evaluating same circuit many times: **3 or 4 circuits** per evaluation

Other approaches:

- **LEGO:** [NielsenOrlandi09,FJNNO13,FJNT15] cut-and-choose on **individual gates**, not circuits
  - Replication factor $2 + O(s/\log N)$ but now $N = \#$ gates
  - Extra costs needed to connect gates together

- **DUPLO:** [KolesnikovNielsenRosulekTrieuTrifiletti17] cut-and-choose on **medium-size components** (between single gate and entire circuit)

- **Pool:** [ZhuHuangCassel17] maintain large fixed-size collection of garbled circuits to support **unlimited** number of evaluations