Efficient Secure Three-Party Computation

Seung Geol Choi[1] and Jonathan Katz[2] and **Alex J. Malozemoff**[2] and Vassilis Zikas[3]

[1]United States Naval Academy

[2]University of Maryland, College Park

[3]University of California, Los Angeles

Presented at the Workshop on Applied Multi-Party Computation, Redmond, Washington, USA, February 20–21, 2014.

Prior Work

Setting: Malicious adversary, arbitrary # corruptions

Prior Work

Setting: Malicious adversary, arbitrary # corruptions

2PC: Many efficient constructions
(e.g., [LP07, LP11, SS11, NNOB12, HKE13, Lin13, MR13, SS13])

- Most based on Yao's garbled circuit approach [Yao82, Yao86]
  - *Boolean* circuits, $\mathcal{O}(1)$ rounds
- Use inherently two-party techniques
  - E.g., cut-and-choose, oblivious transfer, authenticated bit shares, . . .
- Fast in general (and only getting faster)

Prior Work

Setting: Malicious adversary, arbitrary # corruptions

2PC: Many efficient constructions
(e.g., [LP07, LP11, SS11, NNOB12, HKE13, Lin13, MR13, SS13])

- Most based on Yao's garbled circuit approach [Yao82, Yao86]
  - *Boolean* circuits, $\mathcal{O}(1)$ rounds
- Use inherently two-party techniques
  - E.g., cut-and-choose, oblivious transfer, authenticated bit shares, . . .
- Fast in general (and only getting faster)

MPC: SPDZ protocol [BDOZ11, DKL$^+$12, DKL$^+$13, DPSZ12, KSS13]

- *Arithmetic* circuits, $\mathcal{O}(d)$ rounds
- Total running time slow, on-line running time fast

MPC in Practice

Existing MPC deployments mostly utilize *three* parties

- The Danish sugar beet auction [BCD+09]
- Sharemind [BLW08]

MPC in Practice

Existing MPC deployments mostly utilize *three* parties

- The Danish sugar beet auction [BCD$^+$09]
- Sharemind [BLW08]

Why is this?

- Increase in communication/computation cost as # parties increases
- Settings where three parties sufficient (and two is not)

Question

Since 2PC is fast and MPC is slow(er), but 3PC seems useful in practice. . .

Question

Since 2PC is fast and MPC is slow(er), but 3PC seems useful in practice. . .

Question

Can we achieve efficient *three*-party computation using two-party tools?

In particular, can we *lift* cut-and-choose-based 2PC protocols to the three-party setting?

Contribution

Main Contribution

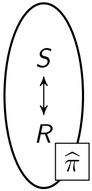
Constant-round maliciously-secure 3PC for boolean circuits at roughly *twice* the cost of underlying cut-and-choose-based 2PC used

- Tolerates arbitrary number of malicious parties
- Can lift [LP07, LP11] and [Lin13] to three-party setting
- Works in Random Oracle model
- Requires almost entirely two-party communication
  - Only three (three-party) broadcast calls needed
- Faster *start-to-finish* running time versus SPDZ
  - No implementation (yet...)
  - SPDZ has faster *on-line* running time

High-level Idea

$\widehat{\pi}(S, R)$: cut-and-choose 2PC protocol between sender S and receiver R

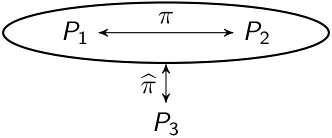
- S generates many garbling circuits using a *circuit garbling scheme*
- R does cut-and-choose on circuits



High-level Idea

We *emulate* $\widehat{\pi}$ using three parties as follows:

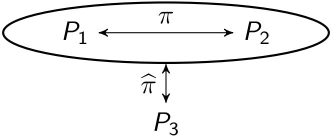
- P_1 and P_2 run two-party protocol π emulating S
  - In particular, the *circuit garbling scheme* of S
- P_3 plays role of R



High-level Idea

We *emulate* $\widehat{\pi}$ using three parties as follows:

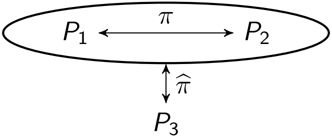
- P_1 and P_2 run two-party protocol π emulating S
  - In particular, the *circuit garbling scheme* of S
- P_3 plays role of R



Note: using "arbitrary" 2PC schemes for $\widehat{\pi}$ and π won't be efficient!

Outline of Rest of Talk

1. Distributing S's circuit garbling scheme
    1.1 (Single party) circuit garbling scheme (i.e., garbling scheme for $\widehat{\pi}$)
    1.2 Distributing the garbling scheme (i.e., π)
2. Adapting 2PC protocols (i.e., $\widehat{\pi}$) to three parties



(Single-party) Circuit Garbling Scheme

1. **Generate mask bits:**
   - For all wires w: Generate $\lambda_w \xleftarrow{\$} \{0,1\}$
2. **Generate keys:**
   - For all wires w: Generate $K_{w,0} \xleftarrow{\$} \{0,1\}^k$ and $K_{w,1} \xleftarrow{\$} \{0,1\}^k$
3. **Garble gates:**
   - For all gates G with input wires α and β and output wire γ:

$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$

(**Note:** This is standard Yao using point-and-permute)

Distributing the Garbling Scheme

Desired properties:

1. Obliviousness
   - Parties cannot know output key/tag being encrypted
2. Correctness
   - If one party malicious, garbled circuit evaluation must either:
     - Compute correct answer
     - Abort, *independent* of honest party's input

Distributing the Garbling Scheme

Desired properties:

1. Obliviousness
   - Parties cannot know output key/tag being encrypted
2. Correctness
   - If one party malicious, garbled circuit evaluation must either:
     - Compute correct answer
     - Abort, *independent* of honest party's input

Solution

Combine distributed garbling techniques [DI05] with authenticated bit shares [NNOB12]

Distributing the Garbling Scheme: Outline

- Building blocks:
  - Authenticated bit shares
  - Sub-protocols on authenticated bit shares
  - Distributed encryption scheme
- Two-party distributed circuit garbling protocol

Building Blocks: Authenticated Bit Shares [NNOB12]

- $\langle b \rangle = (\langle b \rangle^{(1)}, \langle b \rangle^{(2)})$
  - $\langle b \rangle^{(1)} = (b_1, T_1, K_2)$ and $\langle b \rangle^{(2)} = (b_2, T_2, K_1)$
  - $b = b_1 \oplus b_2$

$\mathbf{P_1}$	$\mathbf{P_2}$
$b_1,\ T_1,\ K_1$	$b_2,\ T_2,\ K_2$
$T_1 = MAC_{K_2}(b_1)$	$T_2 = MAC_{K_1}(b_2)$

- Sharing is linear:
  - $\langle b \rangle \oplus \langle b' \rangle = (\langle b \oplus b' \rangle^{(1)}, \langle b \oplus b' \rangle^{(2)})$
  - $\langle b \oplus b' \rangle^{(i)} = (b_i \oplus b'_i, T_i \oplus T'_i, K_j \oplus K'_j)$

Building Blocks: Sub-protocols on authenticated bit shares

Two-party sub-protocols:

- $\mathcal{F}^{G}_{\textbf{gate}}(\langle a \rangle, \langle b \rangle) \rightarrow \langle G(a, b) \rangle$
- $\mathcal{F}^{i}_{\textbf{oshare}}(\langle b \rangle, m_0, m_1) \rightarrow [m_b]$
  - Inputs m_0 and m_1 are *private* to party P_i
- $\mathcal{F}_{\textbf{rand}}() \rightarrow \langle b \rangle$
- $\mathcal{F}^{i}_{\textbf{ss}}(b) \rightarrow \langle b \rangle$
  - Input b is *private* to party P_i

Note: efficient maliciously secure constructions exist

- Use ideas from [NNOB12]; OT tricks

Building Blocks: Distributed Encryption Scheme [DI05]

$$[m] = m_1 \oplus m_2$$
$$K_1 = (s_1^1, s_1^2),\ K_2 = (s_2^1, s_2^2)$$

$\mathbf{P_1}$		$\mathbf{P_2}$
$m_1,\ s_1^1,\ s_2^1$		$m_2,\ s_1^2,\ s_2^2$
	$\mathsf{Enc}_{K_1,K_2}([m]) =$	
$(m_1 \oplus F^1_{s_1^1}(0) \oplus F^2_{s_2^1}(0),$		$m_2 \oplus F^1_{s_1^2}(0) \oplus F^2_{s_2^2}(0))$

- F^1 and F^2 are PRFs
- Encryption is *local*

Two-party Distributed Circuit Garbling Protocol

1. **Generate mask bits:**
   - For all wires w: Generate $\lambda_w \stackrel{\$}{\leftarrow} \{0,1\}$
2. **Generate keys:**
   - For all wires w: Generate $K_{w,0} \stackrel{\$}{\leftarrow} \{0,1\}^k$ and $K_{w,1} \stackrel{\$}{\leftarrow} \{0,1\}^k$

Two-party Distributed Circuit Garbling Protocol

1. **Generate mask bits:**
   - For all wires w: Generate $\lambda_w \stackrel{\$}{\leftarrow} \{0,1\}$
2. **Generate keys:**
   - For all wires w: Generate $K_{w,0} \stackrel{\$}{\leftarrow} \{0,1\}^k$ and $K_{w,1} \stackrel{\$}{\leftarrow} \{0,1\}^k$

Two-party Distributed Circuit Garbling Protocol

1. **Generate mask bits:**
   - P_1's input wires w: P_1 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle\lambda_w\rangle \leftarrow \mathcal{F}^1_{\mathbf{ss}}(\lambda_w)$
     P_2's input wires w: P_2 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle\lambda_w\rangle \leftarrow \mathcal{F}^2_{\mathbf{ss}}(\lambda_w)$
     All other wires w: P_1 and P_2 compute $\langle\lambda_w\rangle \leftarrow \mathcal{F}_{\mathbf{rand}}$
2. **Generate keys:**
   - For all wires w: Generate $K_{w,0} \xleftarrow{\$} \{0,1\}^k$ and $K_{w,1} \xleftarrow{\$} \{0,1\}^k$

Two-party Distributed Circuit Garbling Protocol

1. **Generate mask bits:**
   - P_1's input wires w: P_1 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle \lambda_w \rangle \leftarrow \mathcal{F}^1_{\mathbf{ss}}(\lambda_w)$
     P_2's input wires w: P_2 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle \lambda_w \rangle \leftarrow \mathcal{F}^2_{\mathbf{ss}}(\lambda_w)$
     All other wires w: P_1 and P_2 compute $\langle \lambda_w \rangle \leftarrow \mathcal{F}_{\mathbf{rand}}$
2. **Generate keys:**
   - For all wires w: Generate $K_{w,0} \xleftarrow{\$} \{0,1\}^k$ and $K_{w,1} \xleftarrow{\$} \{0,1\}^k$

Two-party Distributed Circuit Garbling Protocol

1. **Generate mask bits:**
   - P_1's input wires w: P_1 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle \lambda_w \rangle \leftarrow \mathcal{F}^1_{\mathbf{ss}}(\lambda_w)$
     P_2's input wires w: P_2 sets $\lambda_w \xleftarrow{\$} \{0,1\}$; computes $\langle \lambda_w \rangle \leftarrow \mathcal{F}^2_{\mathbf{ss}}(\lambda_w)$
     All other wires w: P_1 and P_2 compute $\langle \lambda_w \rangle \leftarrow \mathcal{F}_{\mathbf{rand}}$
2. **Generate keys:**
   - For all wires w:
     P_i, for $i \in \{1,2\}$, sets $s^i_{w,0} \xleftarrow{\$} \{0,1\}^k$ and $s^i_{w,1} \xleftarrow{\$} \{0,1\}^k$
     Let $K_{w,0} = (s^1_{w,0}, s^2_{w,0})$ and $K_{w,1} = (s^1_{w,1}, s^2_{w,1})$

Two-party Distributed Circuit Garbling Protocol

3. **Garble gates:**
   - For all gates G with input wires α and β and output wire γ:

$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma}\|G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta\oplus1)\oplus\lambda_\gamma}\|G(\lambda_\alpha,\lambda_\beta\oplus1)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha\oplus1,\lambda_\beta)\oplus\lambda_\gamma}\|G(\lambda_\alpha\oplus1,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha\oplus1,\lambda_\beta\oplus1)\oplus\lambda_\gamma}\|G(\lambda_\alpha\oplus1,\lambda_\beta\oplus1)\oplus\lambda_\gamma\right)$$

Two-party Distributed Circuit Garbling Protocol

3. **Garble gates:**
   - For all gates G with input wires α and β and output wire γ:

$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$

Two-party Distributed Circuit Garbling Protocol

3. **Garble gates:**
   - For all gates G with input wires α and β and output wire γ:

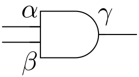
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$

Two-party Distributed Circuit Garbling Protocol

3. **Garble gates:**
   - For all gates G with input wires α and β and output wire γ:

$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta)\oplus\lambda_\gamma\right)$$
$$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(K_{\gamma,G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma} \| G(\lambda_\alpha\oplus 1,\lambda_\beta\oplus 1)\oplus\lambda_\gamma\right)$$

Example: Garbling an AND Gate



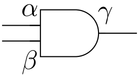
$$\lambda_\alpha = 1,\ \lambda_\beta = 0,\ \lambda_\gamma = 1$$

Standard (single-party) garbling:

Step 1: Compute tags:

i	j	$AND(\lambda_\alpha \oplus i, \lambda_\beta \oplus j) \oplus \lambda_\gamma$
0	0	$AND(1 \oplus 0, 0 \oplus 0) \oplus 1 = 1$
0	1	$AND(1 \oplus 0, 0 \oplus 1) \oplus 1 = 0$
1	0	$AND(1 \oplus 1, 0 \oplus 0) \oplus 1 = 1$
1	1	$AND(1 \oplus 1, 0 \oplus 1) \oplus 1 = 1$

Example: Garbling an AND Gate



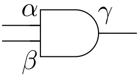
$$\lambda_\alpha = 1,\ \lambda_\beta = 0,\ \lambda_\gamma = 1$$

Standard (single-party) garbling:

Step 2: Encrypt:

i	j	
0	0	$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}(K_{\gamma,1} \| 1)$
0	1	$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}(K_{\gamma,0} \| 0)$
1	0	$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}(K_{\gamma,1} \| 1)$
1	1	$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}(K_{\gamma,1} \| 1)$

Example: Garbling an AND Gate



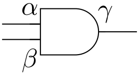
$$\langle \lambda_\alpha \rangle = 1,\ \langle \lambda_\beta \rangle = 0,\ \langle \lambda_\gamma \rangle = 1$$

Distributed garbling:

Step 1: Compute *oblivious sharings* of tags:

i	j	$\langle AND(\lambda_\alpha \oplus i, \lambda_\beta \oplus j) \oplus \lambda_\gamma \rangle$
0	0	$\mathcal{F}_{\mathbf{gate}}^{AND}(\langle 1 \rangle \oplus \langle 0 \rangle, \langle 0 \rangle \oplus \langle 0 \rangle) \oplus \langle 1 \rangle = \langle 1 \rangle$
0	1	$\mathcal{F}_{\mathbf{gate}}^{AND}(\langle 1 \rangle \oplus \langle 0 \rangle, \langle 1 \rangle \oplus \langle 1 \rangle) \oplus \langle 1 \rangle = \langle 0 \rangle$
1	0	$\mathcal{F}_{\mathbf{gate}}^{AND}(\langle 1 \rangle \oplus \langle 1 \rangle, \langle 0 \rangle \oplus \langle 0 \rangle) \oplus \langle 1 \rangle = \langle 1 \rangle$
1	1	$\mathcal{F}_{\mathbf{gate}}^{AND}(\langle 1 \rangle \oplus \langle 1 \rangle, \langle 0 \rangle \oplus \langle 1 \rangle) \oplus \langle 1 \rangle = \langle 1 \rangle$

Example: Garbling an AND Gate



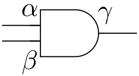
$$\langle \lambda_\alpha \rangle = 1,\ \langle \lambda_\beta \rangle = 0,\ \langle \lambda_\gamma \rangle = 1$$

Distributed garbling:

Step 2: Compute *oblivious sharings* of each party's output sub-keys:

i	j		
0	0	$\mathcal{F}^1_{\text{oshare}}(\langle 1 \rangle, s^1_{\gamma,0}, s^1_{\gamma,1}) = [s^1_{\gamma,1}]$	$\mathcal{F}^2_{\text{oshare}}(\langle 1 \rangle, s^2_{\gamma,0}, s^2_{\gamma,1}) = [s^2_{\gamma,1}]$
0	1	$\mathcal{F}^1_{\text{oshare}}(\langle 0 \rangle, s^1_{\gamma,0}, s^1_{\gamma,1}) = [s^1_{\gamma,0}]$	$\mathcal{F}^2_{\text{oshare}}(\langle 0 \rangle, s^2_{\gamma,0}, s^2_{\gamma,1}) = [s^2_{\gamma,0}]$
1	0	$\mathcal{F}^1_{\text{oshare}}(\langle 1 \rangle, s^1_{\gamma,0}, s^1_{\gamma,1}) = [s^1_{\gamma,1}]$	$\mathcal{F}^2_{\text{oshare}}(\langle 1 \rangle, s^2_{\gamma,0}, s^2_{\gamma,1}) = [s^2_{\gamma,1}]$
1	1	$\mathcal{F}^1_{\text{oshare}}(\langle 1 \rangle, s^1_{\gamma,0}, s^1_{\gamma,1}) = [s^1_{\gamma,1}]$	$\mathcal{F}^2_{\text{oshare}}(\langle 1 \rangle, s^2_{\gamma,0}, s^2_{\gamma,1}) = [s^2_{\gamma,1}]$

Example: Garbling an AND Gate



$$\langle\lambda_\alpha\rangle = 1,\ \langle\lambda_\beta\rangle = 0,\ \langle\lambda_\gamma\rangle = 1$$

Distributed garbling:

Step 3: Use *distributed* encryption to encrypt:

i	j	
0	0	$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,0}}\left(\left[s^1_{\gamma,1}\right] \parallel \left[s^2_{\gamma,1}\right] \parallel \langle 1\rangle\right)$
0	1	$\mathsf{Enc}_{K_{\alpha,0},K_{\beta,1}}\left(\left[s^1_{\gamma,0}\right] \parallel \left[s^2_{\gamma,0}\right] \parallel \langle 0\rangle\right)$
1	0	$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,0}}\left(\left[s^1_{\gamma,1}\right] \parallel \left[s^2_{\gamma,1}\right] \parallel \langle 1\rangle\right)$
1	1	$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(\left[s^1_{\gamma,1}\right] \parallel \left[s^2_{\gamma,1}\right] \parallel \langle 1\rangle\right)$

3PC Using Distributed Garbled Circuits

High-level Idea

- Take existing cut-and-choose protocol (e.g., [LP07, LP11, Lin13])
- Replace sender's circuit generation by distributed circuit generation

(Many details ignored here. . .)

3PC Using Distributed Garbled Circuits

High-level Idea

- Take existing cut-and-choose protocol (e.g., [LP07, LP11, Lin13])
- Replace sender's circuit generation by distributed circuit generation

(Many details ignored here...)

Security Intuition

- Exactly one of P_1 *or* P_2 malicious: garbled circuits either correct or abort independent of input, even with malicious P_3
- Both P_1 *and* P_2 malicious: cut-and-choose by P_3 detects cheating

3PC Using Distributed Garbled Circuits

Efficiency versus underlying 2PC protocol:

- Roughly *two times* more expensive in computation
- Roughly *three times* more expensive in communication

Approach works for several cut-and-choose-based 2PC protocols:

- ✓: Combination of [LP07, LP11] (probably [SS11, KsS12] as well)
- ✓: [Lin13]
- X: [HKE13] and [MR13], due to symmetry between P_1 and P_2

Summary

Can “lift” cut-and-choose-based 2PC to 3PC setting

- Only *twice* as slow as underlying 2PC protocol
- Only three broadcast calls needed
  - Important since broadcast expensive in WAN setting

Work still needs to be done to determine *empirical* efficiency

- Free-XOR? (**very important in practice!**)
- Implementation? Many engineering issues to consider

Paper to be published on ePrint shortly!

Thank you

Extra slides. . .

3PC Using Distributed Garbled Circuits

- Two main challenges of cut-and-choose:
  1. *Input Inconsistency*
     - Malicious generator (either P_1 or P_2) inputs inconsistent sub-keys in two different circuits; P_3 evaluates on different inputs
     - *Solution:* apply Diffie-Hellman pseudorandom synthesizer trick [LP11, MF06]
  2. *Selective Failure*
     - Sender in OT can input invalid keys, potentially learning bit of P_3's input
     - *Solution:* "XOR-tree" approach [LP07, Woo07]

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits
5. P_1/P_2 run (separately) OT protocol with P_3 for each of P_3's inputs; P_1/P_2 input sub-keys and P_3 chooses based on its input

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits
5. P_1/P_2 run (separately) OT protocol with P_3 for each of P_3's inputs; P_1/P_2 input sub-keys and P_3 chooses based on its input
6. P_1/P_2 send (distributed) garbled circuits, along with input consistency commitments, to P_3

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits
5. P_1/P_2 run (separately) OT protocol with P_3 for each of P_3's inputs; P_1/P_2 input sub-keys and P_3 chooses based on its input
6. P_1/P_2 send (distributed) garbled circuits, along with input consistency commitments, to P_3
7. $P_1/P_2/P_3$ run coin-tossing protocol to determine which circuits to open and which to evaluate

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits
5. P_1/P_2 run (separately) OT protocol with P_3 for each of P_3's inputs; P_1/P_2 input sub-keys and P_3 chooses based on its input
6. P_1/P_2 send (distributed) garbled circuits, along with input consistency commitments, to P_3
7. $P_1/P_2/P_3$ run coin-tossing protocol to determine which circuits to open and which to evaluate
8. For check circuits: P_1/P_2 send required info for P_3 to decrypt and verify correctness

3PC Using Distributed Garbled Circuits

Based on [LP07, LP11]:

1. Parties replace input circuit C^0 with a circuit C using "XOR-tree" approach for P_3's input wires
2. P_1/P_2 generate commitments for input consistency, as in [LP11]
3. P_1/P_2 construct s garbled circuits using distributed garbling protocol
4. P_1/P_2 compute authenticated sharings of input bits
5. P_1/P_2 run (separately) OT protocol with P_3 for each of P_3's inputs; P_1/P_2 input sub-keys and P_3 chooses based on its input
6. P_1/P_2 send (distributed) garbled circuits, along with input consistency commitments, to P_3
7. $P_1/P_2/P_3$ run coin-tossing protocol to determine which circuits to open and which to evaluate
8. For check circuits: P_1/P_2 send required info for P_3 to decrypt and verify correctness
9. For evaluation circuits: P_1/P_2 send sub-keys and selector bits to P_3; P_3 checks input consistency using ZKPoK as in [LP11]; evaluates circuits, outputting majority output