Implementing Improvements for Secure Function Evaluation

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 2016

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:

{AL-0: Do I need this?}

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

{AL-2: in primitives, should I clarify what I mean by "break an algorithm"?} {AL-3: Needs to be redone to emphasize chaining, and the work that I did. Right now, it just generally motivates SFE}

Secure Function Evaluation (SFE) is the study and creation of protocols for securely computing a function between multiple parties. Secure, in this context, means that the protocol prevents each party from learning anything about the other parties.

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?

{AL-4: check if using greater than or less than function} Alice and Bob can solve their problem by specifying a function f; in this case f takes two inputs x and y and outputs 1 if $x < y$, and 0 otherwise (f is the less than function). If Alice and Bob can secretly give f their amount of wealth and then retrieve the answer from f, then their problem is solved: they have securely computed who is wealthier.

The goal of SFE broadly is to give Alice, Bob and their friends the ability to securely compute an arbitrary function at their will. For the protocl to be secure, it needs to have the following informal properties:

- **Privacy:** Each party's input is kept secret.

- **Correctness:** The correct answer to the computation is computed.

Originally, the goal of the research community was to develop a secure protocol, define security, and prove the protocol is secure. In more recent times, the focus has shifted to making SFE protocols faster.

If SFE could 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 example is running algorithms on confidential medical data. Consider the case where various hosptitals want to joinly run algorithms on their data for medical research. The hosptical need guarantees, for both legal and ethical reasons, that the data will not be disclosed as the algorithm is being run, as medical data can be very sensitive. The hopsitals use could use SFE to run computation on all of their data, mainintaining the privacy of their patients.

Since research into SFE has focused on creating a method by which an arbitrary function can be computed securely, the application of SFE may be beyond what we can presently conceive of. It's not unlikely that SFE will become a standard in the internet, where when you access the internet, behind the scenes your access is being is plugged into an SFE protocol, sent off to another computer to do some processing. For this future to be realized, work need to be done to make SFE faster and more flexible.

Chapter 1

Cryptographic Primitives

Secure Compuation (SC) is the study of computing functions in a secure fashion. SC is split into two cases: cases where there are two parties involved, referred to as two-party computation (2PC), and cases where there are three or more parties, referred to as multiparty computation (MPC). This thesis will focus primarily on 2PC protocols, but many of the methods are applicable to MPC as well.

2PC protocols are complex cryptographic protocols that rely on a number of cryptographic primitives. In order to understand 2PC, it is not crucial to understand how the cryptographic primitives work, but it is important to understand their inputs, outputs and security guarantees. This first chapter will give an overview of the cryptographic primitives used in 2PC protocols.

1.1 Introducing Cryptographic Security

The goal of this section is to explain cryptographic security, starting at an intuitive level and moving outward. We do not present cryptographic security in a comprehensive fashion; rather, we explain cryptographic security with the goal of explaining 2PC protocols and their security. For more information on cryptographic security, we encourage the reader to peruse Katz & Lindell (2007).

We define a few intuitive terms to get started. A *cryptographic scheme* is a series of instructions designed to perform a specific task. An *adversary* is an algorithm that tries to *break* the scheme. If an adversary *breaks* a scheme, then the adversary has learned information about inputs to the scheme that they shouldn't.

The goal of defining security in cryptography is to build a formal definition that matches real world needs and intuitions. A good starting place is to consider perfect security. A scheme is perfectly secure if no matter what the adversary does, they cannot break the scheme. Even if the adversary has unlimited computational power, in terms of time and space, an adversary cannot break a perfectly secure scheme.

However, perfect security is not the most useful way to think of security, because it makes forces schemes to be slow and communication intensive. We relax the definition of security by requiring that the adversary run in polynomial time. This substantially reduces the power of the adversary, and it matches our intuition. We are really only concerned with what adversaries can reasonably achieve, as opposed to theoretically possible.

Because computers have improved drastically over the years, what was previously considered a reasonable adversary is not what is considered a reasonable adversary today. Modern computing advances have created easy access to faster computation, meaning that modern adversaries can solve harder problems than they could in previous years. As a concrete example, consider an giving an adversary the following problem: find the factors of N. The average computer today can solve the problem for a larger N than the average computer a decade ago.

Changing computational power makes it important to scale how hard a cryptographic scheme is to break. To this end, we introduce a *security parameter*, denoted as λ. A security parameter is a positive integer that represents how hard a scheme is to break. A larger security parameter should mean that the scheme is more difficult to break. More specifically, the security parameter is correlates to the input-size

of the problem underlying the cryptographic scheme. For example, if the underlying problem is factoring large number N, then $N = \lambda$. As N and λ scale up, the factoring problem becomes more difficult and the scheme becomes hard to break.

Finally, we acknowledge that adversaries have access to some random values, hence we strengthen adversaries to be probabilistic algorithms. Probabilistic means that the algorithms have access to a string of uniform random bits, with the implication being that the algorithm is capable of guessing.

In order to reason about the security of cryptographic schemes, it is useful to think about breaking a scheme in terms of a probability. For example, we want to be able to say that the best adversary, that is best probabilistic polynomial-time algorithm, has some probability p of breaking the scheme. We note that p is nonzero, since the adversary can always guess and be right with some nonzero probability. To achieve a probability based formalism, we introduce a negligible function. Informally, a negligible function is smaller than the reciprocal of all polynomial functions. Formally, a negligible function is:

Definition 1 A function $\mu : \mathbb{N} \to \mathbb{R}$ is negligible if for all positive polynomial $p(\cdot)$, there exists positive integer N_p such that for all $x > N_p$,

$$|\mu(x)| < \frac{1}{p(x)}. \tag{1.1}$$

Goldreich (1995) ♢

Examples of negligible functions include 2^{-n}, $2^{-\sqrt{2}}$ and $n^{-\log n}$.

To put a negligible function to use, say an adversary is attacking a cryptographic scheme that is equivalent to solving a problem P with input-size λ and 2^{λ} possible answers. Moreover, say that P is known to be NP-hard such that there is no polynomial time algorithm to solve P. Then, the best that the adversary can do is to guess the answer to P. Hence the probability that the adversary that finds the answer the

P, or breaks the scheme, is

$$Pr[\text{A correctly answers } P] = 2^{-\lambda}$$

Since $2^{-\lambda}$ is a negligible function, we say that the adversary has a negligible probability of breaking the scheme.

In summary, we model an adversary as a probabilistic polynomial-time algorithm. This limits the computational power of the adversary to what is reasonably computable in reality. Moreover, we can scale the security of a scheme or problem by changing λ, the security parameter. A higher security parameter makes the scheme more difficult to break.

1.2 Encryption

Encryption is the process of obfuscating a message, and then later un-obfuscating the message. Say Alice has a message that she wants to send to Bob, but somewhere between Alice and Bob sits Eve, who wants to learn about the message. An encryption scheme enables Alice to send her message to Bob with confidence that Eve cannot learn any information about the message.

An encryption scheme is composed of three algorithms: Enc, Enc^{-1} and Gen; formally, we say an encryption scheme is a tuple $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}^{-1})$[1]. Enc the obfuscating algorithm, Enc^{-1} is the un-obfuscating algorithm and Gen generates a key. The key is extra information that Enc and Enc^{-1} use to obfuscate and un-obfuscate the message respectively. The key, denoted k, is a random[2] string of λ bits, that is k is randomly sampled from $\{0,1\}^{\lambda}$ where λ is the security parameter of the

[1] We use Π here to denote the encryption scheme, because it is a protocol. Protocol starts with a p.

[2] The notion of randomness in cryptography is precisely defined, and in cases where λ is large, it is sufficient for k to be pseudorandom. Pseudorandomness is also precisely defined.

encryption scheme. As per the discussion on security parameters, as λ increases and k grows in length, an encryption scheme should become harder to break.

Enc, the encryption algorithm, takes a message and the key as input and outputs an obfuscated message. Enc^{-1}, the decryption algorithm, takes the encrypted message and the key as input and outputs the original message. We refer to the original message as the plaintext or pt and the encrypted message as the ciphertext or ct.

$$\begin{aligned} \mathsf{Gen}(1^n) &\to k \\ \mathsf{Enc}_k(pt) &\to ct \\ \mathsf{Enc}_k^{-1}(ct) &\to pt \end{aligned} \tag{1.2}$$

We are not concerned with how encryption schemes are implemented or on what problems they rely; rather, we use encryption schemes as subroutines, so we are concerned with the security guarantees that they offer.

We say that an encryption scheme is secure if an adversary cannot tell the difference between two messages. We define security using a thought experiment. In the thought experiment, the adversary has access to the encryption algorithm with key hardcoded in. This means that the adversary can encrypt any message they want, and see how the message would encrypt. The goal of the adversary at this point in the thought experiment is to find a pattern or weakness in the encryption algorithm that they can exploit. The adversary eventually picks any two messages m_0 and m_1 which they did not give to their ecnryption algorithm and see how they encrypt. The adversay shows us m_0 and m_1. We choose one of the messages[3], encrypt the message, and send the resulting ciphertext to the adversary. The adversary's goal now is to determine which message we encrypted. They still may use their encryption algorithm with the key hardcoded in. Evenutally the adversary must output either 0 or

[3]We select the message uniformly at random.

1 indicating that they think we selected m_0 or m_1 respectively. If the adversary picks correctly, then we say that the adversary wins; otherwise, we say that the adversary loses.

Finall and informally, the encryption scheme is considered secure if the probability that the adversary wins is $\frac{1}{2} + \mu(\lambda)$, i.e. the best the adversary can do is guess.

Definition 2 An encryption scheme is secure under a chosen-plaintext attack if for all probabilistic polynomial-time adversaries A, there exist a neglible function μ such that

$$Pr[E_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mu(n) \tag{1.3}$$

where E is the following experiment:

1. Generate key k by running $\mathsf{Gen}(1^n)$.

2. The adversary $\mathcal{A}$ is given 1^n and oracle access to $\mathsf{Enc}_k.$, and outputs a pair of messages m_0 and m_1 of the same length.

3. A uniform bit $b \in \{0, 1\}$ is selected, and then a ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$.

4. $\mathcal{A}$ continues to have oracle access to Enc_k, and outputs a bit b'.

5. The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise. In the former case, we say that $\mathcal{A}$ succeeds.

♢

It is useful for 2PC to create an encryption scheme that requires two keys to encryption and decrypt. An encryption scheme with two keys is called a *dual-key cipher* (DKC) Bellare et al. (2012). It is easy to instantiate a DKC if one has a secure single-key encryption scheme: let k_0 and k_1 be two keys and instantiate the

DKC as follows:

$$\begin{aligned} \mathsf{EncDKC}_{k_0,k_1}(pt) &= \mathsf{Enc}_{k_1}(\mathsf{Enc}_{k_0}(pt)) \\ \mathsf{EncDKC}^{-1}_{k_0,k_1}(ct) &= \mathsf{Enc}^{-1}_{k_0}(\mathsf{Enc}^{-1}_{k_1}(ct)) \end{aligned} \tag{1.4}$$

If the encrytion scheme used to create the DKC is secure, then it is easy to see to that the DKC is also secure. We are formally considering the statement: Enc secure $\implies$ DKC secure. Consider the contrapositive: DKC insecure $\implies$ Enc insecure. If the DKC is insecure, then an adversary can find two messages, m_0 and m_1 such that their ciphertexts are distinguishable. {AL-5: Figure this out, need to break the old intro crypto work}

1.3 Computational Indistinguishability

This section introduces the idea of computational indistinguishability. We do not use computational indistinguishability immedidiately, but it will important later for defining security of a 2PC protocol.

Informally, two probability distribution are indistinguishable if no polynomial-time algorithm can tell them apart. The thought experiment is like this: an algorithm is that there are two distributions, it is given one of the distributions. If the algorithm correctly determines which distribution it was given, then it wins, otherwise the algorithm loses. The algorithm, since it must run in polynomial time, can only sample a polynomial number of values from the distributions.

Formally, computational indistinguishability is:

Definition 3 Let $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n\in\mathbb{N}}$ be distribution ensembles. $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable, denoted $\mathcal{X} \approx_C \mathcal{Y}$, if for all probabilistic

polynomial-time algorithms D, there exists a negligible function μ such that:

$$|Pr_{x \leftarrow X_n}[D(1^n, x) = 1] - Pr_{y \leftarrow Y_n}[D(1^n, y) = 1]| < \mu(n) \tag{1.5}$$

Katz & Lindell (2007). ♢

We quickly break down the definition. The unary input 1^n tells the algorithm D to run in polynomial time in n. The probability distributions X_n and Y_n are restricted by n, which in this context is the security parameter. The phrases $x \leftarrow X_n$ and $y \leftarrow Y_n$ mean that the probability is taken over samples from the distributions.

1.4 Boolean Circuit

A function in a 2PC protocol is represented as a boolean circuit. A boolean circuit takes as input $x \in \{0,1\}^n$, performs a series of small operations on the inputs, and outputs $y \in \{0,1\}^m$. 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 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 1.4 gives the mapping of an XOR gate.

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.

A circuit is a combination of gates that are stringed together. It turns out that circuits are quite powerful: in fact, a circuit composed only of AND gates, XOR

gates and NOT gates can compute any function or algorithm **?**. In other words, if there's some algorithm that can do it, then there is some circuit that can do it as well. Figure 1.4 shows the circuit representation of the less than function, f as specified in equation **??**.

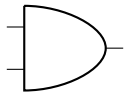

Figure 1.1: An AND gate.

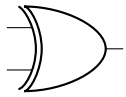

Figure 1.2: An XOR gate.

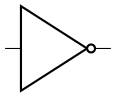

Figure 1.3: A NOT gate.

1.5 Oblivious Transfer

Oblivious Transfer (OT) is a communicating protocol that underlies many more complicated cryptosystems. At the highest level, Alice potentially sends two messages to Bob, and Bob only recieves one of the messages.

Consider the following thought experiment: Alice and Bob want to exchange a message. Alice has two messages m_0 and m_1, and she Alice wants to send one of them Bob but does not care which one. Bob knows that he wants message m_b where $b \in \{0, 1\}$. Alice gives both messages to the trusted post-office. Bob also goes to the trusted post office, and says that he wants m_b. The post office gives m_b to Bob, throws the other message (m_{1-b}) in the trash, and does not tell Alice which message

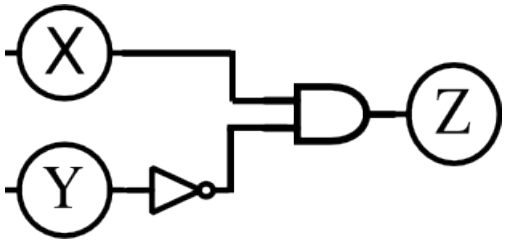

Figure 1.4: A circuit that computes the less or equal to function, equivalent to the less than function for input of two one-bit values.{AL-7: make with tikz}

x	y	$x < y$
0	0	0
0	1	1
1	0	0
1	1	0

Table 1.2: The truth table of the less than circuit.

they gave to Bob. Now, Alice has sent a message to Bob, but is oblivious as to which message was sent.

Oblivious Transfer is a useful protocol and is a fundamental component of 2PC protocols. Oblivious Transfer protocols guarantee the following security properties:

1. Alice does not know whether Bob recieved m_0 or m_1.

2. Bob does not know anything about m_{1-b}, the message that he did not receive.

Chapter 2

Classic 2PC

{AL-8: this paragraph is a little weak} Secure Computation (SC) was first proposed in an oral presesntation by Andrew Yao **?**. Since Yao's presentation, multiple methods for performing SC were developed. One method was developed by Yao himself and is called garbled circuit. Another was developed by a group of researchers, Goldreich, Micali and Widgerson {AL-9: cite}. 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. {AL-10: get source}

This chapter explains the two most prominent methods of SC for the two party case, referred to as Two-Party Protocol (2PC). The first part of the chapter will motivate and describe desirable properties of a 2PC protocol, culminating in a definition. The second part of the chapter will describe in Yao's Garbled Circuits, a method for performing 2PC, and discuss security of 2PC. The third part will provide an overview of GMW's method.

2.1 2PC Security Motivation

Think back to Alice and Bob from the introduction. Alice and Bob are millionaires who wish to determine who is wealthier without disclosing how much wealth they have. More formally, Alice has input x and Bob has input y (x and y are integers corresponding to the wealth of each party), and they wish to compute the less than function f, such that

$$f(x,y) = \begin{cases} 1 & \text{if } x < y; \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

We call the overarching interaction between Alice and Bob protocol Π, and Π 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 party during the execution of Π must be deduced from the output. For example, if Alice learns that she had more money after computing f, then she learns that $y < x$; however, this information about y is deducible from the output therefore it is reasonable. It would be unreasonable if Alice learns that $1,000,000 < y < 2,000,000$, as that information is not deducible from $f(x,y)$.

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

One possible method for constructing 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 import security property that is only relevant in certain cases may be missed. There are many applications of 2PC, and in some cases, there may be certain properties that critical to security. Ideally, a good definition of 2PC

works for all applications, hence capturing all desirable properties. 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 the definition. A definition based on properties in this respect as it becomes the burden of the prover of security to show that all relevant properties are covered Lindell & Pinkas (2009).

We must also think about the aims of each party involved in the protocol. Can we trust that parties are going to obey the protocol? It's relevant in the two party case, but if there are more than three parties, parties may either act independently or collude. These considerations are called the *security setting*. There are two primary security settings: the semi-honest setting and the malicious setting. The work presented in this thesis uses the semi-honest setting. In the semi-honest setting, we assume that each party obeys the protocol but tries to learn as much as possible from the information they are given. This means that parties do not lie about their information, they do not abort, they do not send or withhold messages out of order, or deviate in any way from what is specified in the protocol. In contrast, the malicious setting considers that each party is liable to lie and cheat; parties can take any action to learn more information.

The malicious setting is much more realistic. Parties that are involved in cryptographic protocols are liable to lie and cheat, for why else would they even be engaged in the cryptographic protocol in the first place? There are two main reasons why the semi-honest setting is useful. The first is that many protocols can be constructed for the semi-honest setting, and then improved to function in the malicious setting. There is a strong history of this occurring with protocols. It's simply easier to think through and create protocols for the semi-honest setting; at the very least, it's a valuable starting point for building complex cryptosystems. In the case of 2PC, there exist malicious protocols, and in fact, the primary 2PC protocol that this thesis uses,

garbled circuits, can be improved to be malicious secure without too much difficulty.

The second reason that the semi-honest setting is that it does have use cases in the real world. There are some scenarios where parties want to compute a function amongst themselves, and trust each other to act semi-honestly. One example is hospitals sharing medical data. Hospitals are legally, and arguably ethically, restricted from sharing medical data, but this data can have great value especially when aggregated with datasets from other hospitals. 2PC offers hospitals the means to "share" their data, perform statistics and other operations on it, while keeping the data entirely private. Other examples where semi-honesty is sufficient include mutually trusting companies and government agencies.

2.2 2PC Security Definition

In this section we discuss at a high level the definition of 2PC security, and then give the formal definition. The definition of security for 2PC protocols is the most complicated cryptographic theory that we have encountered thus far.

Recall our setup: Alice and Bob are semi-honest parties with inputs x and y respectively who wish to compute the function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m \times \{0,1\}^m$. Informally, we say Π is scure if when Π is executed in the real world, each party learns the same information as if they had completed the protocol in an ideal world. The ideal world is where there is a trustred third party Carlo who receives the inputs x from Alice and y from Bob, computes the $f(x, y)$, and sends the output to Alice and Bob. Carlo is honest and trusted, so we don't mind that he knows the inputs, and he is trusted to not share the inputs with the opposoring party. This is the ideal world: the function is compute correctly, and no information about the inputs is shared to other parties.

In the real world, Alice and Bob do not have luxury of enlisting Carlo to perform

their computation for them; instead, they use the 2PC protocol Π. To be confident that Π is secure, we show that Π in the real world is *essentially the same* as Alice, Bob and Carlo acting in the ideal world. Specifically, Alice and Bob should not learn any more information in the real world than they do in the ideal world. If they do learn more information, then somewhere the protocol Π is leaking information that cannot be deduced from the output of f. We show that the ideal is world is essentially the same as the ideal world by using the concept of computational indistinguishability presented in chapter 1. To use computational indistinguishability, we need to build probability distrubtions of the information deducible in the real and ideal world. Next, we construct probability distributions of the information that Alice and Bob can deduce from the ideal and real world.

To construct the probability distribution of the ideal world, we introduce *simulators* S_A and S_B. S_A and S_B are probabilistic polynomial-time algorithms who are essentially adversaries, like the adversary in the definition of encryption from chapter 1, that specifically attack the ideal world. Simulator S_A takes as input x, Alice's input, and $f(x, y)$, the output of the function because that is the information that Alice has access to in the ideal world. Likewise, S_B takes input y and $f(x, y)$, since that the is information that Bob has access to in the ideal world.

Given a simulator S_A, think about what the simulator can do. The distribution of the its possible outputs is given by $\{S_A(x, f(x, y)\}_{x\in\{0,1\}^*}$. Let us break this distribution down: S_A is a fixed single algorithm. x is Alice's input, and $f(x, y)$ is the output of the function. The set is indexed by all possible x, so all possible inputs that Alice could have. In sum, $\{S_A(x, f(x, y)\}_{x\in\{0,1\}^*}$ represents the possible information that an algorithm could deduced from all possible x and $f(x, y)$. We think of $\{S_B(y, f(x, y)\}_{y\in\{0,1\}^*}$ for Bob's input similarly.

For constructing the probability distributions of the real world, we need to consider what information Alice and Bob have at their disposal. Recall that Alice and Bob are

semi-honest, which means that Alice and Bob follow all instruction of Π correctly but they also will use any information they receive along the way. More precisely, Alice and Bob obey the protocol, but also maintain a record of all intermediate computations. We call Alice's record of intermediate computations Alice's *view*, $\mathsf{view}_A(x, y)$, which depends on inputs x and y And now we create the probability distribution for Alice: $\{\mathsf{view}_A(x, y)\}_{x,y\in\{0,1\}^*}\}$. This distribution represents the Alice's information from the intermediate computation indexed over all possible inputs x and y. Likewise, we call Bob's record of intermediate computations Bob's view, $\mathsf{view}_B(x, y)$, and his probability distribution of intermediate computations is $\{\mathsf{view}_B(x, y)\}_{x,y\in\{0,1\}^*}\}$

To wrap it up, if Π in the real world is the same as the Alice, Bob and Carlo in the ideal world, then the simulator S_A and S_B should only be able to learn what can be learned from the intermediate computations. That is, the probability distributions $\{S_A(x, f(x, y)\}_{x\in\{0,1\}^*}$ and $\{\mathsf{view}_A(x, y)\}_{x,y\in\{0,1\}^*}\}$ should be essentially the same, i.e., computationally indstinguishable.

With this intuition in mind, we give Goldreich's definition of 2PC security from his textbook *Foundations of Cryptography Volume II* Goldreich (1995).

Setup:

- Let $f = (f_1, f_2)$ be a probabilistic, polynomial time functionality where Alice and Bob compute $f_1, f_2 : \{0, 1\}^n \to \{0, 1\}^m$ respectively.
- 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[1] 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:

$$\{(S_1(x, f_1(x,y), f(x,y)))\}_{x,y} \equiv^C \{(\mathsf{view}_1^{\Pi}(x,y), \mathsf{output}^{\Pi}(x,y))\}_{x,y} \tag{2.2}$$

$$\{(S_2(x, f_2(x,y), f(x,y)))\}_{x,y} \equiv^C \{(\mathsf{view}_2^{\Pi}(x,y), \mathsf{output}^{\Pi}(x,y))\}_{x,y} \tag{2.3}$$

Intuition: We think of view_i^{Π} as all of the information that the ith party 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 the output of the ith party. The value of output_i^{Π} is computable from the tuple view_i^{Π}.

Equations 2.2 and 2.3 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. 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.

[1] {AL-11: TODO} Mention what static means

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 **?**.

2.3 Yao's Garbled Circuit

{AL-12: struggling a bit with the organization of this section} {AL-13: Property that if encryption fails, we know. something that easily added to any encryption scheme and not a big deal, but super useful} We now give an implementation of a generic 2PC scheme created by Yao **?**. The scheme is designed for two parties, Alice and Bob, who have inputs x and y respectively. We suppose that Alice and Bob wish to compute the function $f : \{0,1\}^n \to \{0,1\}^m$.

{AL-14: This should probably be 2-4 sentences, instead of one} At a high level, Yao's scheme has the following steps: (1) Alice builds and encrypts the circuit representing f. (2) Alice sends the garbled (encrypted) circuit to Bob. (3) Bob receives the input labels via Oblivious Transfer. (4) Bob uses the input labels to decrypt the circuit. (5) Bob sends the output of the circuit to Alice.

2.3.1 Step 0: Setup

Alice and Bob wish to compute the function f, which is represented by a circuit (see section **??**). Alice is the garbler, and will create and encrypt the circuit. Bob is the

Algorithm 1 Garble Circuit

Require: Circuit $f(x, \cdot)$
Ensure: 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**
      $T_g[u, v] = \mathsf{Enc}_{W_i^u}(\mathsf{Enc}_{W_j^v}(W_k^{g(u,v)}))$ {AL-15: use DKC}
    end for
    $f(x, \cdot).tables[g] = T_g$.
  end for

Algorithm 2 Evaluate Circuit

Require: $(input\_wires, tables, gates)$
  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** {AL-16: use DKC}
      $temp = \mathsf{Dec}_{w_j}(\mathsf{Dec}_{w_i}(tables[g][u, v]))$
      if $temp$ decrypted correctly **then**
        $w_k = temp$
      end if
    end for
  end for

evaluator, and will compute the circuit.

2.3.2 Step 1: Garbling the Circuit

{AL-17: define a wire and a label} Alice garbles each gate, according to algorithm **??**, resulting in a table T_g for each gate g. The table enables the computation of a single gate 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. {AL-18: TODO}

2.3.3 Step 2: Bob's Input and Computing the Circuit

Alice sends the garbled circuit to Bob. The garbled circuit consists of a garbled table for each gate, $f(x,\cdot).tables$, 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 2.

2.3.4 Explanation of the security of Yao's protocol

To do.

2.3.5 Notes about complexity

1 OT per (input?) wire. 4 cts per gate. How much encryption? Not good enough for practice.

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 general, is a 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 3 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 4. 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.

{AL-19: see mike rosulek first few slides for good images. } {AL-20: illustration about translating bits over to wire labels}

Algorithm 3 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 4 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. $b'_{w_i} = \neg b_{w_i}$

Chapter 3

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.

3.1 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)	$\mathsf{Enc}_{A_0,B_1}(C_1)$
(0,1)	$\mathsf{Enc}_{A_0,B_0}(C_0)$
(1,0)	$\mathsf{Enc}_{A_1,B_1}(C_0)$
(1,1)	$\mathsf{Enc}_{A_1,B_0}(C_0)$

$C_0 \leftarrow \{0,1\}^n$
$C_1 \leftarrow \{0,1\}^n$

Table 3.1: Garbled Gate for Point and Permute

Table 3.2: 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

3.2 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)	$\mathsf{Enc}_{A_0,B_0}(C_0)$
(1,0)	$\mathsf{Enc}_{A_1,B_1}(C_0)$
(1,1)	$\mathsf{Enc}_{A_1,B_0}(C_0)$

$C_0 \leftarrow \{0,1\}^n$
$C_1 \leftarrow \mathsf{Enc}^{-1}_{A_0,B_1}(0^n)$

Table 3.3: 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.

3.3 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 3.4: 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

3.4 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

3.5 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.

3.6 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 3.6 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
$H(B) \oplus C$	$H(B) \oplus C$	$\rightarrow$	$H(B) \oplus C$
$H(B) \oplus C$	$H(B) \oplus C \oplus \Delta$		$H(B) \oplus C \oplus a\Delta$

Table 3.5: 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 3.6 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 3.6: 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:

$$a \wedge b = a \wedge (r \oplus r \oplus b) = (a \wedge r) \oplus (a \wedge (r \oplus b)). \tag{3.1}$$

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

3.7 Improving Oblivious Transfer

{AL-21: Fix this up} Oblivious transfer has been improved in two relevant ways for 2PC protocols. These improvements are called OT-extension and OT-preprocessing. In OT-extension a constant number of OT's can be run to generate a polynomial number of exchanged values. In OT-preprocessing the OT's occur before the actual protocol, during what is called the offline phase, and then when the OT's needed during the online phase, simpler and efficient communication realizes the exchanged values. This is useful because the OT step requires a large amount of communication to be sent between the sender and receiver, often resulting in OT being the bottleneck of 2PC protocols.

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

Chapter 4

Chaining Garbled Circuits

the "theory"

talk about our idea about chaining here.

4.1 The Random Oracle Model and Random Permutation Model

4.2 Chaining

4.3 Security of Chaining

4.4 Single Communication Multiple Connections

4.5 Security of SCMC

Chapter 5

Implementation

talk about my implementation here. be clear about what I did.

5.1 JustGarble

5.2 Our Implementation: CompGC

5.3 Adding SCMC

Chapter 6

Experiments and Results

talk about experiments and results here

6.1 Experimental Setup

talk about what computer was used, bandwidth, network simulation

6.2 Experiments

AES, CBC, Levenshtein

6.3 Results

Chapter 7

Future and Related Work

talk about future and related work here

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.

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.

Boneh, D. (1998). *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, chap. The Decision Diffie-Hellman problem, (pp. 48–63). Berlin, Heidelberg: Springer Berlin Heidelberg. http://dx.doi.org/10.1007/BFb0054851

Goldreich, O. (1995). Foundations of cryptography.

Goldreich, O., Micali, S., & Wigderson, A. (1987). How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, (pp. 218–229). New York, NY, USA: ACM. http://doi.acm.org/10.1145/28395.28420

Katz, J., & Lindell, Y. (2007). *Introduction to Modern Cryptography: Principles and Protocols (Chapman & Hall/CRC Cryptography and Network Security Series)*. Chapman and Hall/CRC. http://www.amazon.com/Introduction-Modern-

Cryptography-Principles-Protocols/dp/1584885513%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1584885513

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.