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

MPC is a complex construction with many moving parts. Each moving part is itself a unique cryptographic primitive, which when combined give rise to MPC. The following sections give a summary of the crytographic tools required for MPC.

1.1.1 Encryption

Encryption is the process of encoding a message such that only parties with the key can read the message. As we discuss MPC, we will be using symmetric-key encryption, the case of encryption where encryption and decryption use the same key. Public key encryption, in constrast, uses two different keys: one key for encryption and a different key for decryption. While early MPC schemes used public key encryption, modern methods do not require public key encryption, and symmetric-key encryption algorithms are faster, making them preferable.

A symmetric-key encryption protocol is composed of two parts. The first part is the encryption algorithm which disguises the message, and the second part is the decryption algorithm which unobfuscates the disguised message. We notate the decryption algorithm, Dec as Enc^{-1}. We notate encryption and decryption with

$$\begin{aligned} \text{Enc}_k(pt) &= ct \\ \text{Enc}_k^{-1}(ct) &= pt \end{aligned} \tag{1.1}$$

where pt stands for plaintext and is the original message, ct stands for ciphertext and is the encrypted message, and k is the secret key, that only authorized readers of the message hold. For the algorithm to be effective, the secret key k must be a randomstring of 0s and 1s of length λ. [1]

λ is the security parameter of our protocol. If we increase λ, thereby increasing

[1] The notion of randomness in cyptoraphy has a precise definition, and in cases where λ is large, it is sufficient for k to be pseuodorandom. Pseudorandom also has precise cryptographyic definition.

the size of the key, then the encyrption algorithm becomes harder for an adversary to break.

It is useful for MPC to define a specific type of symmetric-key encryption algorithm called a Dual-Key Cipher (DKC) **?**. A DKC requires two secret keys to encrypt and decrypt the message, in constrast to classic encryption which only requires one. It is easy to instantiate a DKC if one has a secure encryption scheme: let k_0 and k_1 be the two secret keys, and instantiate the DKC as follows:

$$\begin{aligned} \mathrm{EncDKC}_{k_0,k_1}(pt) &= \mathrm{Enc}_{k_1}(\mathrm{Enc}_{k_0}(pt)) \\ \mathrm{EncDKC}^{-1}_{k_0,k_1}(ct) &= \mathrm{Enc}^{-1}_{k_0}(\mathrm{Enc}^{-1}_{k_1}(ct)) \end{aligned} \tag{1.2}$$

This construction of a DKC is slow, and there are many faster methods for instantiating DKCS. For more information, see **?**.

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 done inside of a circuit are performed 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 **??** 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 or algorithm. **Find details and citation** In other words, if there's some algorithm that do it, then there is some

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.

circuit that can do it as well. Figure **??** shows the circuit representation of the less than function, f as specified in equation **??**.

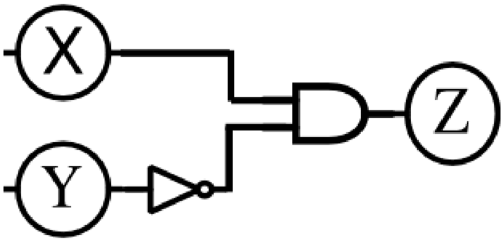


Figure 1.1: A circuit that computes the less or equal to function, equivalent to f for input of two one-bit values. **TODO, add truth table?**

1.1.3 Technical Definition of a Circuit

It is useful at times to have a formal definition of a circuit. Here we give a formal definition of a circuit proposed by BHR in **?**.

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 $\text{Wires} = \{1, \dots, n+q\}$, $\text{InputWires} = \{1, \dots, n\}$, $\text{OutputWires} = \{n+1-m+1, \dots, n+q\}$, and $\text{Gates} = \{n+1, \dots, n+q\}$. Then $A : \text{Gates} \to \text{Wires} \setminus \text{OutputWires}$ identifies each gate's first incoming wire. And $B : \text{Gates} \to \text{Wires} \setminus \text{OutputWires}$ identifies each gate's second incoming wire. Finally, $G : \text{Gates} \times \{0,1\}^2 \to \{0,1\}$ identifies the functionality of each gate.

To evaluate a circuit, we use a canonical algorithm ev_{circ}, which takes as input a function f (encoded as a string) and a string $x = x_1 x_2 \dots x_n$ 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 $x_{n+q-m+1} \ldots x_{n+q}$
```

1.1.4 Computational Indistinguishability and the DDH Assumption

Add in computational indist defn here. use lindellpinkas paper Check if this is ever used - it is used. see eqn 1.6, ensebles Give encryptiong example?

When a cryptographic scheme is said to be *secure*, cryptographers actually mean something much more precise. It means that an adversary can beat the scheme only if the adversary can tackle the hardness assumptions on which the scheme is based. Hardness assumption, in the way I'm using it, means a problem, like factoring a number or finding the greatest common divisor of two numbers. At a high level then a cryptographic scheme is secure so long as there is no efficient algorithm for solving the underlying problem.

For example, say we have some encryption scheme with has algorithms Enc and Enc^{-1}. Then in order to make the statement "Enc is secure" substantial and precise, we might say that "an adversary can determine m given Enc$_k(m)$ and the Enc algorithm if and only if the same adversary can solve problem P". The real trick for making a good encryption algorithm is threefold: (1) finding a problem P which is hard to solve, (2) making Enc dependent on P, and (3) ensuring that Enc^{-1} is efficient to solve (because we need to be able to decrypt our messages quickly).

This formulation of security turns out to be useful for a number of reasons. One reason is that we will know when the scheme becomes insecure. If the underlying problem, like factoring large numbers, becomes solvable efficiently, then we immediately know that our scheme in insecure. There is a major disadvantage to this formulationation. Once researchers found a few good hardness assumption, almost all cryptographic schemes became dependent on these problems; all of the eggs were put in a few baskets. So if one big problem is found to have an efficient solution, such as factoring prime numbers, then many cryptographic schemes will break.

A common hardness assumption is the Decisional Diffie-Helman Assumption (DDH Assumption), an assumption about solving a problem concerning discrete logs.

Informally, the DDH assumption is:

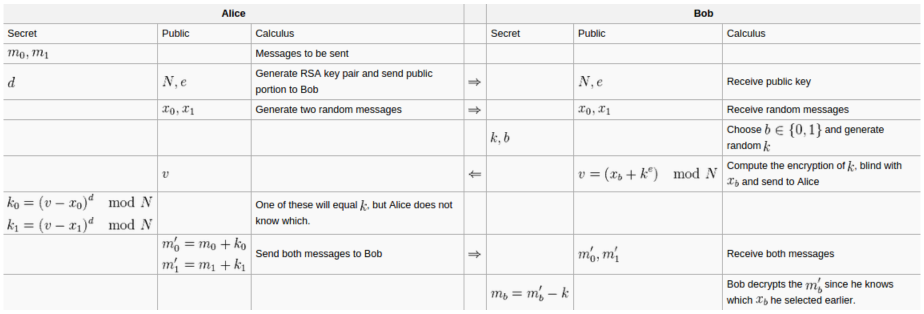
$$\begin{aligned}&\text{Let } G \text{ be a group of order } q \text{ with generator } g.\\&\text{Let } a, b \text{ and } c \text{ be random elements from } \mathbb{Z}_q.\\&\text{Then, } (g^a, g^b, g^{ab}) \approx_D (g^a, g^b, b^c).\end{aligned} \tag{1.3}$$

Give encryptiong example?

1.1.5 Oblivious Transfer

TODO Remove gender 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



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.

relies on the DDH assumption (see section **??**) 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.

1.2 Classic 2PC

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 developed by Yao himself, and the other was developed 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, which has since been termed garbled circuit. At this point, it is unclear which method is better, so researchers continue to study both methods.

Section xxx gives the definition gives a definition of security for an MPC protocol. Section yyy gives ...

1.2.1 2PC Security Motivation

Think back to Alice and Bob in section xxx. Alice and Bob were millionaires who wanted to determine who was wealthier without disclosing how much wealth they had. Formally, we say that Alice has input x and Bob has input y (x and y are the wealth of each party), and they want to compute the greater than function f, such that **Is this equation still somewhere else**

$$f = . \tag{1.4}$$

We call the overarching interaction between Alice and Bob the protocol Π, and it consists of all messages exchanged and computations performed. Based on the setup of the problem, we can list a few properties that Alice and Bob wish Π to have.

Privacy Parties only learn their output. Any information learned by a single during the execution of Π must be deduced from the output. For example, in the case of Alice and Bob, if Alice learns that they had more money, then they learn that $y < x$, which is some information about y, but is deduced from the output so it is reasonable. It would be unreasonable if during the execution of Π Alice learns that $1,000,000 < y < 2,000,000$, as that information is not deducable from $f(x, y)$.

Correctness Each party receives the correct output. In the case of Alice and Bob, this simply means that they learn correclty who has more money.

Fairness Each party receives their input if and only if the other party receives their input. In the case of Alice and Bob, this means that Alice receives their input if and only if Bob recieves their input. This property prevents the case where Alice sends her input to Bob, Bob computes f with Alice's input and their own input, and then Bob refuses to send his input to Alice. Hence Bob has learned to output of f, but Alice hasn't. We do not want to this scenario to happen, so we require that each party receive their input if and only if the party does. We do not require that all parties always receive their inputs, because a party could abort from the execution in the middle of the protocol. [2]

Semi-honesty Each party must obey the protocol. We are assuming that Alice and Bob are semi-honest agents, which means that Alice and Bob are not malicious, trying to steal information from each other. They are simply trying to compute f, but would use the information that they receive in order to deduce information about the other party. In contrast, we could imagine the scenario where Alice and Bob are untrustworthy: they would send lies, send the wrong input, or deviate in any number of ways to get the upper hand in the protocol. While assuming semi-honesty is not the most realistic scenario, most cryptographic protocols secure with semi-honest agents can be transformed into stronger protocols protecting against malicious agents. Morever, the semi-honest protocols are more intuitive.

These properties themselves do not constute definition, but rather any secure protocol should have these properties. One possible method for making a definition of security would be to list a number of properties a secure protocol must have. This approach is unsatisfactory for a number of reasons. One reason is that an important property of security may be missed. Many different applications of 2PC have different requirements, and which means that an important requirement may be missed in corner cases. Ideally, a good definition of 2PC works for all applications, henc capturing all of those requirements. A second reason that the property based definition is unsatisfactory is that the definition should be simple. If the definition is simple, then it should be clear that *all* possible attacks against the protocol are prevented by

[2] Technically this is incorrect since we assume that each party is semihonest, but the property generalizes more easily when this requirement when considering aborts.

the definition.

1.2.2 2PC Security Definition

A number of definitions formalize what it means for a 2PC protocol to be secure. 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 $\gamma \in \{0, 1\}$.

TODO mention that this doesn't use the properties.

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 **??**.

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:

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

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

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 **??** and **??**. 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 **?**.

1.2.3 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
          $T_g[u, v] = \mathrm{Enc}_{W_i^u}(\mathrm{Enc}_{W_j^v}(W_k^{g(u,v)}))$
      end for
      $f(x, \cdot).tables[g] = T_g$.
  end for
```

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 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**

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 **??**.

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.

1.2.4 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 **??** 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 **??**. 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:

$$
\begin{aligned}
A \oplus B &= C \\
(A \oplus \Delta) \oplus B = (A \oplus B) \oplus \Delta &= C \oplus \Delta \\
A \oplus (B \oplus \Delta) = (A \oplus B) \oplus \Delta &= C \oplus \Delta \\
(A \oplus \Delta) \oplus (B \oplus \Delta) = (A \oplus B) \oplus (\Delta \oplus \Delta) &= C
\end{aligned}
$$

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 improvement Yao's garbled circuit. The goal of Half Gates is to make AND gates cost two ciphertexts, while preserving properties necessary for Free XOR. I'm going to introduce Half Gates in a different way from the original paper.

We consider the case of an AND gate $c = a \wedge b$, where the generator knows the value of a. If $a = 0$, then the generator will create a unary gate that always outpost false, c. Otherwise if $a = 1$, then the generator create a unary identity gate that always outputs b. Table **??** shows the garbled gates for different values of a.

Since the evaluator has the wire label corresponding to b (either B or $B \oplus \Delta$), the evaluator can compute the label of the output wire of the AND gate by xoring the rows in the garbled table by $H(B)$ or $H(B \oplus \Delta)$ to get C or $C \oplus \Delta$ respectively.

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

Garbled Table for $a = 0$	Garbled Table for $a = 1$		Garbled Table for any a
$H(B) \oplus C$	$H(B) \oplus C$	→	$H(B) \oplus C$
$H(B) \oplus C$	$H(B) \oplus C \oplus \Delta$		$H(B) \oplus C \oplus a\Delta$

Table 2.5: Generator's Garbled Half Gate for $a = 0$, $a = 1$, and written more succinctly with $a\Delta$ for $a \in \{0, 1\}$. If $a = 0$, then $a\Delta = 0$. Otherwise if $a = 1$, then $a\Delta = \Delta$.

A further improvement can be garnered by using the garbled row-reduction trick. We choose C such that the first of the top row of the garbled table is the all zeros ciphertext. The top row may not necessarily be $H(B) \oplus C$, since for security the rows are permuted. Because the top row is all 0s, it does not need to be sent to the evaluator. If the evaluator should decrypt the cipher text on the top row (as directed by point and permute), then the evaluator assumes the cipher text to be all 0s. Overall, computing $a \wedge b = c$ requires two having operations by the generator, a single hash operation by the evaluator, and the communication of one ciphertext.

We now consider computing $a \wedge b = c$, where the evaluator somehow already knows the value of a. If $a = 0$, then the evaluator should acquire C. Otherwise if $a = 1$, then the evaluator should acquire $C \oplus b\Delta$, in which case it is sufficient for the evaluator to obtain $\Omega = C \oplus B$ (then xor Ω with the wire label corresponding to b). Table **??** shows cipher texts which the garbler gives to the evaluator.

This table is different from other garbled tables. First, the table does not need to be permuted. Secondly, evaluation is different. If $a = 0$, then the evaluator uses wire label A to decrypt the top row of the table and acquire C. If $a = 1$, then the evaluator uses $A \oplus \Delta$ to decrypt the second row, yielding $C \oplus B$. The evaluator then xors wire label $B + b\Delta$ by $C \oplus B$ to obtain $C \oplus b\Delta$. As before, the ciphertext on the top row does not need to be communicatd by using the garbled row-reduction trick. The generator sets C such that $H(A) \oplus C = 0$ (i.e. $C = H(A)$). The total cost of this half gate is the same as before: two hashes by the generator, one hash by the evaluator, and once cipher text.

Garbled Table for any A
$H(A) \oplus C$
$H(A \oplus \Delta) \oplus C \oplus B$

Table 2.6: Evaluator's half gate garbled table.

We now put the two half gates together to form an AND gate. Consider the following where r is a random bit generated by the generator:

$$a \wedge b = a \wedge (r \oplus r \oplus b) = (a \wedge r) \oplus (a \wedge (r \oplus b)). \tag{2.1}$$

The first AND gate, $a \wedge r$, can be computed with a generator-half-gate - the generator "knows" r. Furthermore, if we can let the evaluator know the value of $r \oplus b$, then the second AND gate, $(a \wedge (r \oplus b)$, can be computed with an evaluator-half-gate - the evaluator "knows" $r \oplus b$. And the final XOR can be computed with free xor at the cost of no ciphertexts.

It is secure for the garbler to give $r \oplus b$ to the evaluator, since r is random and blinds the value of b. The value of $r \oplus b$, while only a single bit, can be communicated to the evaluator for free: use the select bit (from the point and permute technique) of the false wire label on wire b (so r is the select bit on the true wire label of wire b).

The overall cost of using Half Gates for AND gates is four calls to H for the generator, two calls to H for the evaluator, and the communication of 2 ciphertexts. Half Gates guarantees only two ciphertexts are needed per AND gate, but the tradeoff is the additional computation for both parties (i.e. computing H). With FleXOR, the number of ciphertexts that need to be communicated may vary, but there is less computation required.

When MPC becomes used in real operations, the circuit will likely not be optimized for XOR gates (reducing the number of AND gates). In this case, where an arbitrary function is being computed, it is hypothesized that Half Gates will outperform FleXOR. But in many of the cases being examined now - most notably computing AES - the circuit has been optimized heavily to have very few AND gates, hence the benefits of Half Gates are less noticeable.[7]

Garbled Circuit Improvement	**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	2	0	2	0	4	0	a

Table 2.7: Summary of Garbled Circuit Improvements. GRR3 stands for garbled row reduction 3 and GRR2 stands for garbled row reduction 2

[7] At present, the AES circuit is 80% XOR gates

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