Efficient Three-Party Computation from Cut-and-Choose

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

[1]United States Naval Academy

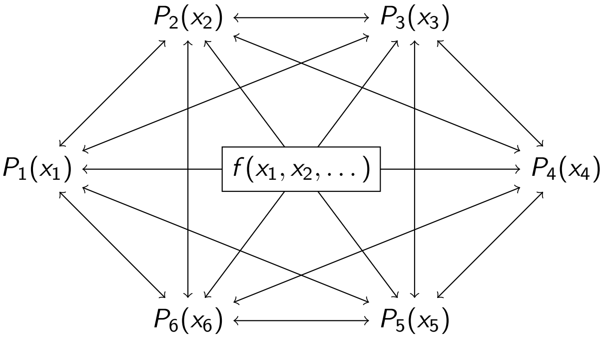
[2]University of Maryland

[3]ETH Zurich

Presented at CRYPTO, Santa Barbara, California, USA, August 17–21, 2014.

Background

Secure Computation: Parties P_1, P_2, ..., P_n compute some (common) function $f(x_1, x_2, \ldots, x_n)$ while keeping x_1, x_2, ..., x_n private, *even if $n - 1$ parties are corrupt!*



Note: Interested in **malicious** security, where adversaries can deviate *arbitrarily*

Secure Computation: 2PC vs. MPC

Considered separately in the literature:

2PC

- Two parties, 1 corruption
- Many efficient constructions
- Most based on *garbled circuits*
  - Boolean circuits
  - $\mathcal{O}(1)$ rounds
  - Preprocessing time: **none**
  - Online time: **fast**

MPC

- n parties, $\leq n-1$ corrupt
- Fewer efficient constructions
- Most efficient scheme: SPDZ
  - Arithmetic circuits
  - $\mathcal{O}(\text{depth})$ rounds
  - Preprocessing time: **slow**
  - Online time: **fast**

Secure Computation: 2PC vs. MPC

Considered separately in the literature:

2PC

- Two parties, 1 corruption
- Many efficient constructions
- Most based on *garbled circuits*
  - Boolean circuits
  - $\mathcal{O}(1)$ rounds
  - Preprocessing time: **none**
  - Online time: **fast**

MPC

- n parties, $\leq n-1$ corrupt
- Fewer efficient constructions
- Most efficient scheme: SPDZ
  - Arithmetic circuits
  - $\mathcal{O}(\text{depth})$ rounds
  - Preprocessing time: **slow**
  - Online time: **fast**

Question: Say we want to do secure computation with (fixed) $\boldsymbol{n > 2}$.
Do we need all the MPC machinery?

Secure Computation: 2PC vs. MPC

Considered separately in the literature:

2PC

- Two parties, 1 corruption
- Many efficient constructions
- Most based on *garbled circuits*
  - Boolean circuits
  - $\mathcal{O}(1)$ rounds
  - Preprocessing time: **none**
  - Online time: **fast**

MPC

- n parties, $\leq n - 1$ corrupt
- Fewer efficient constructions
- Most efficient scheme: SPDZ
  - Arithmetic circuits
  - $\mathcal{O}(\text{depth})$ rounds
  - Preprocessing time: **slow**
  - Online time: **fast**

Question: Say we want to do secure computation with $\boldsymbol{n = 3}$.
Do we need all the MPC machinery?

Three-Party Computation: Challenges

1. Not 2PC, so not clear that two-party protocols/ideas apply
   - e.g., cut-and-choose, oblivious transfer, authenticated bits

2. Do not want to resort to complexity/cost of full MPC
   - Only need efficiency for *three* parties, not arbitrary n

Contribution

Main Contribution

Efficient $\mathcal{O}(1)$-round maliciously-secure 3PC protocol for Boolean circuits

Contribution

Main Contribution

Efficient $\mathcal{O}(1)$-round maliciously-secure 3PC protocol for Boolean circuits

- Tolerates ≤ 2 (static) corruptions

Contribution

Main Contribution

Efficient $\mathcal{O}(1)$-round maliciously-secure 3PC protocol for Boolean circuits

- Tolerates ≤ 2 (static) corruptions
- Lifts existing cut-and-choose 2PC schemes to three-party setting
    - Roughly $8\times$ more expensive than underlying 2PC scheme

Contribution

Main Contribution

Efficient $\mathcal{O}(1)$-round maliciously-secure 3PC protocol for Boolean circuits

- Tolerates ≤ 2 (static) corruptions
- Lifts existing cut-and-choose 2PC schemes to three-party setting
  - Roughly $8\times$ more expensive than underlying 2PC scheme
- Requires almost entirely two-party communication
  - Only *three* broadcasts needed
    - Existing schemes require broadcast in every round

Contribution

Main Contribution

Efficient $\mathcal{O}(1)$-round maliciously-secure 3PC protocol for Boolean circuits

- Tolerates ≤ 2 (static) corruptions
- Lifts existing cut-and-choose 2PC schemes to three-party setting
  - Roughly $8\times$ more expensive than underlying 2PC scheme
- Requires almost entirely two-party communication
  - Only *three* broadcasts needed
    - Existing schemes require broadcast in every round
- Faster *start-to-finish* running time versus SPDZ
  - SPDZ has faster *on-line* running time

Recall: Cut-and-Choose

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

Semi-Honest 2PC (High-Level Idea):

Garble(C) $\longrightarrow$

Recall: Cut-and-Choose

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

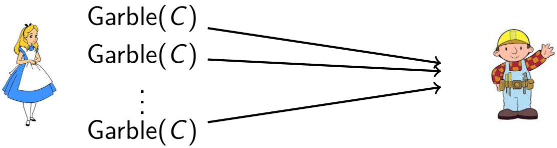
Semi-Honest 2PC (High-Level Idea):

Garble(C') $\longrightarrow$

Recall: Cut-and-Choose

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

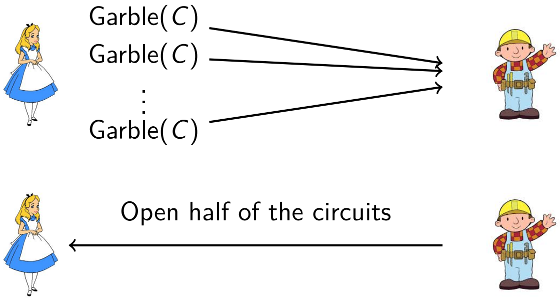
Cut-and-Choose (High-Level Idea) [LP07]:



Recall: Cut-and-Choose

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

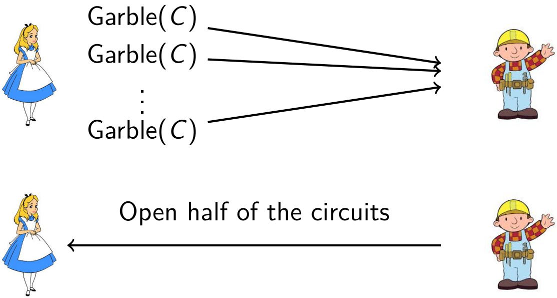
Cut-and-Choose (High-Level Idea) [LP07]:



Recall: Cut-and-Choose

Cut-and-Choose: Lifts semi-honest 2PC schemes to malicious security

Cut-and-Choose (High-Level Idea) [LP07]:



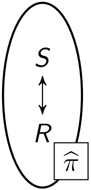
If "checked" circuits constructed correctly, w.h.p. majority of unopened garbled circuits constructed correctly

3PC: High-level Idea

How to lift cut-and-choose 2PC protocol to three-party setting:

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

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

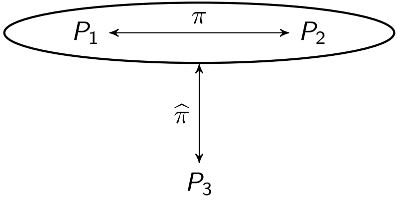


3PC: High-level Idea

How to lift cut-and-choose 2PC protocol to three-party setting:

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

- 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

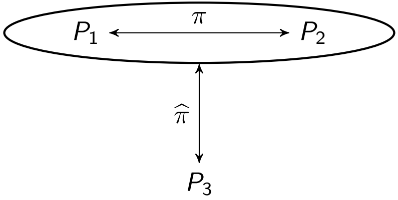


3PC: High-level Idea

How to lift cut-and-choose 2PC protocol to three-party setting:

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

- 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 generic 2PC schemes for $\widehat{\pi}$ and π not efficient!

3PC: Main Steps

Two main steps:

3PC: Main Steps

Two main steps:

1. Distribute S's circuit garbling scheme between two parties

3PC: Main Steps

Two main steps:

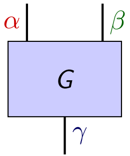
1. Distribute S's circuit garbling scheme between two parties
2. Modify existing 2PC protocol to use distributed circuit garbling scheme

3PC: Main Steps

Two main steps:

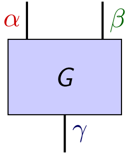
1. Distribute S's circuit garbling scheme between two parties
2. Modify existing 2PC protocol to use distributed circuit garbling scheme

Recall: (Single-Party) Circuit Garbling Scheme



Wire	Keys		Mask Bit
α	$K_{\alpha,0}$	$K_{\alpha,1}$	λ_α
β	$K_{\beta,0}$	$K_{\beta,1}$	λ_β
γ	$K_{\gamma,0}$	$K_{\gamma,1}$	λ_γ

Recall: (Single-Party) Circuit Garbling Scheme



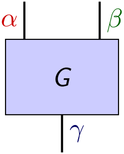
Wire	Keys		Mask Bit
α	$K_{\alpha,0}$	$K_{\alpha,1}$	λ_α
β	$K_{\beta,0}$	$K_{\beta,1}$	λ_β
γ	$K_{\gamma,0}$	$K_{\gamma,1}$	λ_γ

Garbled Gate:

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

Note: This is standard Yao using point-and-permute

Recall: (Single-Party) Circuit Garbling Scheme



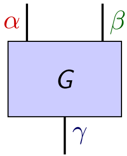
Wire	Keys		Mask Bit
α	$K_{\alpha,0}$	$K_{\alpha,1}$	λ_α
β	$K_{\beta,0}$	$K_{\beta,1}$	λ_β
γ	$K_{\gamma,0}$	$K_{\gamma,1}$	λ_γ

Garbled Gate:

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

↑ Keys

Recall: (Single-Party) Circuit Garbling Scheme



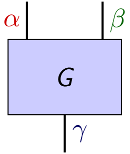
Wire	Keys		Mask Bit
α	$K_{\alpha,0}$	$K_{\alpha,1}$	λ_α
β	$K_{\beta,0}$	$K_{\beta,1}$	λ_β
γ	$K_{\gamma,0}$	$K_{\gamma,1}$	λ_γ

Garbled Gate:

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

↑ Tags

Recall: (Single-Party) Circuit Garbling Scheme



Wire	Keys		Mask Bit
α	$K_{\alpha,0}$	$K_{\alpha,1}$	λ_α
β	$K_{\beta,0}$	$K_{\beta,1}$	λ_β
γ	$K_{\gamma,0}$	$K_{\gamma,1}$	λ_γ

Garbled Gate:

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

Note: Garbling party knows keys/tags being encrypted

Distributing the Garbling Scheme

Goal: P_1 and P_2 together compute garbled circuit such that no party "knows" garbled values

Distributing the Garbling Scheme

Goal: P_1 and P_2 together compute garbled circuit such that no party "knows" garbled values

Desired properties:

Distributing the Garbling Scheme

Goal: P_1 and P_2 together compute garbled circuit such that no party "knows" garbled values

Desired properties:

1. Obliviousness
   - Neither party should know key/tag being encrypted

Distributing the Garbling Scheme

Goal: P_1 and P_2 together compute garbled circuit such that no party "knows" garbled values

Desired properties:

1. Obliviousness
   - Neither party should know 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

Goal: P_1 and P_2 together compute garbled circuit such that no party "knows" garbled values

Desired properties:

1. Obliviousness
   - Neither party should know 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 encryption [DI05] with authenticated bit shares [NNOB12]

Building Blocks (1): Distributed Encryption Scheme [DI05]

Goal: P_1 and P_2 want to encrypt secret shared message $[m] = m_1 \oplus m_2$ using keys K_1, K_2

Building Blocks (1): Distributed Encryption Scheme [DI05]

Goal: P_1 and P_2 want to encrypt secret shared message $[m] = m_1 \oplus m_2$ using keys K_1, K_2

Keys are split into *sub-keys*: $K_1 = (s_1^1, s_1^2)$, $K_2 = (s_2^1, s_2^2)$

Building Blocks (1): Distributed Encryption Scheme [DI05]

Goal: P_1 and P_2 want to encrypt secret shared message $[m] = m_1 \oplus m_2$ using keys K_1, K_2

Keys are split into *sub-keys*: $K_1 = (s_1^1, s_1^2)$, $K_2 = (s_2^1, s_2^2)$

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

Building Blocks (1): Distributed Encryption Scheme [DI05]

Goal: P_1 and P_2 want to encrypt secret shared message $[m] = m_1 \oplus m_2$ using keys K_1, K_2

Keys are split into *sub-keys*: $K_1 = (s_1^1, s_1^2)$, $K_2 = (s_2^1, s_2^2)$

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

Note: Encryption is **local**!

Building Blocks (1): Distributed Encryption Scheme [DI05]

Goal: P_1 and P_2 want to encrypt secret shared message $[m] = m_1 \oplus m_2$ using keys K_1, K_2

Keys are split into *sub-keys*: $K_1 = (s_1^1, s_1^2)$, $K_2 = (s_2^1, s_2^2)$

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

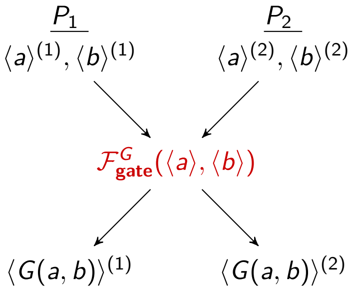
Note: Encryption is **local**!

Note: Cost per party (in PRF calls) to encrypt message of length ℓ is 2ℓ

Building Blocks (2): Functionalities Needed

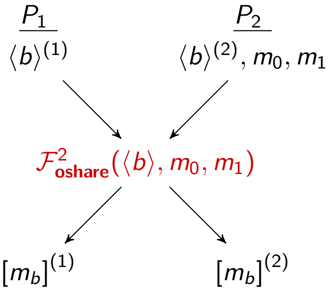
Note: $\langle\cdot\rangle$ denotes (form of authenticated and linear) bit secret sharing
$[\cdot]$ denotes (standard) secret sharing
$\langle\cdot\rangle^{(i)}$, $[\cdot]^{(i)}$ denotes P_is share

Building Blocks (2): Functionalities Needed



Note:
$\langle \cdot \rangle$ denotes (form of authenticated and linear) bit secret sharing
$[\cdot]$ denotes (standard) secret sharing
$\langle \cdot \rangle^{(i)}$, $[\cdot]^{(i)}$ denotes P_is share

Building Blocks (2): Functionalities Needed



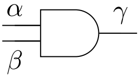
Note:
$\langle \cdot \rangle$ denotes (form of authenticated and linear) bit secret sharing
$[\cdot]$ denotes (standard) secret sharing
$\langle \cdot \rangle^{(i)}$, $[\cdot]^{(i)}$ denotes P_is share

Building Blocks (2): Functionalities Needed

Note: efficient maliciously secure constructions exist

- Uses ideas from [NNOB12]
- See paper for details

Example: Garbling an AND Gate



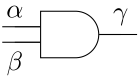
$$\lambda_\alpha = 1,\ \lambda_\beta = 0,\ \lambda_\gamma = 1$$

Standard (single-party) garbling:

Step 1: S computes 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



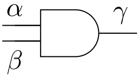
$$\langle \lambda_\alpha \rangle = 1,\ \langle \lambda_\beta \rangle = 0,\ \langle \lambda_\gamma \rangle = 1$$

Distributed garbling:

Step 1: P_1 and P_2 compute *oblivious sharings* of tags:

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

Example: Garbling an AND Gate



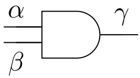
$$\lambda_\alpha = 1,\ \lambda_\beta = 0,\ \lambda_\gamma = 1$$

Standard (single-party) garbling:

Step 2: S encrypts key + tag:

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



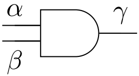
$\langle \lambda_\alpha \rangle = 1$, $\langle \lambda_\beta \rangle = 0$, $\langle \lambda_\gamma \rangle = 1$

Distributed garbling:

Step 2a: P_1 and P_2 compute *oblivious sharings* of each party's sub-keys:

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

Example: Garbling an AND Gate



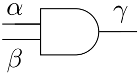
$$\langle\lambda_\alpha\rangle = 1,\ \langle\lambda_\beta\rangle = 0,\ \langle\lambda_\gamma\rangle = 1$$

Distributed garbling:

Step 2b: P_1 and P_2 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	$\mathsf{Enc}_{K_{\alpha,1},K_{\beta,1}}\left(\left[s^1_{\gamma,1}\right] \parallel \left[s^2_{\gamma,1}\right] \parallel \langle 1\rangle\right)$

Example: Garbling an AND Gate



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

Distributed garbling:

Step 2b: P_1 and P_2 use *distributed encryption* to encrypt:

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

Note: Cost of encryption $8\times$ cost in 2PC setting

3PC: Main Steps

Two main steps:

1. Distribute S's circuit garbling scheme between two parties
2. Modify existing 2PC protocol to use distributed circuit garbling scheme

Lifting 2PC Protocols to Three Party Setting

Goal: Construct 3PC scheme using distributed garbling protocol

Lifting 2PC Protocols to Three Party Setting

Goal: Construct 3PC scheme using distributed garbling protocol

High-Level Idea

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

Lifting 2PC Protocols to Three Party Setting

Goal: Construct 3PC scheme using distributed garbling protocol

High-Level Idea

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

Note: Not exactly this straightforward; see paper for details

Lifting 2PC Protocols to Three Party Setting

Goal: Construct 3PC scheme using distributed garbling protocol

High-Level Idea

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

Note: Not exactly this straightforward; see paper for details

Security Intuition

- Exactly one of P_1 *or* P_2 malicious: garbled circuits either correct or abort independent of input
- Both P_1 *and* P_2 malicious: cut-and-choose by P_3 detects cheating
- P_3 malicious: covered by security of garbling protocol

Summary

Can "lift" cut-and-choose 2PC protocols to 3PC setting

- Provides efficient constant round 3PC protocol
- Only $\approx 8\times$ slower than underlying 2PC protocol
- Approach works for combination of [LP07, LP11] and [Lin13]
- Only *three* broadcast calls needed
  - Important in WAN settings where broadcast is expensive
- Faster *start-to-finish* time than existing 3PC solutions

Summary

Can "lift" cut-and-choose 2PC protocols to 3PC setting

- Provides efficient constant round 3PC protocol
- Only $\approx 8\times$ slower than underlying 2PC protocol
- Approach works for combination of [LP07, LP11] and [Lin13]
- Only *three* broadcast calls needed
  - Important in WAN settings where broadcast is expensive
- Faster *start-to-finish* time than existing 3PC solutions

Future Work:

- Support free-XOR
- Optimize distributed encryption scheme (à la JustGarble [BHKR13])

Thank You

Any questions?

E-mail: amaloz@cs.umd.edu
URL: https://www.cs.umd.edu/~amaloz
ePrint: https://eprint.iacr.org/2014/128