New Consistency Checks and Implementing Online/Offline Yao

Joint work with: Asaf Cohen, Moriya Farbstein and Yehuda Lindell.

Part 1:
Background

Recap: The Cut-and-Choose Technique

Input x

OTs for $X_1, \ldots, X_s$ $\longleftrightarrow$

$gc_1, gc_2, \ldots, gc_s$ $\longleftarrow$

$S \subset [s]$ $\longrightarrow$

$\{PrfKey_j\}_{j \in S}, \{Y_j\}_{j \notin S}$ $\longleftarrow$

Input y

Checks the "opened"
garbled circuits,
evaluates the rest.

2^{-s} security, while running an additional, lighter 2PC with $3s$ small garbled circuits for *cheating recovery* [Lindell-13].

Recap: Protecting Against Selective OT Attacks

Technique	**Complexity**	**Assumptions**
Cut-and-choose OT	$\mathcal{O}(ns)$ exponentiations	DDH
Randomized Encoding	$\mathcal{O}(n)$ OTs, $\mathcal{O}(ns^2)$ encryptions	Standard
$+$ OT-Ext.,Free-XOR	$\mathcal{O}(s)$ OTs, $\mathcal{O}(ns)$ encryptions	Free-XOR, etc

where n is Alice's input length, and s is a security parameter.

Recap: Protecting Against Selective OT Attacks

Technique	Complexity	Assumptions
Cut-and-choose OT	$\mathcal{O}(ns)$ exponentiations	DDH
Randomized Encoding	$\mathcal{O}(n)$ OTs, $\mathcal{O}(ns^2)$ encryptions	Standard
$+$ OT-Ext.,Free-XOR	$\mathcal{O}(s)$ OTs, $\mathcal{O}(ns)$ encryptions	Free-XOR, etc

where n is Alice's input length, and s is a security parameter.

(Open: Can we get optimal complexity with standard assumptions?)

Recap: Checking Bob's Input Consistency

Technique	Complexity	Assumption	Drawback
DDH ZK	$\mathcal{O}(ns)$ exp.	DDH	Efficiency
[Mohassel-R-13]	$\mathcal{O}(s)$ OTs, $\mathcal{O}(ns)$ encryptions	ROM	**Complicated**
[shelat-Shen-13]	$\mathcal{O}(ns)$ encryptions	Free-XOR*	**Offline/Online setting**

where n is Bob's input length, and s is a security parameter.

Recap: Checking Bob's Input Consistency

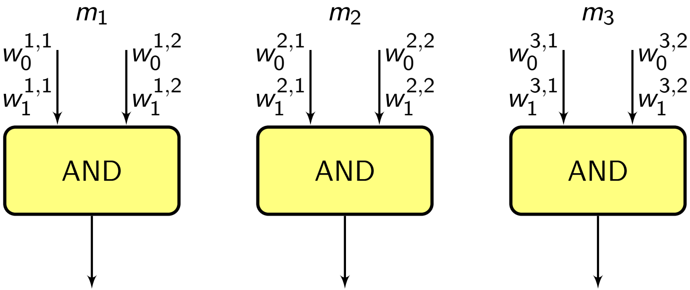
Technique	**Complexity**	**Assumption**	**Drawback**
DDH ZK	$\mathcal{O}(ns)$ exp.	DDH	Efficiency
[Mohassel-R-13]	$\mathcal{O}(s)$ OTs, $\mathcal{O}(ns)$ encryptions	ROM	**Complicated**
[shelat-Shen-13]	$\mathcal{O}(ns)$ encryptions	Free-XOR*	**Offline/Online setting**

where n is Bob's input length, and s is a security parameter.

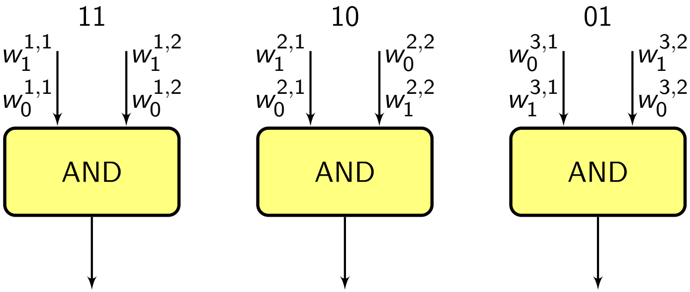
The goal is to verify consistency ONLY between good circuits!

Part 2:
A New Consistency Check

Start with Standard GCs

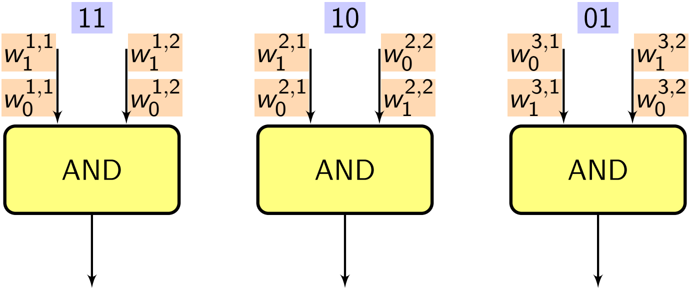


For example ...



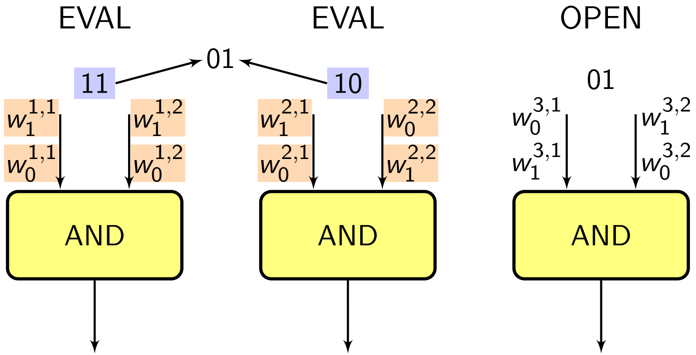
11
$w_1^{1,1}$
$w_0^{1,1}$
$w_1^{1,2}$
$w_0^{1,2}$
AND
10
$w_1^{2,1}$
$w_0^{2,1}$
$w_0^{2,2}$
$w_1^{2,2}$
AND
01
$w_0^{3,1}$
$w_1^{3,1}$
$w_1^{3,2}$
$w_0^{3,2}$
AND

Commit on Input Labels and Masks

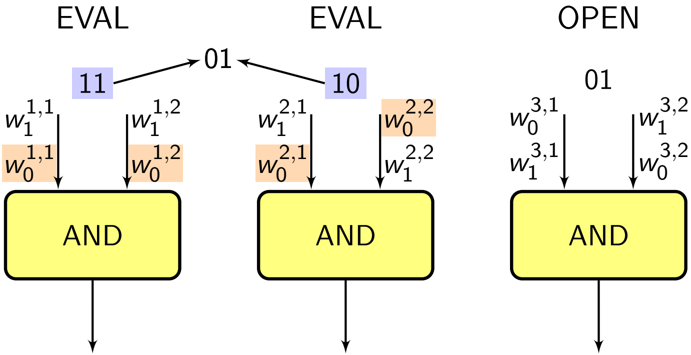


- Orange rectangle = standard commitment.
- Blue rectangle = commitment that also allows decommiting the XOR of two commitments:
  - Given $\mathsf{HCom}(m), \mathsf{HCom}(m') \rightarrow$ can decommit $m \oplus m'$.

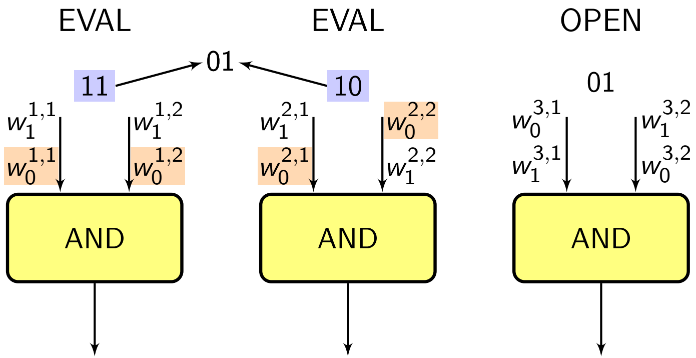
The Cut-and-Choose



Say that Bob's Input is $y = 11$



Repeating the Steps



Implementing HCom($\cdot$) Efficiently [Rabin et al-12]

Committing on $m \in \{0,1\}^n$

- Pick $m_0 \in \{0,1\}^n$ at random. Let $m_1 = m_0 \oplus m$.
- Send Com(m_0) and Com(m_1).

Implementing HCom($\cdot$) Efficiently [Rabin et al-12]

Committing on $m \in \{0,1\}^n$

- Pick $m_0 \in \{0,1\}^n$ at random. Let $m_1 = m_0 \oplus m$.
- Send Com(m_0) and Com(m_1).

Decommitting XOR of two commitments

Let m_0^1, m_1^1 and m_0^2, m_1^2 be the committed values.

- Sender sends $M_0 = m_0^1 \oplus m_0^2$ and $M_1 = m_1^1 \oplus m_1^2$.
- Receiver sends a random bit b.
- Sender decommits m_b^1 and m_b^2.
- Receiver verifies that $M_b = m_b^1 \oplus m_b^2$ and outputs $M_0 \oplus M_1$.

Improving Security of XOR Decommitment

Use k pairs of commitments for soundness 2^{-k}.
For example, with $k = 4$,

$m_0^{1,1}$	$m_1^{1,1}$	M_0^1	M_1^1	$m_0^{2,1}$	$m_1^{2,1}$
$m_0^{1,2}$	$m_1^{1,2}$	M_0^2	M_1^2	$m_0^{2,2}$	$m_1^{2,2}$
$m_0^{1,3}$	$m_1^{1,3}$	M_0^3	M_1^3	$m_0^{2,3}$	$m_1^{2,3}$
$m_0^{1,4}$	$m_1^{1,4}$	M_0^4	M_1^4	$m_0^{2,4}$	$m_1^{2,4}$

where $M_0^i \oplus M_1^i$ should be the same for $i = 1, \ldots, 4$.

Improving Security of XOR Decommitment

Use k pairs of commitments for soundness 2^{-k}.
For example, with $k = 4$,

$\boxed{m_0^{1,1}}$	$m_1^{1,1}$	M_0^1	M_1^1	$\boxed{m_0^{2,1}}$	$m_1^{2,1}$
$\boxed{m_0^{1,2}}$	$m_1^{1,2}$	M_0^2	M_1^2	$\boxed{m_0^{2,2}}$	$m_1^{2,2}$
$m_0^{1,3}$	$\boxed{m_1^{1,3}}$	M_0^3	M_1^3	$m_0^{2,3}$	$\boxed{m_1^{2,3}}$
$\boxed{m_0^{1,4}}$	$m_1^{1,4}$	M_0^4	M_1^4	$\boxed{m_0^{2,4}}$	$m_1^{2,4}$

where $M_0^i \oplus M_1^i$ should be the same for $i = 1, \ldots, 4$.

Performance and Assumptions

Performance.

When the commitment's message domain is large, the number of commitments is amortized.

For example, for security parameter $k = 40$, 80 commitments are needed. If the message domain is 80-bit long, then **the amortized number of commitments per input bit is 1!**

Performance and Assumptions

Performance.

When the commitment's message domain is large, the number of commitments is amortized.

For example, for security parameter $k = 40$, 80 commitments are needed. If the message domain is 80-bit long, then **the amortized number of commitments per input bit is 1!**

Assumptions.

Option 1: DDH and any Com (but requires additional two exponentiations per circuit).

Option 2: ROM (without exponentiations).

Other Advantages

- Much easier to implement than the method of [Mohassel-R-13].

Other Advantages

- Much easier to implement than the method of [Mohassel-R-13].
- Can be used in the Offline/Online 2PC protocols we have (as opposed to the method of [shelat-Shen-13]).

Part 3:
Implementing Online/Offline Yao

The Offline/Online Setting

Offline stage: Inputs are unknown, but we are willing to work a bit harder. (The circuit in use is known.)

Online stage(s): Inputs are known, and we wish to compute the function with minimal latency once an input arrives.

Obviously, the running time of the online stage must depend on $|C|$.

Amortized Cut-and-Choose [Lindell-R-14, Huang et al-14]

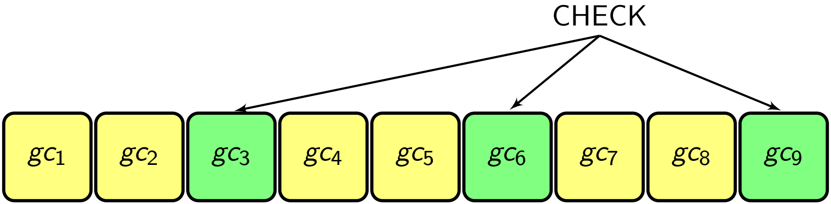
Instead of a single 2PC execution, consider N executions. This allows us to amortize the cost of the checked-circuits over many executions.

- Amortized complexity of $\mathcal{O}(\frac{s}{\log N})$ garbled circuits per invoked 2PC.
- In the ROM, the communication of the online stage is independent of $|C|$. (Very significant in practice!)

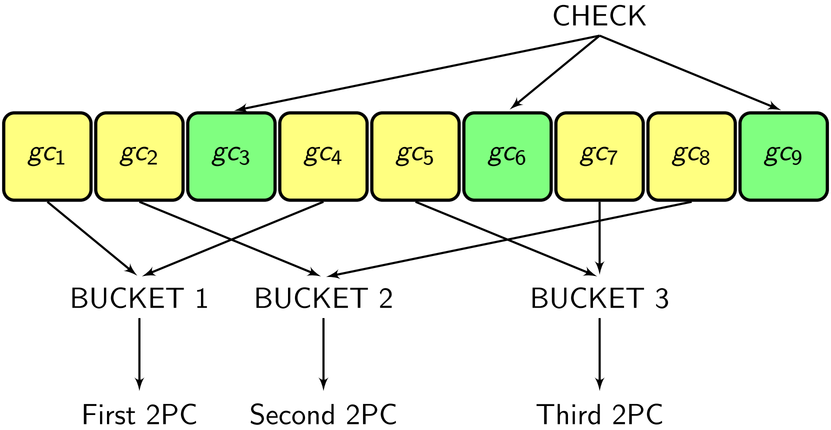
The New Cut-and-Choose: Bob Sends Many GCs

gc_1	gc_2	gc_3	gc_4	gc_5	gc_6	gc_7	gc_8	gc_9

The New Cut-and-Choose: Checking and Bucketing



The New Cut-and-Choose: Checking and Bucketing



How Many Circuits Are Needed?

N	**total #circuits**	**#eval circuit per 2PC**	**#circuits per 2PC**
10	200	11	20
32	351	8	10.96
	437	6	13.65
128	998	6	7.79
	1143	5	8.92
1024	5627	5	5.49
	5689	4	5.55
4096	18005	4	4.39
	25600	3	6.25

We can use the same technique also for checking the cheating recovery circuits.

- For 32 computations, only 30 garbled circuits are needed on average per execution.
- For 1024 computations, only 11.76 are needed.

(Recall that [Lindell-13] requires about 125 circuits.)

Prototype Implementation

- Designed a new protocol based on the protocol of [Lindell-R-14] and the new input-consistency check protocol. Heavily optimized in the ROM.
- Most steps are implemented using SCAPI. A number of CPU-intensive steps are implemented in C.
- Works with the recent OT-extension library of [Asharov et al-15] and a new library for fixed-key garbling.

Performance

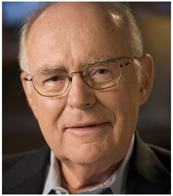
Circuit	#executions	Offline total	Offline per 2PC	Online 1	Online 4	Online 8
ADD	32	8325	260	17	15	14
	128	19787	155	10	9	11
	1024	103170	101	7	7	-
AES	32	12244	383	32	27	25
	128	30766	240	21	19	18
	1024	159144	155	16	16	14
SHA-1	32	21157	661	71	62	42
	128	55762	436	48	-	40
	1024	331192	323	37	-	27

All times are in ms. Offline is with 8 threads (and is roughly 20% – 40% slower than with a single thread).

Four Orders of Magnitude in Six Years

How much has performance of cut-and-choose 2PC improved over the years?

- 2009: 1114 seconds.
- 2011: 264 seconds.
- 2012: 1.4 seconds (cluster with 512 nodes, $s = 80$).
- 2013: 40 seconds (cluster with 8 nodes, $s = 80$).
- 2014: 0.46 seconds (using GPUs).
- 2015: < 0.2 seconds (and ≤ 32 ms for online time).

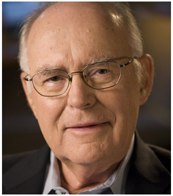


2^3!

Four Orders of Magnitude in Six Years

How much has performance of cut-and-choose 2PC improved over the years?

- 2009: 1114 seconds.
- 2011: 264 seconds.
- 2012: 1.4 seconds (cluster with 512 nodes, $s = 80$).
- 2013: 40 seconds (cluster with 8 nodes, $s = 80$).
- 2014: 0.46 seconds (using GPUs).
- 2015: < 0.2 seconds (and ≤ 32 ms for online time).



2^3!

(How much lower can Online/Offline 2PC with GPUs get us?)