Implementing Component-Based Garbled Circuits

Alex Ledger

Reed College

May 2, 2016

Overview

1. Cryptographic Primitives
2. Security and Classical Garbled Circuits
3. Improvements to Garbled Circuits
4. Component-Based Garbled Circuits + SCMC
5. Implementation and Results

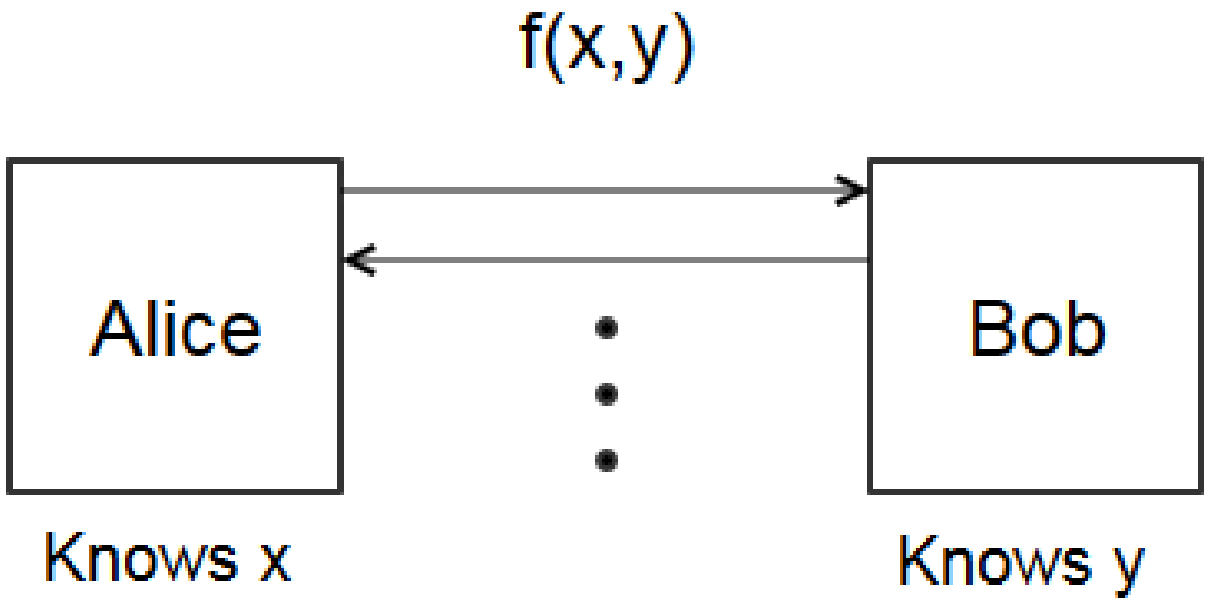
Introduction

TODO

The Millionaire Problem

- Alice and Bob wish to determine who is wealthier.
- They do not want to disclose their wealth to each other.
  - Alice has $\$x$, Bob has $\$y$.
  - Alice should not learn anything about y.
  - Bob should not learn anything about x.

$$f(x,y) = \begin{cases} Alice, & y \leq x; \\ Bob, & y > x. \end{cases} \tag{1}$$



Security Parameter

- asdf

Encryption

$$\mathsf{Gen}(1^n) \to k$$
$$\mathsf{Enc}_k(pt) \to ct$$
$$\mathsf{Enc}_k^{-1}(ct) \to pt$$

Security of Encryption

An encryption scheme is secure under a chosen-plaintext attack if for all probabilistic polynomial-time adversaries A, there exists a negligible function μ such that

$$Pr[E_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mu(n)$$

where E is the following experiment:

1. Generate key k by running $\text{Gen}(1^n)$.
2. The adversary $\mathcal{A}$ is given 1^n and oracle access to Enc_k, and outputs a pair of messages m_0 and m_1 of the same length.
3. A uniform bit $b \leftarrow \{0, 1\}$ is sampled uniformly at random, and then ciphertext $c \leftarrow \text{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$.
4. Then $\mathcal{A}$ continues to have oracle access to Enc_k, and outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise. If at any point $\mathcal{A}$ encrypts m_0 or m_1 with their encryption oracle, the output is 0. 1 indicates that the adversary wins, and 0 indicates that the adversary loses.

Computational Indistinguishability

Intuition:

- Let $\mathcal{X}$ and $\mathcal{Y}$ be distribution ensembles.
- Give an adversary one of the distribution ensembles.
- Can the adversary sample from the distribution a polynomial number of times to determine which distribution they were given?

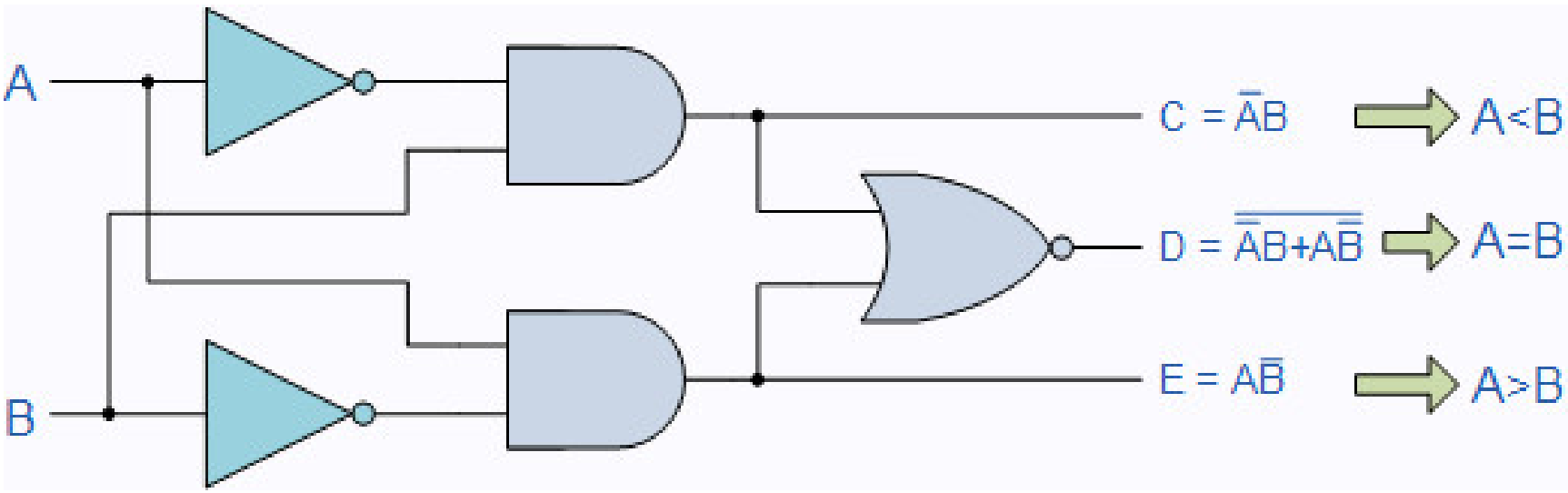
Formal Definition:

- Let $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n\in\mathbb{N}}$ be distribution ensembles.
- Then $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable, denoted $\mathcal{X} \approx_C \mathcal{Y}$, if for all probabilistic polynomial-time algorithms D, there exists a negligible function μ such that:

$$|Pr_{x\leftarrow X_n}[D(1^n, x) = 1] - Pr_{y\leftarrow Y_n}[D(1^n, y) = 1]| < \mu(n)$$

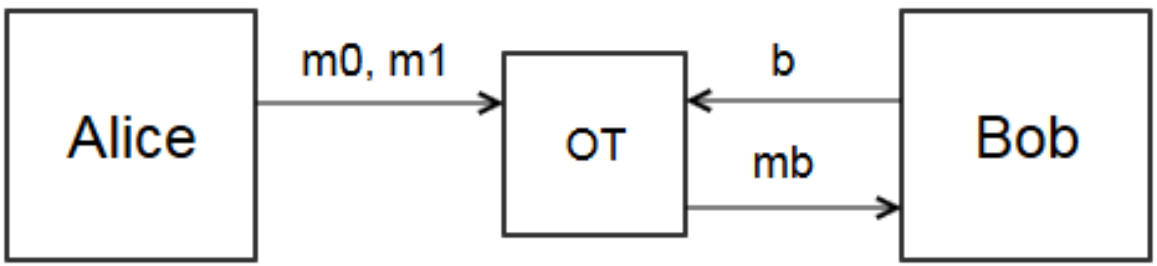
Boolean Circuits

- We encode a function f into a circuit C.
- Circuit C is made of AND, XOR and NOT gates.
- Each gate has two input wires and a single output wire

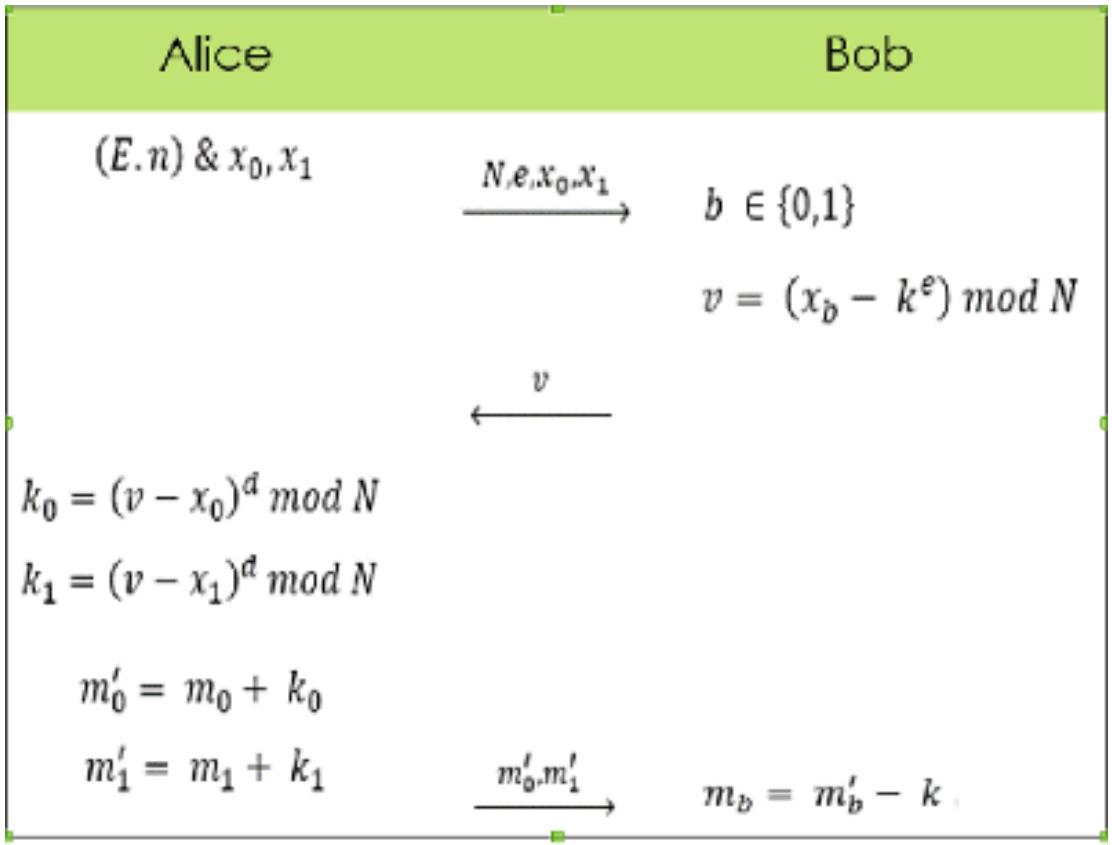


Oblivious Transfer (OT)

- Alice potentially sends either m_0 or m_1 to Bob.
- Bob, without seeing m_0 or m_1, decides that he wants m_b.
- Bob receives m_b.
- Property 1: Alice does not know which message Bob recieved.
- Property 2: Bob doesn't anything about m_{1-b}.



Oblivious Transfer 2



RSA-based oblivious transfer

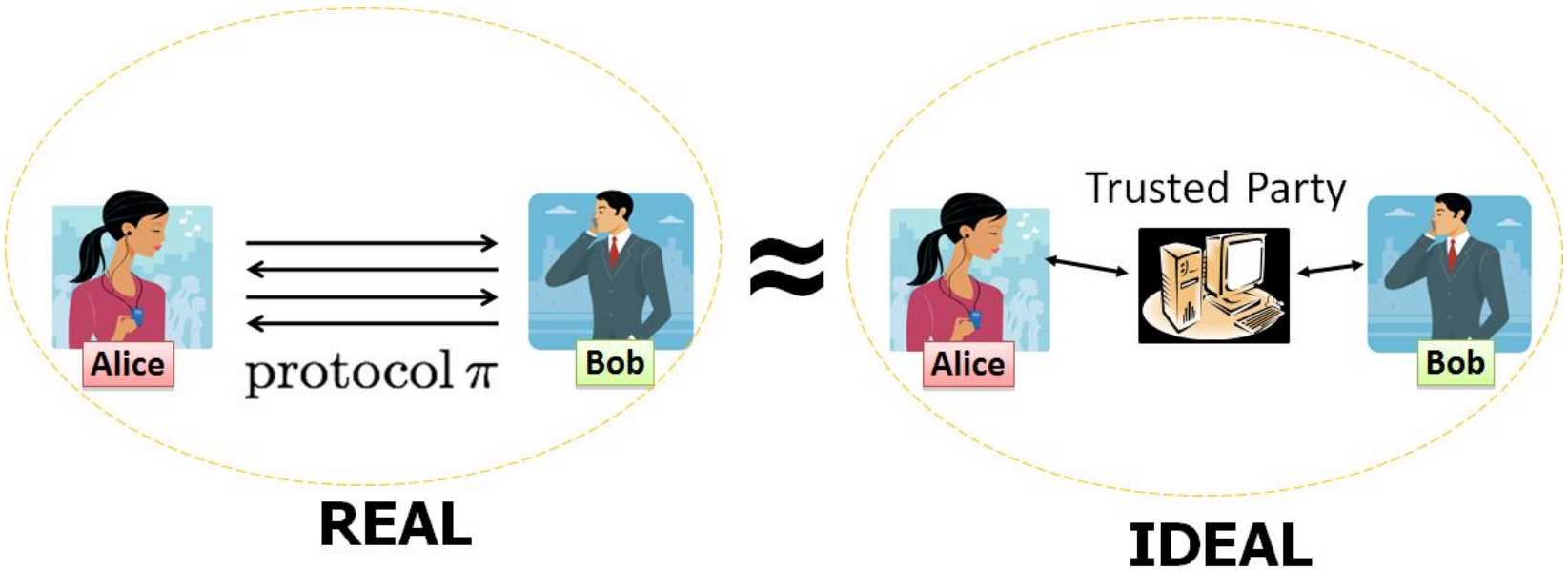
Setting up Security of 2PC

- Privacy of inputs
  - Alice and Bob do not learn anything about the each other's input.
  - Except for info that is inferable from x and $f(x, y)$.
  - Bob should not learn that $1,000,000 < x \leq 2,000,000$.
  - But if $y < x$ and $y = 2,000$, then he learns $x < 2,000$.
- Correctness
  - Alice and Bob receive $f(x, y)$.
  - As opposed to some value near $f(x, y)$.
  - Or not receiving a value at all

Semi-Honest

- We assume that each party obeys the protocol, but attemps to learn extra information from its interactions
- In contrast, Alice and Bob could lie, deceive, not send some messages, etc.
  - This is called the malicious setting
- Reasons we use the semi-honest setting
  - It's easier
  - Some realistic scenarios (e.g, hosptitals)
  - JustGarble is written for semi-honest setting

Security of 2PC (Overview)



A secure computation protocol is secure if Alice and Bob learn the same information in the real world as they would in ideal world.

Security of 2PC (Ideal World)

- We use computational indistinguishability
- Model information that Alice and Bob can infer in real and ideal world
- Simulators: adversaries in the real world
  - We have S_A and S_B
  - Take as input x and $f(x, y)$.
- So our *distribution ensembles* of information in the ideal world are

$$\{S_A(x, f(x, y)\}_{x \in \{0,1\}^*} \text{ and } \{S_B(y, f(x, y)\}_{y \in \{0,1\}^*}$$

Security of 2PC (Real World)

- In the real world, we introduce $\text{view}_A(x, y)$ and $\text{view}_B(x, y)$.
- Think of $\text{view}_A(x, y)$ as the set of all messages that Alice sends and receives during the execution of the protocol.
- We construction distribution ensembles as

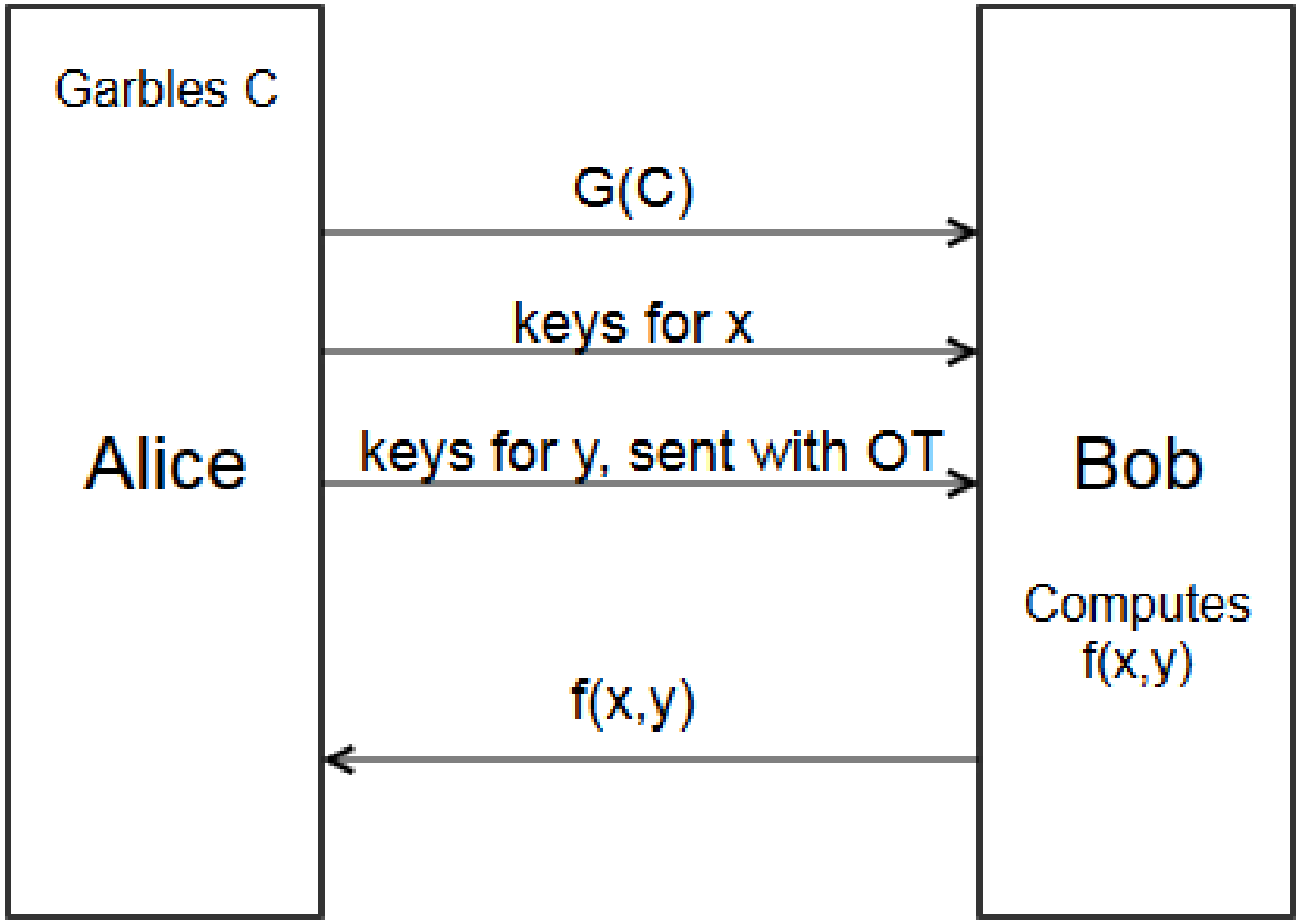
$$\{\text{view}_A(x, y)\}_{x,y\in\{0,1\}^*}$$
$$\{\text{view}_B(x, y)\}_{x,y\in\{0,1\}^*}$$

Security of 2PC (Real World)

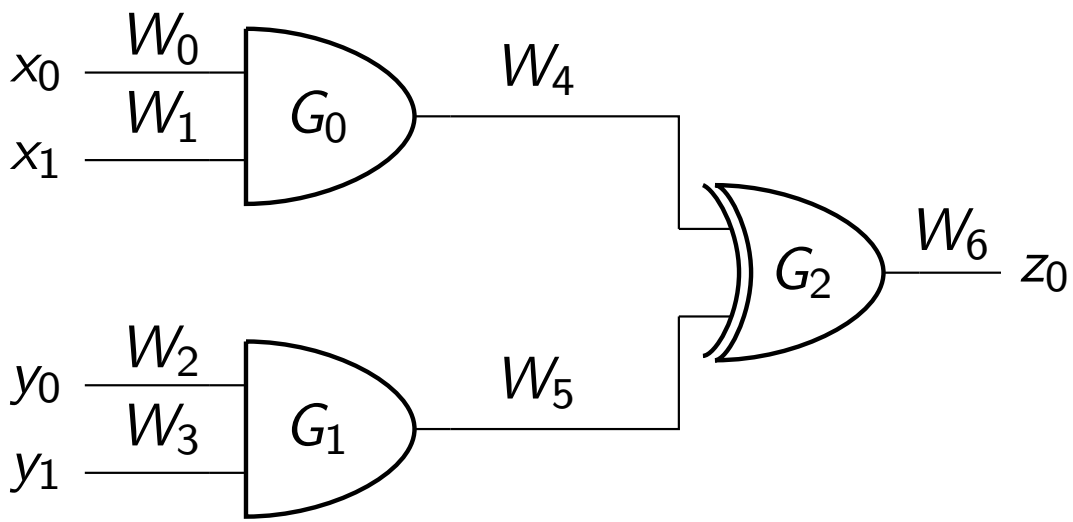
- Define $output_i^{\Pi}(n, x, y)$ as the output of the ith party on input (x, y) and security parameter n.
- Then Π is secure if

$$\{S_A(x, f_A(x, y), f(x, y))\}_{x,y} \approx_C \{(\text{view}_A^{\Pi}(x, y), \text{output}^{\Pi}(x, y))\}_{x,y}$$
$$\{S_B(x, f_B(x, y), f(x, y))\}_{x,y} \approx_C \{(\text{view}_B^{\Pi}(x, y), \text{output}^{\Pi}(x, y))\}_{x,y}$$

Garbled Circuits



Garbled Circuits



Garbled Circuits

1. Alice assigns wire labels
2. Alice make garbled tables
   - Alice permutes rows
3. Alice sends garbled tables to Bob
4. Alice sends input wire labels to Bob
   - Some are sent via OT
   - Others are simply sent
5. Bob trial decrypts each row of garbled table with input wire labels
6. Bob acquires output wire label
7. Bob uses output wire label in subsequent gates
8. Bob eventually acquires output wire labels

Security of Garbled Circuits

- Alice only receives messages from Bob during OT
  - We assume OT is secure, so Alice does not learn any information
- Bob recieves the following:
  - Input wire labels
  - OT messages
  - Garbled tables
- Bob can only decrypt a gate if he has two wire labels
- Bob can only acquire two wire labels per gate with this information
- Hence Bob cannot learn any extra information

The Cost of Garbled circuits

- Alice sends 4 ciphertexts per gate, since the garbled table has 4 rows, to Bob.
  - This is the bandwidth, or the amount of the communication required.
  - Based on empirical work, bandwidth is the biggest bottleneck in garbled circuits.
  - So reducing the size of the garbled table is of utmost priority.
- Alice performs 4 encryptions.
- Bob performs on average 2.5 decryptions
  - Since garbled table has 4 rows

Improvements to Garbled Circuits

Garbled Circuit Improvement	Table Size (xλ)		Garble Cost		Eval Cost	
	XOR	AND	XOR	AND	XOR	AND
Classical	4	4	4	4	4	4
Point and Permute	4	4	4	4	1	1
GRR3	3	3	4	4	1	1
Free XOR	0	3	0	4	0	1
GRR2	2	2	4	4	1	1
FleXOR	{0,1,2}	2	{0,2,4}	4	{0,1,2}	1
Half Gates	2	0	2	0	0	2

Table: Table size is number of ciphertexts. Garble cost is number of encryptions the garbler performs. Eval cost is number of decryptions the evaluator performs.

Point and Permute

All examples use an AND gate with input wires W_i and W_j and output wire W_k.

Select Bit	Wire Label
0	W_i^0
1	W_i^1
1	W_j^0
0	W_j^1

Select Bits	Encryption
(0,0)	$\mathrm{Enc}_{W_i^0, W_j^1}(W_k^0)$
(0,1)	$\mathrm{Enc}_{W_i^0, W_j^0}(W_k^0)$
(1,0)	$\mathrm{Enc}_{W_i^1, W_j^1}(W_k^1)$
(1,1)	$\mathrm{Enc}_{W_i^1, W_j^0}(W_k^0)$

Table: Garbled AND gate for Point and Permute

Garbled Row Reduction 3

Select Bit	Wire Label
0	W_i^0
1	W_i^1
1	W_j^0
0	W_j^1

Select Bits	Encryption
(0,1)	$\mathsf{Enc}_{W_i^0, W_j^0}(W_k^0)$
(1,0)	$\mathsf{Enc}_{W_i^1, W_j^1}(W_k^1)$
(1,1)	$\mathsf{Enc}_{W_i^1, W_j^0}(W_k^0)$

$$W_k^0 \leftarrow \mathsf{Enc}^{-1}_{W_i^0, W_j^1}(0^n)$$
$$W_k^1 \leftarrow \{0,1\}^n$$

Free XOR

- Let $\Delta \leftarrow \{0,1\}^n$ be fixed globally in a circuit.
- For each input wire A, sample a single ciphertext; call it A.
  - The *zeroith* wire label is A.
  - The *first* wire label is $A \oplus \Delta$.
- Set output wire C of a gate to be $A \oplus B$.
  - The xor of its input wires.
- Bob *evaluates* the XOR gate by XORing the two input labels.

$$(A \oplus a\Delta) \oplus (B \oplus b\Delta) = A \oplus B \oplus (a \oplus b)\Delta$$

- So XOR gates do not require a garbled table, aka they're Free.

Others

- FleXOR
- Half-gates

OT-preprocessing

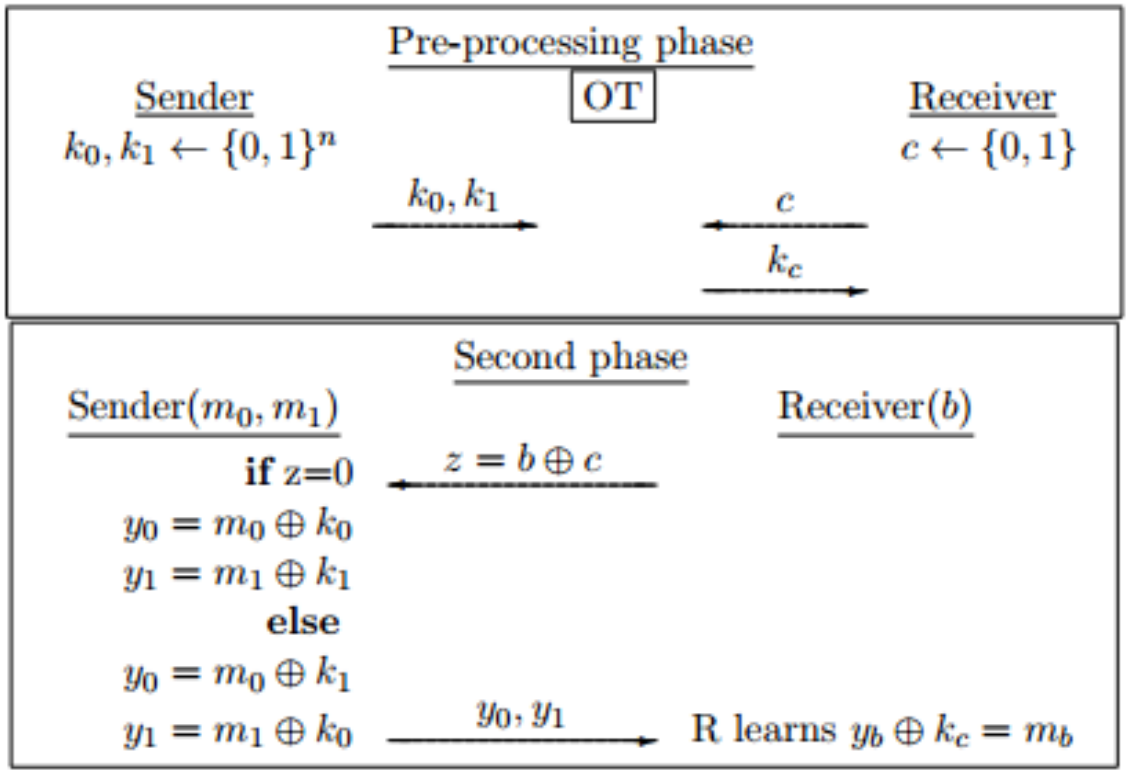


Figure 1: Protocol for Pre-processing OT

Source: Katz lecture notes

Motivating Component-Based GCs

- Split protocols into offline/online
  - Garble and send circuit in offline phase.
  - Determine inputs during online phase, evaluate.
- Problem: function chosen ahead of time - no flexiblity

The basics of Component-Based GCs

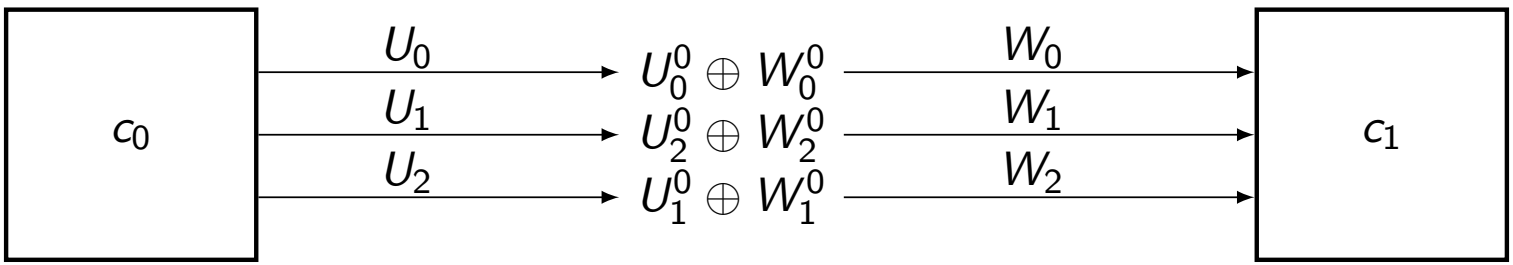
- Observe that many useful functions are composed of standard components
- Component-based garbled circuits garble and send a *library* of components during offline phase
- Link or stitch together components during online phase to build the function

Example uses of Component-Based GCs

- Bank ATM
- Online poker
- Election or auction

Details of Component-Based Garbled Circuits (1)

Details of Component-Based GCs (2)



Efficiency of chaining garbled circuits

- In online phase, chaining requires communicating a ciphertext per chain
- This can be a lot of communication
  - Suppose 100 by 100 matrix with entries of max value 2^8
  - Then requires $8,000$ ciphertexts
  - That's 256 kBs.
- I came up with a method to reduce this to a single ciphertext per data object

Security of Component-Based GCs

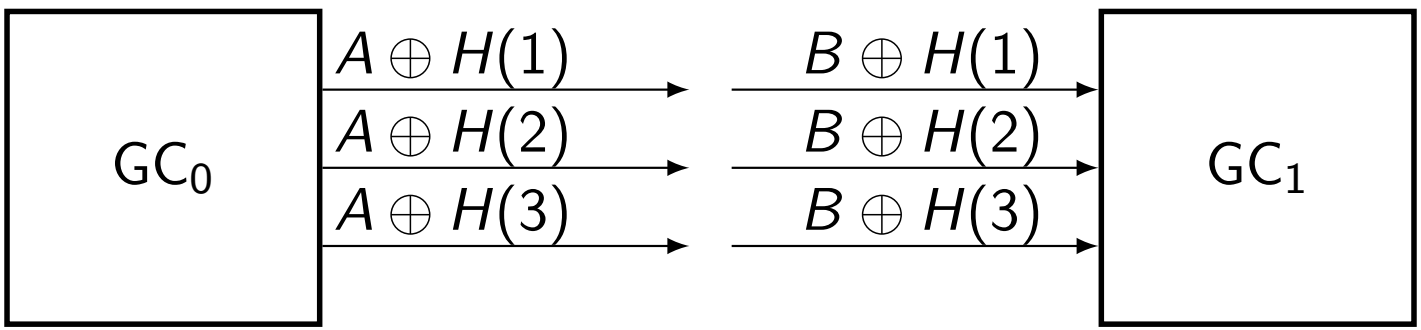
- We think through security using a *hybrid* argument.
  - Rely on fact that computational indistinguishability is transitive
  - By stripping away information from the real world, we slowly transform it into the ideal world.
- We remove:
  - Link labels
  - Input wire labels
  - Garbled tables
  - Output map

Single Communication Multiple Connections SCMC

- Problem: number of ciphertexts communicated scales linearly with the number of chains
- Key observation:
  - the chaining is predictable, consecutive wires are chained to consecutive wires
  - Pieces of data move around in a function, like matrices or numbers or text
- Solution: give consecutive wire labels a predictable pattern

Single Communcation Multiple Connections SCMC

- For each piece of data, sample a label A.
- Set ith wire label of data to $A \oplus H(i)$.
- Where H is a hash function



- Technically, we set the links to $A \oplus H(T \oplus (i||b))$

Security of SCMC

- SCMC security follows from naive component-based garbled circuits security
- Link labels are find to send
- And wire labels are *pseudorandom* since H is a random oracle

Implementation

- I implemented component-based GCs and SCMC in a program called CompGC
- Written in C
- It does the entire garbled circuits protocol
- Takes circuits and inputs and securely computed the output

Some code

The experiments

- AES Encryption
  - AES-round was only component
- CBC Mode of Operation
  - A method for encrypting arbitrary size messages
  - XOR and AES-round were components
- Levensthein Distance Algorithm
  - Used 8-bit alphabet
  - Ran 30 symbols and 60 symbols

Results

	Time (localhost)		Time (simulated network)		Communication	
	Naive	CompGC	Naive	CompGC	Naive	CompGC
AES	4.4 ± 0.0 ms	3.0 ± 0.2 ms	542.6 ± 0.7 ms	68.5 ± 0.2 ms	24 Mb	254 Kb
CBC, 10 blocks	45.8 ± 4.0 ms	22.7 ± 1.4 ms	4.8 ± 0.0 s	216.7 ± 0.2 ms	235 Mb	2.6 Mb
Leven, 30 symbols	28.9 ± 6.6 ms	24.3 ± 1.2 ms	2.2 ± 0.0 s	315.9 ± 0.5 ms	108 Mb	6.3 Mb
Leven, 60 symbols	109.8 ± 7.0 ms	62.2 ± 0.7 ms	10.6 ± 0.0 s	742.5 ± 2.0 ms	524 Mb	25 Mb

Table: Experimental results. Time is online computation time, not including the time to preprocess OTs, but including the time to load data from disk. All timings are of the evaluator's running time, and are the average of 100 runs, with the value after the ± denoting the 95% confidence interval. The communication reported is the number of bits received by the evaluator.

Results Without Loading Time

	Time (localhost)		Time (simulated network)	
	Naive	CompGC	Naive	CompGC
AES	4.4 ± 0.0 ms	1.3 ± 0.1 ms	542.6 ± 0.7 ms	66.9 ± 0.1 ms
CBC mode, 10 blocks	45.8 ± 4.0 ms	8.8 ± 0.5 ms	4.8 ± 0.0 s	204.3 ± 0.2 ms
Levenshtein, 30 symbols	28.9 ± 6.6 ms	14.1 ± 0.4 ms	2.2 ± 0.0 s	305.6 ± 0.2 ms
Levenshtein, 60 symbols	109.8 ± 7.0 ms	27.1 ± 0.4 ms	10.6 ± 0.0 s	703.4 ± 1.5 ms

Table: Experimental results without counting the evaluator time to load data from disk.

Comparing Naive to SCMC

	Time (simulated network)		**Communication**	
	Standard	SCMC	Standard	SCMC
AES	134.4 ± 0.1 ms	68.5 ± 0.2 ms	656 Kb	254 Kb
CBC mode, 10 blocks	321.5 ± 0.9 ms	216.7 ± 0.2 ms	7.4 Mb	2.6 Mb
Levenshtein, 30 symbols	371.0 ± 0.9 ms	315.9 ± 0.5 ms	10.0 Mb	6.3 Mb
Levenshtein, 60 symbols	1119.6 ± 2.1 ms	742.5 ± 2.0 ms	44 Mb	25 Mb

Table: Comparison of the two approaches for component-based garbled circuits: the standard approach and the SCMC approach. The experiments are run on the simulated network.

Future work

- Malicious setting
- MPC - more than two parties
  - Then we can do auctions and elections with arbitrarily many parties
- Make linking faster
- Reduce memory footprint
- Develop a library of components