Two Party Computation

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Alex Ledger

May 2015

Approved for the Division
(Mathematics)

Adam Groce

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class.

List of Abbreviations

You can always change the way your abbreviations are formatted. Play around with it yourself, use tables, or come to CUS if you'd like to change the way it looks. You can also completely remove this chapter if you have no need for a list of abbreviations. Here is an example of what this could look like:

ABC	American Broadcasting Company
CBS	Columbia Broadcasting System
CDC	Center for Disease Control
CIA	Central Intelligence Agency
CLBR	Center for Life Beyond Reed
CUS	Computer User Services
FBI	Federal Bureau of Investigation
NBC	National Broadcasting Corporation

Table of Contents

List of Tables

List of Figures

Abstract

The preface pretty much says it all.

Dedication

You can have a dedication here if you wish.

Introduction

Introduction goes here

Chapter 1

Background

Multiparty computation (MPC) is the study and creation of protocols for computing a function between multiple parties, such that no party learns the input of any other party.

The idea is best communicated through an example: suppose Alice and Bob are millionaires and wish to determine who is wealthier, but Alice and Bob are also secretive, and do not want to disclose their exact amount of wealth. Is there some method by which they can determine who has more money?

The goal of MPC is to design a protocol which will help Alice and Bob solve their problem. The desired properties of a secure MPC scheme can be informally described as follows:

- **Privacy:** Each party's input is kept secret.
- **Correctness:** The correct answer to the computation is computed.

Originally, the goal was to come up with a protocol that was secure and prove that the protocol was secure. In more recent times, the focus has shifted to making the MPC faster, fast enough that it can be used regularly in the real world.

If MPC can be made fast enough, it could serve a wide range of applications. For example, imagine that two companies who operate in a similar industry want to work together, but they don't want to disclose any company research which the other doesn't know. These companies could a run set intersection function (a function that given two inputs finds their intersection, or overlap), to determine what information they can disclose without giving away important information.

Another interesting example of MPC is to improve the outsourcing of computation. As it is right now, cloud computing companies, such as Amazon and Google, have really nice computers which they will rent out to you. You can pay them someone money, write a program, and run it on their computers. The problem is that you may not trust the cloud computing company, and you want some guarantees that they are going to respect the privacy of your computation. An MPC protocol, in this setting, would allow you to run computation in the cloud, with the guarantee that the inputs to your computation are disguised.

Since the research into MPC has focused on creating a method by which an arbitrary function can be computed securely, the application of MPC beyond what we can presently conceive of. It's not unlikely that MPC protocols will become a standard in the internet, where when you access the internet, behind the scenes your access is being is plugged into an MPC protocol, sent off to another computer to do some processing. As cryptography improves, research in MPC and other areas of cryptography, the hope is that the security of our computer systems will improve as well. However, there is no guarantee. The modern cryptography needs to implemented and used, perhaps in some cases built into low-level standards, and used correctly. At this point, the outlook of cryptography is bright, but the future will only be realized positvely if it is actively worked towards.

1.1 Tools for MPC

Imagine again that millionaires Alice and Bob wish to determine who has more wealth. Suppose that Alice has x_a dollars and Bob has y_b dollars. Then the function that they wish to compute is $f : \{0,1\}^n \rightarrow \{0,1\}$ as defined by:

$$f(x_a, x_b) = \begin{cases} 0, & x_a \leq x_b; \\ \\ 1, & \text{otherwise.} \end{cases} \tag{1.1}$$

Why is this here? If Alice and Bob successfully compute $f(x, y)$ and each gets the output, then each party has gained some knowledge about the nature of the other's input. For example, if $f(x_a, x_b)$ outputs 0, then Alice and Bob know that Alice has more money. Alice has learned that Bob has less than x dollars, and Bob has learned that Alice has more x_b dollars. Therefore it's not possible for MPC to guarantee that *no* information about the other parties' inputs is learned, only that nothing more is learned than what can be inferred from a single party's input and the output of f.

MPC requires a combination of several cryptographic tools. The next few sections will describe these basic tools, and then MPC will described in section x.

1.1.1 Encryption

Encyprtion is the process of encoding a message such that only parties with the key can read the message. An encryption protocol is composed of two parts. The first part is the encryption function which disguises the message, and the second part is the decryption function which unobfuscates the disguised message. We notate encryption and decryption with where pt stands for plaintext and is the original message, ct stands for ciphertext and is the encrypted message, k is the secret key, that only authorized readers of the message hold, and is a random sequence of λ 0s and 1s, and λ is a security parameter of our protocol (As λ increases, the size of the key increases, and so the encryption becomes harder to break.)

The encryption decribed above is more precisely called symmetric-key encryption, as opposed to public-key encryption. The difference between the two is that for symmetric-key encryption, there is a single key and all communicating must have the same key in order to acheive secure communication, whereas in public-key encryption, the encryption key is published publicly, and anyone can send a message using the public key, and the message can only be decrypted by those with the secret key. For more information on encryption, we encourage the reader to peruse the many great online resources.

The MPC protocols that will be examined here exclusively use symmetric-key encryption.

1.1.2 Boolean Circuit

A function for an MPC protocol is represented by a boolean circuit. A boolean circuit takes as input a sequence of n 0s and 1s, (i.e. a value in $\{0,1\}^n$), performs a series of small opreations on the inputs, and outputs a sequence of m 0s and 1s (i.e. a value in $\{0,1,\}$). You may have encountered circuits and logical operators in another context, where the inputs and outputs were True and False. For our usage, True will correspond to the value 1, and False will corresond to the value 0.

The small operations performed inside of a circuit are performem by an object called a *gate*. A gate is composed of three wires: two input wires and one output wire, where a *wire* can have a value either 0 or 1. A gate performs a simpler operation on the two inputs, resulting in a single output bit. Table 1.1.2 gives the mapping of an XOR gate.

A circuit is a combination of gates. In fact, a circuit built out of only AND gates, XOR gates and NOT gates can compute any function. **Find details and citation**

x	y	xor(x,y)
1	1	0
1	0	1
0	1	1
0	0	0

Table 1.1: The mapping of an XOR gate.

In other words, if there's some algorithm that do it, then there is some circuit that can do it as well. Hence, a circuit is a sufficient representation of the function f. Figure 1.1.2 shows the circuit representation of a circuit that computes the less than function, the function f, specified in equation 1.1 that millionaires Alice and Bob wanted to compute.

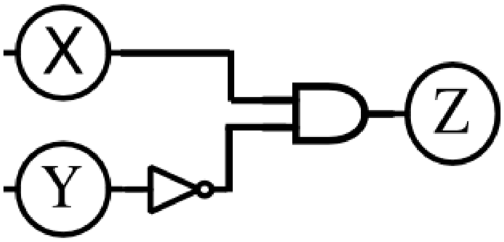


Figure 1.1: A circuit that computes the less or equal to function, equivalent to f for input of two one-bit values. **TODO, add truth table?**

A circuit **with what constraints???** can compute any function.

1.1.3 Technical Definition of a Circuit

Use / begin definition A circuit is formalized by a 6-tuple $g = (n, m, q, A, B, G)$, where n is the number of inputs, m is the number of outputs and q is the number of gates. We define $r = n+q$ to the number of wires inside of the circuit. We let $\text{Wires} = \{1, \ldots, n+q\}, \text{InputWires} = \{1, \ldots, n\}, \text{OutputWires} = \{n+1-m+1, \ldots, n+q\}$, and $\text{Gates} = \{n+1, \ldots, n+q\}$. Then $A : \text{Gates} \rightarrow \text{Wires} \setminus \text{OutputWires}$ identifies each gate's first incoming wire. And $B : \text{Gates} \rightarrow \text{Wires} \setminus \text{OutputWires}$ identifies each gate's second incoming wire. Finally, $G : \text{Gates} \times \{0,1\}^2 \rightarrow \{0,1\}$ identifies the functionality of each gate.

For example, the less than circuit shown in figure 1.1.2 has values:

¡¡¡¡¡¡¡ HEAD

To evaluate a circuit, we use a canonical algorithm ev_{circ}. ev_{circ} takes as input a string f and a string $x = x_1 x_2 \ldots x_n$ and does the following:

Algorithm 1 ev_{circ}

```
Input: Function $f$ and String $x = x_1, \ldots x_n$.
Output: Output of function $f$ on input $x$.
  $(n, m, a, A, B, G) \leftarrow f$                    ▷ parse $f$ as a circuit
  for $g = n + 1$ to $n + q$ do                        ▷ loop over all gates.
      $a \leftarrow A(g)$                              ▷ compute each gate
      $b \leftarrow B(g)$
      $x_g \leftarrow G_g(x_a, x_b)$
  end for
  return $x_{n+q-m+1} \ldots x_{n+q}$
```

Definition 1 Topological Circuit A topological circuit, denoted f^-, is the 5-tuple (n, m, q, A, B) for some circuit $f = (n, m, q, A, B, G)$. Furthermore, define $\mathrm{Topo}(f)$ such that $f^- = \mathrm{Topo}(f)$. ♢

A topological circuit is the same as its conventional circuit f, except that the functionality of the gates is not specified. If one holds a topological circuit, then they know the structure of the function, but not what is being computed. =======

1.1.4 Encryption

Encryption is a method of encoding a message such that only parties with the key can read the message. An encryption protocol is composed of two parts. The first part is the encryption function which disguises the message, and the second part is the decryption function which unobfuscates the disguised message. We notate encryption and decryption with

$$\begin{aligned} \mathrm{Enc}_k(pt) &= ct \\ \mathrm{Dec}_k(ct) &= pt \end{aligned} \tag{1.2}$$

where *pt* stands for plaintext and is the original message, *ct* stands for ciphertext and is the encrypted message, k is the secret key, that only authorized readers of the message hold, and is a random sequence of λ 0s and 1s, and λ is a security parameter of our protocol (As λ increases, the size of the key increases, and so the encryption becomes harder to break.)

The encryption decribed above is more precisely called symmetric-key encryption, as opposed to public-key encryption. The difference between the two is that for symmetric-key encryption, there is a single key and all communicating must have the same key in order to acheive secure communication, whereas in public-key encryption, the encryption key is published publicly, and anyone can send a message using the public key, and the message can only be decrypted by those with the secret key. For more information on encryption, we encourage the reader to peruse the many great online resources.

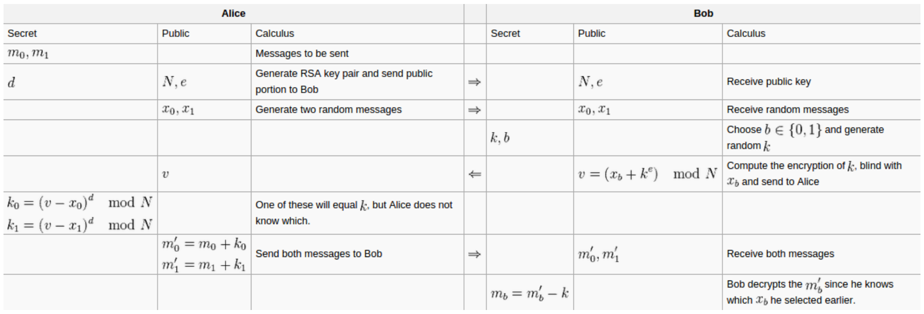
The MPC protocols that will be examined here exclusively use symmetric-key encryption.

1.1.5 Digital Signatures/MACS

Do we need this?

1.1.6 The DDH Assumption

When a cryptographic scehe is said to be *secure*, cryptographers actually mean something much more precise. When a cryptographic scheme is considered secure, it actually means that an adversary can beat the scheme only if the adversary can tackle the hardness assumptions on which the scheme is based. A common hardness assumption in crytoraphy is the Decisional Diffie-Helman Assumption (DDH Assumption), an assumption about solving a problem concerning discrete logs.



Alice				Bob		
Secret	Public	Calculus		Secret	Public	Calculus
m_0, m_1		Messages to be sent				
d	N, e	Generate RSA key pair and send public portion to Bob	$\Rightarrow$		N, e	Receive public key
	x_0, x_1	Generate two random messages	$\Rightarrow$		x_0, x_1	Receive random messages
				k, b		Choose $b \in \{0, 1\}$ and generate random k
	v		$\Leftarrow$		$v = (x_b + k^e) \mod N$	Compute the encryption of k, blind with x_b and send to Alice
$k_0 = (v - x_0)^d \mod N$ $k_1 = (v - x_1)^d \mod N$		One of these will equal k, but Alice does not know which.				
	$m'_0 = m_0 + k_0$ $m'_1 = m_1 + k_1$	Send both messages to Bob	$\Rightarrow$		m'_0, m'_1	Receive both messages
				$m_b = m'_b - k$		Bob decrypts the m'_b since he knows which x_b he selected earlier.

Figure 1.2: Semi-honest Naor-Pinkas oblivious transfer.

Informally, the DDH assumption is:

$$\begin{gathered}\text{Let } G \text{ be a group of order } q \text{ with generator } g. \\ \text{Let } a, b \text{ and } c \text{ be random elements from } \mathbb{Z}_q. \\ \text{Then, } (g^a, g^b, g^{ab}) \approx_D (g^a, g^b, b^c).\end{gathered} \tag{1.3}$$

Define computationally indistinguishability somewhere

1.1.7 Oblivious Transfer

Oblivious Transfer is a special method of communicating a message between two parties where a sender sends one of two messages the receive, and the sender remains oblivious as which message was sent. The setup is the following: Alice has two messages, m_1 and m_2, and he wants to send a message to Bob under the following conditions: First, Alice sends either m_1 or m_2 but not both. Second, Alice does not know which message he sent to Bob. Third, Bob selects which message he wants to receive.

Here we give the Naor-Pinkas protocol of $1 - 2$ oblivious transfer. The protocol relies on the DDH assumption (see section 1.1.6) and is secure in the semi-honest setting (see section **??**).

Researchers have also developed methods for performing k-out-of-n oblivious transfer, where the sender sends exactly k messages out of a possible n. The method described above is called $1-2$ oblivious transfer, and is the OT used in MPC protocols.

Since the first proposal of OT in **year**, several improvements have been developed. Preprocessing. OT extension. Semi-honest/Malicious.

1.1.8 Other

TODO Paper has topological circuit next to real circuit change arbitrary function to arbitrary boolean function.

1.2 Classic MPC

MPC was first proposed by Andrew Yao in an oral presentation on secure function evaluation. After Yao's presentation, two methods for performing MPC were developed. One method was contributed by Yao himself, and the other was contributed by a group of researchers, Beaver, Micali and Widgerson. The two methods are premised on a similar idea: encrypt a circuit by encrypting its gates, with has since been termed garbled circuit. At this point, it is unclear which method is better, both in terms of security and in terms of speed speed. As a result, research is still being done on both protocols.

This section will first give a new definition of a garbling scheme, first presented by BHR in 2012, and then give Yao's garbled circuit protocol, and finally GMW's garbled circuit protocol.

1.2.1 Garbling Schemes

Recall millionaires Alice and Bob who want to determine who has more wealth. Alice and Bob start with a function f (defined in **??**) and their inputs, x_a and x_b, which correspond to their amount of wealth. In order to compute who has more money, the function f needs to be transformed such that they can compute the function by sending messages between each other their inputs, x_a and x_b, which correspond to their amount of wealth. Informally, a garbling scheme transforms a specification of a function into a collection of algorithms which can in combination can compute the function.

Definition 2 Garbling Scheme A garblering scheme is a 5 tuple of algorithms $G = (\mathrm{Gb}, \mathrm{En}, \mathrm{De}, \mathrm{Ev}, \mathrm{ev})$ where the algorithms are

- $\mathrm{Gb}(f, k) \to (F, e, d)$.
- $\mathrm{En}(e, x) \to X$.
- $\mathrm{Ev}(F, X) \to Y$.
- $\mathrm{De}(d, Y) \to y$.
- $\mathrm{ev}(f, x) \to y$. where
  - f is a function.
  - $k \in \mathbb{N}$ is the security parameter.
  - $x \in \{0, 1\}^n$ is the initial input.
  - X is the garbled input
  - Y is the garbled output
  - y is the final output
  - F is a string that describes the operation of the Ev algorithm.
  - e is a string describing the operation of the En algorithm. In particular, e describes how to obscure the input x.
  - d is a string describing the operation of the De algorithm. In particular, d describes how to unobscure the output Y.
- A garbling scheme guarantees correctness if $\mathrm{De}(d, \mathrm{Ev}(F, \mathrm{En}(e, x))) = \mathrm{ev}(f, x)$.

A garbling schemes map the information:

$$(f, k, x) \rightarrow^{\mathrm{Gb}} (F, e, d, x) \rightarrow^{\mathrm{En}} (F, d, X) \rightarrow^{\mathrm{Ev}} (d, Y) \rightarrow^{\mathrm{De}} y \tag{1.4}$$

♢

To make the idea a garbling scheme more concrete, below is an example of how Alice and Bob would compute who is wealthier. The inputs to their computation are f, the function to be computed which is defined in equation 1.1, and their inputs x_a and x_b.

1. Alice has the function f to be computed, her input x_a, and Bob has his input x_b.
2. Alice hard-codes her inputs into the function, resulting in $\hat{f}$.
3. Alice runs the probabilistic algorithm $\mathrm{Gb}(\hat{f}, k)$. She now has (F, e, d).
4. Alice encodes all possible inputs that Bob could, all possible sequences of 0s and 1s, into X using En.
5. Alice sends (F, X, d) to Bob.
6. Bob runs $\mathrm{Ev}(F, X)$ to get Y. This computes the function, but the output, Y, is obscured.
7. Bob runs $\mathrm{De}(d, Y)$ to get y. This unobscures the output Y. Bob now has $y \in \{0, 1\}$. If $y = 0$, then Bob knows that Alice is wealthier. If $y = 1$, then Bob knows that he is wealthier.
8. Bob sends y to Alice.

The definition of a garbling scheme is a recently created definition, proposed for the first time ****, *** years after Yao's garbled circuit was proposed. The benefits of thinking of a garbled scheme in terms of this definition are significant. For one, it easier to map a theoretically garbling protocol into practice, i.e program it. The methods are already separated, whereas before the algorithm was generally one giant idea. When the methods are separated, the programmer does not need to intimately understand the security guarantees and pitfalls of the method, as the sequence of events that need to happen is explicit in the construction. A second reason that this definition of a garbling scheme is good is that it presents garbling as a larger cryptographic abstraction. Garbling schemes are simply methods for garbling a function and input, and this definition matches the intuition.

1.2.2 Projective Garbling Schemes

Perhaps a helpful abstraction to have later.

1.2.3 Security of Garbling Schemes

This section will give the definition of security of a garbling scheme.

1.2.4 MPC Security Definitions

A number of definitions which formalize what it means for a multiparty computation have been formalized. Yao, in his 1982 paper on secure function evaluation, says a 2PC protocol is secure if it has the property:

> If one participant behaves according the the protocol, the probability that the other participant successfully cheats is at most γ for $\gamma \in \{0, 1\}$.

Since Yao's original definition, many other definitions for the security of 2PC protocols have been proposed. Here we give Goldreich's definition of 2PC, which is given in his textbook *Foundations of Cryptography Volume II*, and is reproduced in various papers, most notably Lindell and Pinkas 2009 survey paper on multiparty computation **??**.

1.2.5 Goldreich and Lindell's Definition of Security for 2PC

Setup:

- Let $f = (f_1, f_2)$ be a probabilistic, polynomial time functionality.
- Let Π be a two party protocol for computing f.
- Define $\mathsf{view}_i^\Pi(n, x, y)$ (for $i \in \{1, 2\}$) as the view of the ith party on input (x, y) and security parameter n. $\mathsf{view}_i^\Pi(n, x, y)$ equals the tuple $(1^n, x, r^i, m_1^i, \ldots, m_t^i)$, where r^i is the contents of the ith party's internal random tape, and m_j^i is the jth message that the ith party received.
- Define $output_i^\Pi(n, x, y)$ as the output of the ith party on input (x, y) and security parameter n. Also denote

$$\mathsf{output}^\Pi(n, x, y) = (\mathsf{output}_1^\Pi(n, x, y), \mathsf{output}_2^\Pi(n, x, y)).$$

- Note that view_i^Π and output_i^Π are random variables whose probabilities are taken over the random tapes of the two parties. Also note that for two party computation.

Definition: We say that Π securely computes f in the presence of static semi-honest adversaries if there exists probabilistic polynomial time algorithms S_1 and S_2 such that for all $x, y \in \{0, 1\}^*$, where $|x| = |y|$, the following are true:

$$\{(S_1(x, f_1(x, y), f(x, y)))\}_{x,y} \equiv^C \{(\mathsf{view}_1^\Pi(x, y), \mathsf{output}^\Pi(x, y))\}_{x,y} \tag{1.5}$$

$$\{(S_2(x, f_2(x, y), f(x, y)))\}_{x,y} \equiv^C \{(\mathsf{view}_2^\Pi(x, y), \mathsf{output}^\Pi(x, y))\}_{x,y} \tag{1.6}$$

Intuition: We think of view_i^Π as all of the information that the ith has to operate with, such that any conclusion that the ith party can come to could be determined from $view_i^\Pi$. Moreover, output_i^Π is simply a complicated way of writing the output of the ith party. The value of output_i^Π is computable from the tuple view_i^Π.

Let's dig a little deeper into the meanings of equation 1.5 and 1.6. They state that a probabilistic, polynomial time algorithm, denoted S_1 and S_2, which is given access *only* to the party's input and output can compute the view of a party. For example, the definition requires that S_1 on input $(x, f(x, y))$ must be able to compute $\mathsf{view}_i^\Pi(x, y)$, in particular the messages received by party 1, such that the generated view is indistinguishable from the actual view. If there exists an algorithm that can perform the aforementioned task, then Π does not adequately conceal information, so we should not consider Π to be secure.

Finally, the definition requires that $|x| = |y|$; however, this constraint can be overcome in practice by padding the shorter input.

The definition of security provided here only applies when adversaries are semi-honest (see **??**). Definitions of security in settings with malicious adversaries require substantially more complexity. As a result, these definitions are often simulation based definitions. They imagine an ideal world, where the function f must be computed securely, and by a series of comparisons, show that the real world where Π computes f is essentially the same as the ideal world. For an easy to understand security definition of 2PC with malicious adversaries, we refer to reader to **?**.

1.2.6 Yao's Garbled Circuit

We now give an implementation of a generic 2PC protocol created by Yao **?**. The protocol works for two parties; we will call party 1 Alice, denoted A, and party 2 Bob, denoted B, who have inputs x and y respectively. Suppose f is the function that Alice and Bob wish to compute.

Yao's protocol depends on first encoding the function f as a circuit, as discussed in more detail in section **??**, and then Alice and Bob together evaluate the circuit.

Algorithm 2 Garble Circuit

```
Input: Circuit $f(x, \cdot)$
Output: Populate garbled tables $f(x, \cdot).tables$.
  for wire $w_i$ in $f(x, \cdot).wires$ do
      Generate two encryption keys, called garbled values, $W_i^0$ and $W_i^1$.
      Assign $(W_i^0, W_i^1)$ to $w_i$.
  end for

  for gate $g$ in $f(x, \cdot).gates$ do
      Let $w_i$ be $g$'s first input wire.
      Let $w_j$ be $g$'s second input wire.
      Let $w_k$ be $g$'s output wire.
      for $(u, v) \in \{(0,0),(0,1),(1,0),(1,1)\}$ do
          $T_g[u, v] = \text{Enc}_{W_i^u}(\text{Enc}_{W_j^v}(W_k^{g(u,v)}))$
      end for
      $f(x, \cdot).tables[g] = T_g$.
  end for
```

Step 0: Setup

Alice and Bob want to compute the function $f(x, y)$, where $x, y \in \{0,1\}^*$. Alice is the garbler, and will create the circuit. Bob is the evaluator, and will compute the circuit. Alice first hardwires her inputs into the circuit, yielding a circuit that computes $f(x, \cdot)$.

Step 1: Garbling the Circuit

Alice garbles each gate, according to the algorithm outlined in Algorithm **??**, which produces a table T_g for each gate g. The table enables the computation of g, if one is given either W_i^0 or W_i^1 and either W_j^0 or W_j^1 where wire i and j are the input wires to g. Figure **??** gives an example of a table for an AND gate. **TODO**

Algorithm 3 Evaluate Circuit

```
Input: $(input\_wires, tables, gates)$
  for Input wire $w_i$ in $input\_wires$ do          ▷ retrieve garbled values of input wires
      Perform OT$(w_i, x_i)$                          ▷ retrieve $W_i^{x_i}$ from Alice
      Save the value to $w_i$.
  end for
  for Gate $g$ in $gates$ do                          ▷ compute the output of each gate.
      Let $w_i$ be $g$'s first input wire.
      Let $w_j$ be $g$'s second input wire.
      Let $w_k$ be $g$'s output wire.
      Require $w_i$ and $w_j$ have been assigned garbled values.
      Require $w_k$ has not been assigned a garbled value.
      for $(u, v) \in \{(0,0), (0,1), (1,0), (1,1)\}$ do
          $temp = \mathrm{Dec}_{w_j}(\mathrm{Dec}_{w_i}(tables[g][u, v]))$
          if $temp$ decrypted correctly then
              $w_k = temp$
          end if
      end for
  end for
```

Step 2: Bob's Input and Computing the Circuit

Alice sends the garbled circuit to bob, which consists of the garbled tables, $f(x, \cdot).tables$, and the rules for connecting the gates together. In order for Bob to compute the circuit, he needs the garbled values of all input wires. Once he has the garbled values of the input wires, he can decrypt the first few gates, and acquire the decryption keys of the other gates until he has the keys to decrypt all of the gates in the circuit, yielding the output. Bob can acquire the garbled values of the input wires from Alice using 1-out-of-2 oblivious transfer on each input wire (see section **??**). If necessary, Bob sends the final output of the circuit to Alice. This is necessary only if $f_1(x, y) = f_2(x, y)$. Bob's protocol is outlined in more detail in Algorithm 3.

Explanation of the security of Yao's protocol

To do.

Notes about complexity

1 OT per (input?) wire. How much encryption? Not good enough for practice.

1.2.7 GMW

Where Yao's protocol is premised on encrypting gates individually, GMW's protocol for garbling circuits is premised on secret sharing, and performing operations on the shared secrets. Secret sharing, in its general idea, is class of methods for distributing a secret to a group of participants, where each participants is allocated a *share* of the secret. The secret can only be reconstructed when a sufficient number of the participants combine their shares, but any pool of insufficient shares yields no information about the secret.

GMW begins by having Alice and Bob secret share their inputs, so each party now has a collection of *shares*. Algorithm 4 describes this process in more detail. Then Alice and Bob perform a series of operations on their shares, which are dictated by the gates in the function they wish to compute. As with Yao's protocol, a gate may either compute XOR, AND or NOT. Each operation requires a different series of operations, which are described in Algorithm 5. Finally, Alice and Bob publicize their shares to each other, at which point each party will have sufficient shares to compute the output of the function.

Algorithm 4 GMW Setup

```
Alice does the following on input Circuit $f(x,\cdot)$ and $x = x_0 x_1 \dots x_n$
for wire $w_i$ in $f(x,\cdot).wires$ do
    Assign $a^1_{w_i} \leftarrow \{0,1\}$        ▷ a uniform random selection of 0 or 1.
    Assign $b^1_{w_i} = x_i \oplus a^1_{w_i}$
end for
Bob does likewise on input Circuit $f(x,\cdot)$ and $y = y_0 y_1 \dots y_n$
Hence Alice has generated shares $\{a^1_w, b^1_w\}_w$
and Bob has generated shares $\{a^2_w, b^2_w\}_w$
Alice and Bob divide the shares such that Alice has all $a_w$ and Bob has all $b_w$.
```

Algorithm 5 GMW Gate Evaluation

```
XOR Gate    ▷ $x_i \oplus y_i = (a^1_{w_i} \oplus b^1_{w_i}) \oplus (a^2_{w_i} \oplus a^2_{w_i})$
Alice evaluates $a^1_{w_i} \oplus a^2_{w_i}$
Bob evaluates $b^1_{w_i} \oplus b^2_{w_i}$

AND Gate    ▷ $x_i \wedge y_i = (a^1_{w_i} \oplus b^1_{w_i}) \wedge (a^2_{w_i} \oplus a^2_{w_i})$
Alice samples $\sigma \leftarrow \{0, 1\}$    ▷ a uniform random selection of 0 or 1
Alice constructs table $T$:
for $(u, v) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ do
    $T[u, v] = (a^1_{w_i} \oplus u) \wedge (a^2_{w_i} \oplus v)$
    $s[u, v] = \sigma \oplus T[u, v]$.
end for
Do 1-4OT. Alice sends $(s[0, 0], s[0, 1], s[1, 0], s[1, 1])$
Bob selects result based on $(u, v) = (b^1_{w_i}, b^2_{w_i})$.

NOT Gate    ▷ $w_i = (\neg a_{w_i}) \oplus (\neg b_{w_i})$
▷ Evaluate the negative of a particular wire $w_i$.
$w_i = a_{w_i} \oplus b_{w_i}$
Let $a'_{w_i} = 1 \oplus a_{w_i}$    ▷ i.e. $a'_{w_i} = \neg a_{w_i}$
Let $b'_{w_i} = 1 \oplus b_{w_i}$    ▷ i.e. $b'_{w_i} = \neg b_{w_i}$
```

see mike rosulek first few slides for good images. illustrative about translating bits over to wire labels

Chapter 2

Improving MPC

A number of improvements have been made to Yao's garbled circuit since its inception in 1986. The first scheme for a garbled circuit, the one described in section **??** has the following requirements: **include chart here**. Let's look into how much work is required in classical garbled circuits. **mention that we look gate-wise** The first metric that we look at is the size of the garbled table. **check this definition**:The garbled table, if you call recall from earlier, is the set of ciphertexts that is sent from the garbler to the evaluator, like in figure **??**. Hence the size refers to the number of ciphertexts that an evaulator needs to compute a single gate. The XOR column gives the size of the garbled table for an xor gate, and naturaly the AND columns gives the of the garbled table for an AND gate. The cost associated with the size of a garbled table manifests in effecting the amount of bandwidth that needs to be transmitted between the garbler and evaluator. In order to evaluate the gate, the evaluator needs the entire garbled table (**research idea: what if they don't? what if they can know when to request more information, to reduce the average amount of bandwidth? maybe this could be used to reduce calls to OT?**), so reducing the size of the garbled table reduces bandwidth. It turns out that bandwidth contributes a large hit to the total time required to perform mpc with garlbed circuits, and as a result, many of the improvements to garbled circuits have specifically targted reducing the size of the garbled table down from 4 ciphertexts. **maybe mention lower bound?**

The second metric we use to analyze garbled circuits is eval cost. Eval cost is the amount of computation that the evaluator must perform in order to process a single gate. In the classical setting, the evaluator goes down the garbled table decrypting each ciphertext until one of the decrypts correctly. In the worst case, this requires 8 decryptions, because each wire is encrypted twice, once with the key from the first input wire and then again with the key from second input wire. If a dual-key cipher (an encryption scheme that takes two keys, see section **??**), then the evaluator only needs to perform four decryptions in the worst case. The metric that we consider is the garble cost of a gate. The garble cost of a gate is amount of computation that the garbler needs to perform. The improvements to MPC have generally increased the garble cost; having the garbler do more work, can reduce the size of the garble table which reduces bandwidth, and can reduce the amount of computation the evaluator needs to perform **fact check this**. Before the research of **?** and the research presented in this thesis, there were not substantial benefits to garbler preprocessing, that is having the garbler do work *before* the actual computation time. We call this having the garbler do offline work, because they can do the work at their leisure, like at night when computers are less used. Then the online work, when the actual comptuation is performed, is reduced.

As we walk through the improvements to Yao's garbled circuit, we are going to think about the improvements affect the three metrics described above. It should also be noted before we begin that these are improvements to the the computation of a single gate. Speedups on the gate level propagately widely through the computation of the entire garbled circuit. For example, the computation of AES now requires approximatley $40,000$ gates, so reducing the number of ciphertexts being transmitted per gate by 1, reduces the total amount of ciphertexts that need to be sent by $10,000$ - a substantial speedup indeed.

2.1 Point and Permute

Beaver, Micali and Rogaway contributed the first major improvement to Yao's garbled circuit in 1990. A detail that I may or may not (**check this**) have left out in the description of Yao's garbled circuit is that the order of the garbled table needs to be permuted. If the first key sent is always the key corresponding to the input wires's 0 keys, then the evaluator can learn what the value of the input wires is - a violation of security. The solution to this problem is easy: permute the order of the garbled table. Now the evaulator trial decrypts each of the 4 ciphertexts until there is a valid decryption . [1]

The *point and permute* technique speeds up the evaluator's computation of the garbled table by removing the need to trial decrypt the ciphertexts; instead, the garbler subtly communicates which single ciphertext to decrypt. Using point and permute, the garbler randomly assigns a select bit 0 or 1 to each wire label in a wire **Is it clear what a wire label is, as opposed to a wire? Should be by now.** The assignment of the select bit is independent of the truth value of the wire, so two things can happen. First, the garbler can permute the garbled table based on the select bits, and second, the select bits can be sent to the evaluator. Upon receving the garbled table, the evaluator knows exactly which ciphertext to decrypt based on the select bits of the input wires.

Point and permute slightly increases the amoount of garbler-side computation to substantially decrease the amount of evaluator computation. The garbler needs to

[1] There exists encryption/decryption schemes that in addition to decrypting a ciphertext will also output a bit indicating if the decryption occurred correctly. Using schemes like this, the evaluator can decrypt all 4 ciphertexts until one of them decrypts correctly, as indicated by the extra bit output by the decryption algorithm.

sample $n + q$[2] additional random bits. Without point and permute, the evaluator needs to decrypt 2.5 ciphertexts on average, hence we estimate that there are roughly $1.5(n + q)$ fewer decryptions required. The overall bandwidth is increased by 2 bits per wire, from 256 bits to 258 bits: a small constant increase. [3]

Select Bit	Wire Label
0	A_0
1	A_1
1	B_0
0	B_1

Select Bits	Encryption
(0,0)	$\text{Enc}_{A_0,B_1}(C_1)$
(0,1)	$\text{Enc}_{A_0,B_0}(C_0)$
(1,0)	$\text{Enc}_{A_1,B_1}(C_0)$
(1,1)	$\text{Enc}_{A_1,B_0}(C_0)$

$$\boxed{\begin{array}{l} C_0 \leftarrow \{0,1\}^n \\ C_1 \leftarrow \{0,1\}^n \end{array}}$$

Table 2.1: Garbled Gate for Point and Permute

Table 2.2: Example garbled gate using point and permute. The gate being computed is given in figure **Make it 23:30 in mike's talk**

add chart if you want

2.2 Garbled Row Reduction 3

Garbled Row Reduction 3 (GRR3) is technique proposed by **who** which decreases the number of ciphertexts that need be communicated between the garbler and evaluator. To use GRR3, the garbler must also use the point and permute method. Using Garbled Row Reduction, the garbler sets the ciphertext in the top row of the garbled table equal to a value that decrypts to 0^n. [4] Upon evaluating the garbled gate, if the evaluator sees that the select bits of the input wires indicate to decrypt the first row, the evaluator simply assumes the ciphertext be of value O^n.

Discuss security.

GRR3 does not have a significant effect on garbler side compuation. The difference being that for constructing some wire lables, the garbler may need to compute Enc^{-1} instead of generating random bits. The evaluator needs to perform slightly

[2] Recall from **todo some previous section** that $n + q =$ number of wires.

[3] The value is constant in the sense that it is independent of the security parameter.

[4] In previous constructions the value for the ciphertext was chosen randomly.

Select Bit	Wire Label
0	A_0
1	A_1
1	B_0
0	B_1

Select Bits	Encryption
(0,1)	$\text{Enc}_{A_0,B_0}(C_0)$
(1,0)	$\text{Enc}_{A_1,B_1}(C_0)$
(1,1)	$\text{Enc}_{A_1,B_0}(C_0)$

$$\boxed{\begin{array}{c} C_0 \leftarrow \{0,1\}^n \\ C_1 \leftarrow \text{Enc}^{-1}_{A_0,B_1}(0^n) \end{array}}$$

Table 2.3: Example garbled gate using point and permute and garbled row reduction 3. The gate being computed is given in figure **Make it 23:30 in mike's talk**

less computation: in the event that the first row needs to be decrypted, the evaluator doesn't need to perform the decryption algorithm. This events occur with probabilty $\frac{1}{4}$, so we can extrapolate that the evaluator needs to perform $\frac{1}{4}$ fewer decryptions than wwould be necessary if only PP were being used. GRR3 reduces the size of the garbled table from 4 ciphertexts to 3 ciphertexts, a 25% reduction.

2.3 Free XOR

The Free XOR technique was developed by **who** in **when**. The technique, as the name suggests, makes the computation of XOR gates essentially free. Say the garbler is determining the lables for wire A. The label associated with truth value 0, A_0, is randomly sampled from $\{0,1\}^n$. Then the label assocated with the truth value is set such that $A_1 \leftarrow A_0 \oplus \Delta$, where Δ is global, randomly sampled value from $\{0,1\}^n$. If the garbler is garbling an XOR gate, then they set C_0 to $A \oplus B$, and like wire A, the garbler sets $C_1 = C_0 \oplus \Delta$. With these wire labels, it turns out that the garbler doesn't need to send a garbled table: the evaluator can compute C_0 and C_1 by XORing the inputted wire labels. The math for this operation is shown below:

$$A \oplus B = C$$

$$(A \oplus \Delta) \oplus B = (A \oplus B) \oplus \Delta = C \oplus \Delta$$

$$A \oplus (B \oplus \Delta) = (A \oplus B) \oplus \Delta = C \oplus \Delta$$

$$(A \oplus \Delta) \oplus (B \oplus \Delta) = (A \oplus B) \oplus (\Delta \oplus \Delta) = C$$

Wire Label	Value
A_0	A
A_1	$A \oplus \Delta$
B_0	B
B_1	$B \oplus \Delta$
C_0	$A \oplus B$
C_1	$C_0 \oplus \Delta$

Table 2.4: Example XOR garbled gate wires using PP, GRR3 and Free XOR.

The Free XOR technique is also compatible with GRR3, but since XOR doesn't require a garbled table, GRR3 is only used on AND gates.

Using the Free XOR technique, the size of the garbled table is zero for all XOR gates and of size 3 for AND gates. This fact incentivizes the construction of circuits that optimize the number of XOR gates, minimize the number AND gates (while minimizing the size of the entire circuit of course). One interesting implication of using the Free XOR technique is that an added assumption must be made to our encryption algorithm. Since Δ is part of the key and part of the the payload[5] of the encryption algorithm, the encryption algorithm must be secure under the circularity assumption. Fortunately, modern encryption uses AES-128, which is presumed to be secure under the circularity assumption.

[5] The payload is the value that is being encrypted

2.4 Garbled Row Reduction 2

Garbled row reduction 2 (GRR2) reduces the size of the garbled table for AND gates to 2 ciphertexts, but it is not compatible with Free XOR, so XOR gates also require two ciphertexts. The approach of GRR2 is quite different from GRR2. At a high level, it creates two third degree polynomials, one based on the input wires which should output C_0, and a second polynomial which should output C_1. Since the polynomials are of three degrees, they are each uniquely generated by 3 points. To construct the polynomials, the garbler uses x, y, z for the first polynomail and u, v, w for the second polynomial. The garbler sends x, y, u, v to the evaluator, who can interpolate the polynomial and plug in values a, b, c.

GRR2 is incompatible with Free XOR because the ciphertext values cannot be set to the xor of the inputer wires (e.g. $C \leftarrow A \oplus B$), since the value of C is always determined by the polynomial.

Add to this, but no need to spend too much time on it since it's not used

2.5 FleXOR

Before the creation of FlexXOR, MPC was at an awkward point where some circuits with a large amount of xors could be computed faster with Free XOR and circuits with fewer XORS could be computed faster with GRR2. FleXOR serves as a method to reconcile GRR2 with Free XOR, solving the problem where the ciphertext values of AND gates are independent of Δ.

FleXOR's construction is straightforward: use GRR2 on AND gates, and use Free XOR on XOR gates, correcting the delta value where necessary. To correct the delta value, FleXOR adds in a unary gate that maps $A, A \oplus \Delta_1 \rightarrow A', A' \oplus \Delta_2$.

Suppose we have the following situation: we computing an XOR gate with input wires A, B and output wire C. Input wires A and B have each come from a different AND gate, so their labels are the result of the polynomial interpolation of GRR2. To perform Free XOR, A, B and C need to be using the same delta value. The garbler adds an extra gate between A and B and the XOR gate that adjusts their XOR value to the correct value. The unary[6] gate maps $A, A \oplus \Delta_1 \rightarrow A', A' \oplus \Delta_2$, where Δ_2 is the correct delta value for the XOR gate.

A garbled AND table costs 2 ciphertexts, and a garbled XOR table costs $0, 1$ or 2 ciphertexts depending on how many ciphertexts need to be corrected. This turns the creation of the circuit into a combinatorial optimization problem, where the offset of each wire needs to be determined to minimize the total cost of each XOR gate and be subject to the compatibility of GRR2 for AND gates. Without doing too much (if any) optimization of the circuit, FleXOR usually requires 0 or 1 ciphertext in practice.

2.6 Half Gates

Half Gates is the most recent technique developed to improve Yao's garbled circuit. The goal was to make AND gates cost 2 ciphertexts, and for Free XOR to always be free. The methodology at high level is to make AND gates rely on Free XOR, so that AND gates result in values with the correct Free XOR delta.

[6] Unary means has one input and one output.

Mode Transition Times Duration (Typical)	**Size (xλ)**		**Eval Cost**		**Garble Cost**		**Assumption**
	XOR	**AND**	**XOR**	**AND**	**XOR**	**AND**	
Classical	1365	1260	1024	μ s	a	b	a
Point and Permute	TBD	TBD	TBD	μ s	a	b	a
GRR3	TBD	TBD	TBD	μ s	a	b	a
Free XOR	TBD	TBD	TBD	μ	a	bs	a
GRR2 XOR	TBD	TBD	TBD	μ	a	bs	a
FleXOR	TBD	TBD	TBD	μ	a	bs	a
Half Gates	TBD	TBD	TBD	μ	a	bs	a

Table 2.5: Summary of Garbled Circuit Improvements. GRR3 stands for garbled row reduction 3 and GRR2 stands for garbled row reduction 2

Chapter 3

Template tips

This is the first page of the first chapter. You may delete the contents of this chapter so you can add your own text; it's just here to show you some examples.

3.0.1 Footnotes and Endnotes

You might want to footnote something.[1] Be sure to leave no spaces between the word immediately preceding the footnote command and the command itself. The footnote will be in a smaller font and placed appropriately. Endnotes work in much the same way. More information can be found about both on the CUS site.

3.1 Bibliographies

Of course you will need to cite things, and you will probably accumulate an armful of sources. This is why BibTeX was created. For more information about BibTeX and bibliographies, see our CUS site (`web.reed.edu/cis/help/latex/index.html`)[2]. There are three pages on this topic: *bibtex* (which talks about using BibTeX, at `/latex/bibtex.html`), *bibtexstyles* (about how to find and use the bibliography

[1] footnote text

[2] **?**

style that best suits your needs, at `/latex/bibtexstyles.html`) and *bibman* (which covers how to make and maintain a bibliography by hand, without BibTeX, at at `/latex/bibman.html`). The last page will not be useful unless you have only a few sources. There used to be APA stuff here, but we don't need it since I've fixed this with my apa-good natbib style file.

3.1.1 Tips for Bibliographies

1. Like with thesis formatting, the sooner you start compiling your bibliography for something as large as thesis, the better. Typing in source after source is mind-numbing enough; do you really want to do it for hours on end in late April? Think of it as procrastination.

2. The cite key (a citation's label) needs to be unique from the other entries.

3. When you have more than one author or editor, you need to separate each author's name by the word "and" e.g.
   `Author = {Noble, Sam and Youngberg, Jessica},`.

4. Bibliographies made using BibTeX (whether manually or using a manager) accept LaTeX markup, so you can italicize and add symbols as necessary.

5. To force capitalization in an article title or where all lowercase is generally used, bracket the capital letter in curly braces.

6. You can add a Reed Thesis citation[3] option. The best way to do this is to use the phdthesis type of citation, and use the optional "type" field to enter "Reed thesis" or "Undergraduate thesis". Here's a test of Chicago, showing the second cite in a row[4] being different. Also the second time not in a row[5] should be

[3] ?

[4] ?

[5] ?

different. Of course in other styles they'll all look the same.

3.2 Anything else?

If you'd like to see examples of other things in this template, please contact CUS (email cus@reed.edu) with your suggestions. We love to see people using LaTeX for their theses, and are happy to help.

Conclusion

Here's a conclusion, demonstrating the use of all that manual incrementing and table of contents adding that has to happen if you use the starred form of the chapter command. The deal is, the chapter command in LaTeX does a lot of things: it increments the chapter counter, it resets the section counter to zero, it puts the name of the chapter into the table of contents and the running headers, and probably some other stuff.

So, if you remove all that stuff because you don't like it to say "Chapter 4: Conclusion", then you have to manually add all the things LaTeX would normally do for you. Maybe someday we'll write a new chapter macro that doesn't add "Chapter X" to the beginning of every chapter title.

4.1 More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

The First Appendix

Appendix B

The Second Appendix, for Fun

References

Bellare, M., Hoang, V. T., Keelveedhi, S., & Rogaway, P. (2013). Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium of Security and Privacy*, (pp. 478–492).

Bellare, M., Hoang, V. T., & Rogaway, P. (2012). Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, (pp. 784–796). ACM.

Lindell, Y., & Pinkas, B. (2009). Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, *1*(1), 5.

Snyder, P. (????). Yaos garbled circuits: Recent directions and implementations.