Scalable Secure Computation

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFIMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

Faculty of Engineering

BY

Dheeraj Ram

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2016

Declaration of Originality

I, **Dheeraj Ram**, with SR No. **04-04-00-10-41-14-1-11168** hereby declare that the material presented in the thesis titled

Scalable Secure Computation

represents original work carried out by me in the **Deparment of Computer Science and Automation** at **Indian Institute of Science** during the years **2014-2016**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discusions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

My parents

who are my true inspiration

Acknowledgements

I express my sincere gratitude to my project advisor *Dr. Arpita Patra* for the guidance and motivation during this project. I also thank my labmates and friends who involved in the active discussions about my work.

Abstract

Secure Multi-Party Computation (MPC) is the area of computing a function on the inputs of several parties, without compromising the secrecy of their individual inputs. The area has been explored extensively starting with pioneering work of Yao (FOCS '82, FOCS '86) for two parties, from late 80's. Yao has shown how to perform secure MPC on any generic function with a *constant round* protocol, by *garbling* the logical circuit of the function. Constant round secure computation protocols are of great interest as they lead to protocols with low latency. Later Beaver, Micali and Rogaway, (henceforth known as BMR protocol) (STOC '90) extended the Yao's protocol to n parties, where $n > 2$ retaining the constant round complexity. There are several optimizations proposed on garbled circuits such as Free-XOR technique, garbled row reduction technique that lead to a number of variants of Yao protocol that are more efficient than the original Yao construction. However, the scope of these techniques are limited to the two-party case. Since a real-life MPC application may involve more than two parties and the circuit may contain millions of gates, it is very important to scale the optimizations on garbled circuits for the n-party case where $n > 2$. In this work we introduce couple of optimizations called *cheap-XOR* and *Almost Free-XOR* on BMR protocol, which functionally resembles FreeXOR and an XOR optimization from GLNP15[8] which are applicable in only Yao's protocol and allows to evaluate XOR gates *free of cost*. We also optimized NOT gate so that no ciphertext is needed for NOT gate evaluation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Multi-Party Computation (MPC) is the area where a set of n mutually distrustful parties can securely compute a functionality on the inputs provided by those parties. An MPC protocol must guarantee certain properties such as *correctness* (which ensures the protocol outputs same as what the function would output from the inputs), *privacy* (each party should learn only the output) and the *independence of input* (ensuring each party choosing their input independent of the others). Formally, the security proof of an MPC protocol is done in *Real-world Ideal-world paradigm*. This paradigm ensures that the output distribution of a real world execution of the proposed protocol is same as that of ideal world execution, where there is a trusted third party who will execute the function using the inputs given by the parties. The distrust between the parties can be represented by an entity called *Adversary* who may corrupt t out of n parties. We consider dishonest majority setting, that is $t < n$. There can be two different setting for an MPC protocol by modelling the adversary as either *semi-honest* who is curious, but follows the protocol or *malicious* who may even deviate from protocol.

1.2 Background

For a generic function to be executed securely, there are two approaches that have been followed; the first one uses Yao's garbled circuits [16] and the second utilizes the GMW protocol [7]. Both protocols require the given function to be represented by a Boolean circuit and then evaluated "securely" revealing nothing but the circuit output. While GMW demands interaction for each AND gate of the circuit causing an overall round complexity in the order of the multiplicative depth of the circuit, Yao's protocol runs in constant (2) rounds. Even though the garbled circuit construction is intricate and involved, the property of constant round results in faster

execution of protocol.

In Yao's protocol, the responsibilities of the two parties are not alike; one party acts as *garbled circuit constructor* and the other one as *garbled circuit evaluator*. The circuit is represented as a set of wires and Boolean gates. To garble a circuit, every wire will be associated with two indistinguishable keys (namely k_w^0 and k_w^1 for wire w), representing bits 0 and 1 which are chosen by the constructor. The idea of Yao's protocol is to let the evaluator evaluate the circuit with precisely one key per wire and without knowing the meaning of the keys except the circuit-output wires. To accomplish this, all the two-input *garbled gates* are represented as a set of four ciphertexts, which can be decrypted with the key pair from the input wires. The message encrypted will be the corresponding key of the output wire, depending on the gate and input keys (For eg: if a, b are input wires and c is the output wire of a gate g, $k_c^{g(\alpha,\beta)}$ is encrypted using k_a^α and k_b^β). All the garbled gates of a circuit put together defines a garbled circuit. Yao's protocol make sure that evaluator will obtain all the keys of input wires, without revealing the private inputs of parties to each other. After this, evaluator on receiving the garbled circuit from the constructor can decrypt exactly one ciphertext out of four, revealing precisely one key on each wire. In the end, evaluator will be left with the key of the output wire. Garbled circuit constructor needs to provide the output wire keys to their bit mappings along with the circuit so that, evaluator will get to know the output of the MPC function. Yao's protocol takes two rounds in semi-honest setting. The computational and communication efficiency of Yao's protocol depends on the efficiency involved in constructing and evaluating an garbled circuit for a given circuit and respectively on the size of the garbled circuit.

Garbled circuits, apart from having application in MPC, find application in verifiable or delegatable computation [6], Zero-Knowledge Proofs [5] and therefore is of immense interest in cryptography community as an independent area of research [2, 3, 4, 13]. There has been constant interest in optimizing the size of garbled circuits and in improving the construction and evaluation of the same. There are several optimization techniques for Yao's protocol, such as Free-XOR [9], Garbled Row Reduction (GRR) [14], FleXOR [10], Garbled XOR With a Single Ciphertext [8] (referred to as *Almost Free-XOR* henceforth), etc. Both the GRR and Free-XOR techniques reduce the size of garbled circuit immensely and bring down the computational cost of garbled circuit construction and evaluation. In fact, the Free-XOR technique enables the XOR gates to be garbled and evaluated at *free of cost*. This is accomplished by maintaining a constant distance between the two keys of any wire ($k_w^0 \oplus k_w^1 = R$ for any wire w). This enables the XOR of any two input key pair to be one of two possible values; if a and b are input

wires, the result of XOR will be either $k_a^0 \oplus k_b^0$ or $k_a^0 \oplus k_b^0 \oplus R$. These two will be chosen as the k_c^0 and k_c^1 respectively where c is the output wire, and thereby enabling XOR gate evaluation as XOR of the input keys. FleXOR [10], introduced by Kolesnikov et al., is a similar concept to Free-XOR. Almost-Free-XOR [8] optimization questions the necessity to use non-standard assumptions (such as Random Oracle model; refer to Section 2) in Free-XOR technique, by introducing an optimization which is close to Free-XOR in efficiency and using only standard assumptions.

The BMR protocol [1] is a variant of Yao's protocol that runs in a multi-party setting with more than two parties. In this protocol, all parties collaboratively garble the circuit in offline phase, where parties need to know only the Boolean circuit corresponding to the functionality. This offline phase is followed by an online phase where, parties will start evaluating the circuit as a local computation using the garbled circuit from the offline phase along with their inputs. However, in the case of malicious adversaries this protocol suffers from two main drawbacks: (1) Security is only guaranteed if at most a minority of the parties are corrupt i.e. $t < n/2$; (2) The protocol uses generic protocols secure against malicious adversaries (say, the GMW protocol [7]) that evaluate the pseudo-random generator used in the BMR protocol. This non black-box construction results in an extremely high overhead. Recently, [12] extends BMR protocol in dishonest majority setting. However, thus far none of the optimizations techniques for Yao's garbled circuit are known to apply in BMR protocol. Specifically, the scope of Free-XOR, GRR and Almost Free-XOR are limited to the two-party case. Since a real-life MPC application may involve more than two parties and the circuit may contain millions of gates, it is very important to explore whether scaling of the optimizations on garbled circuits for the n-party case where $n > 2$ is feasible or not.

1.3 Our Contribution and Future Work

Our goal is to construct scalable MPC protocols. We would like to see if the optimizations of Yao's protocol resulted from techniques like Free-XOR, GRR etc, can be applied in multi-party scenario. Towards that, we introduce a new optimization to BMR called *Cheap-XOR*, which functionally resembles Free-XOR. That is, our new method enables us to garble and evaluate the XOR gates for free. We have defined a generic garbling scheme where the multi-party protocols can fit into, and prove properties like privacy with ease. We also extended the optimization Almost Free-XOR to BMR protocol. Although Almost Free-XOR demands one ciphertext for XOR gates unlike Free-XOR, this optimization is under standard assumption where Free-XOR isn't. Finally we also show an optimization on NOT gates, where no ciphertext is needed.

We leave checking the scalability and relevance of techniques like GRR [14], FleXOR [10], etc. in multi-party case as a future work.

1.4 Roadmap

In this Report, first we will explore Yao's protocol (Section 2.1), followed by BMR, Free-XOR and Almost Free-XOR (Section 2.2, 2.3 and 2.4), discussing circuit construction and evaluation in detail. Then we will look into Cheap-XOR circuit construction and evaluation (Section 3.1). We will conclude after giving an intuitive level proof of security for Cheap-XOR (Section 3.1.4). We also defines a generic distributed garbling scheme (in Section 3.2) and show that BMR fits into definition (in Section 3.3). We show our optimizations Almost Free-XOR (in Section 3.4)and NOT gate with no ciphertext (in Section 3.5) in BMR and prove the security (in Section 3.4.4).

Chapter 2

Preliminaries

Before we discuss our contribution, we elaborate in detail on the required techniques and protocols, namely, Yao's protocol [16], BMR protocol [1], Free-XOR [9] and Almost Free-XOR [8].

2.1 Yao's protocol

Yao introduced an idea called *garbled circuits* in oral presentations about Secure Function Evaluation (SFE) [16, 15]. This idea lead to the first constant round MPC protocol known as *Yao's protocol*. The first written account of the protocol is by Goldreich, Micali, and Wigderson [7]. We briefly discuss Yao's protocol based on garbled circuits. In this protocol, the desired function f is represented by a Boolean circuit C that is computed gate by gate from the input wires to the output wires. We distinguish four different types of wires used in a given Boolean circuit: (**a**) circuit-input wires; (**b**) circuit-output wires; (**c**) gate-input wires (that enter some gate g); and (**d**) gate-output wires (that leave some gate g). We first discuss how to garble a circuit and the underlying ideas. The first step to garble a circuit is to associate every wire w with two random values, say k_w^0, k_w^1, such that k_w^0 represents the bit 0 and k_w^1 represents the bit 1. Then the garbled table for each gate, also referred as a garbled gate, maps random input values to random output values, with the property that given two input values it is only possible to learn the output value that corresponds to the output bit. This is accomplished by viewing the four potential inputs to a gate, k_a^0, k_a^1 (values associated with the first input wire, denoted as a) and k_b^0, k_b^1 (values associated with the second input wire, denoted as b), as encryption keys. So that the output key values k_c^0, k_c^1 are encrypted under the appropriate input keys. More details on how to construct a garbled gate follow.

Let us denote the considered logic gate as g, the input wires as a and b and output wire

as c. k_a^0 and k_a^1 be the keys of wire a corresponding to bits 0 and 1 respectively, and similar notation for wires b and c. We have four combinations of key pair $\langle k_a^\alpha, k_b^\beta \rangle$. We will encrypt four messages using these four key pairs. The message which is encrypted each time will be one of the keys of wire c (k_c^0 or k_c^1). Which one to encrypt depends on the key pair and g. i.e, with key pair $\langle k_a^\alpha, k_b^\beta \rangle$, we will be encrypting the key k_c^σ where $\sigma = g(\alpha, \beta)$ and $\alpha, \beta, \sigma \in \{0, 1\}$. We can visualize these ciphertexts as boxes which needs two keys to open.

It is obvious to see how to create a garbled table for NOT gate; we will possess only two cipher-texts which are encryption of k_c^0 using k_a^1 and encryption of k_c^1 using k_a^0. For an AND gate g, k_c^0 (that corresponds to bit 0) is encrypted under the pair of keys associated with the values $(0, 0)$, $(0, 1)$, $(1, 0)$. Whereas, k_c^1 is encrypted under the pair of keys associated with $(1, 1)$ which yields the following four ciphertexts

$$\mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^0)), \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^0)),$$
$$\mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^0)) \text{ and } \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^1))$$

where $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private key encryption scheme that has *chosen double encryption security*; see [11] for the formal definitions. The idea of garbling an AND gate is pictorially given in Figure 2.1.

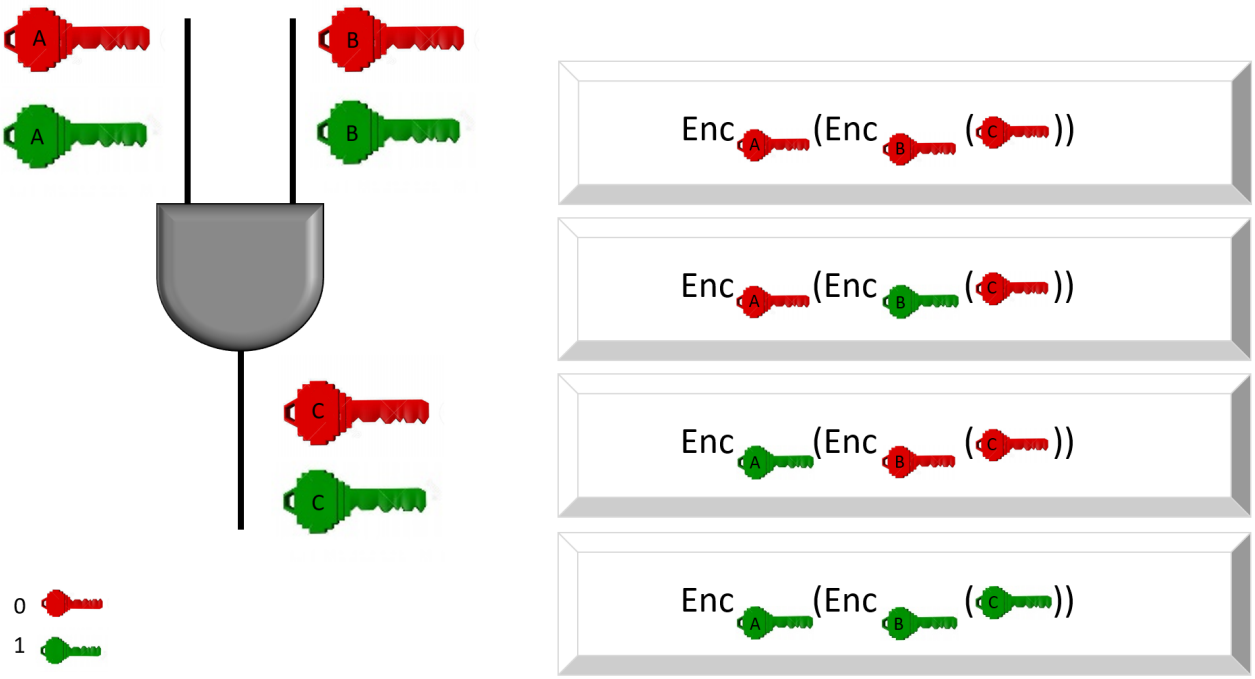

Figure 2.1: Garbled AND gate in Yao's circuit

It is also important to randomly permute these four cipher-texts to lose the meaning of mapping from key-pair to their values. Then, given the input wire keys $k_a^{\alpha}, k_b^{\beta}$ that correspond to the bits α and β and the garbled table containing the four encryptions, it is possible to obtain the output wire key $k_c^{g(\alpha,\beta)}$. The other gates are garbled similarly. The description of the garbled circuit is concluded with the *output decryption tables*, mapping the random values on the circuit output wires back to their corresponding Boolean values. In summary, a garbled circuit GC for a C includes: (a) All garbled gates (four or two ciphertexts per gate constructed as discussed above) and (b) output decryption tables for the circuit-output wires.

Now we discuss Yao's protocol that relies on the circuit garbling technique discussed above. In this protocol, there are two parties among which one party act as the garbled circuit constructor, and the other one as garbled circuit evaluator. Without loss of generality, we can call the circuit constructor as P_0 and evaluator as P_1. Once the given circuit C is garbled, P_0 provides P_1 the following to evaluate the circuit:

1. The garbled circuit GC for C.

2. Exactly one key corresponding to the input bit, for all input wires of P_0.

3. A mechanism to know the key corresponding to input bit of P_1 in each of his input wires.

It is not possible to provide both the keys for each of P_1's input wires; that may help a corrupted evaluator to execute multiple evaluations of same circuit with fixed input from P_0, and that may lead to gain some information about P_0's input[1]. This problem of exchanging exactly 1 key out of 2 keys, with the choice of receiver can be done using Oblivious Transfer (OT is an MPC primitive where one party called *sender* inputs two messages m_0, m_1 and another party *receiver* inputs a selection bit b. The OT execution will result in receiver's knowledge of m_b without revealing $m_{\bar{b}}$ to the receiver or b to the sender). If there are k number of input wires for P_1, we need to run k OTs with P_1 providing input bit as choice bit, and P_0 providing keys for 0 and 1 as it's messages, as in figure 2.2.

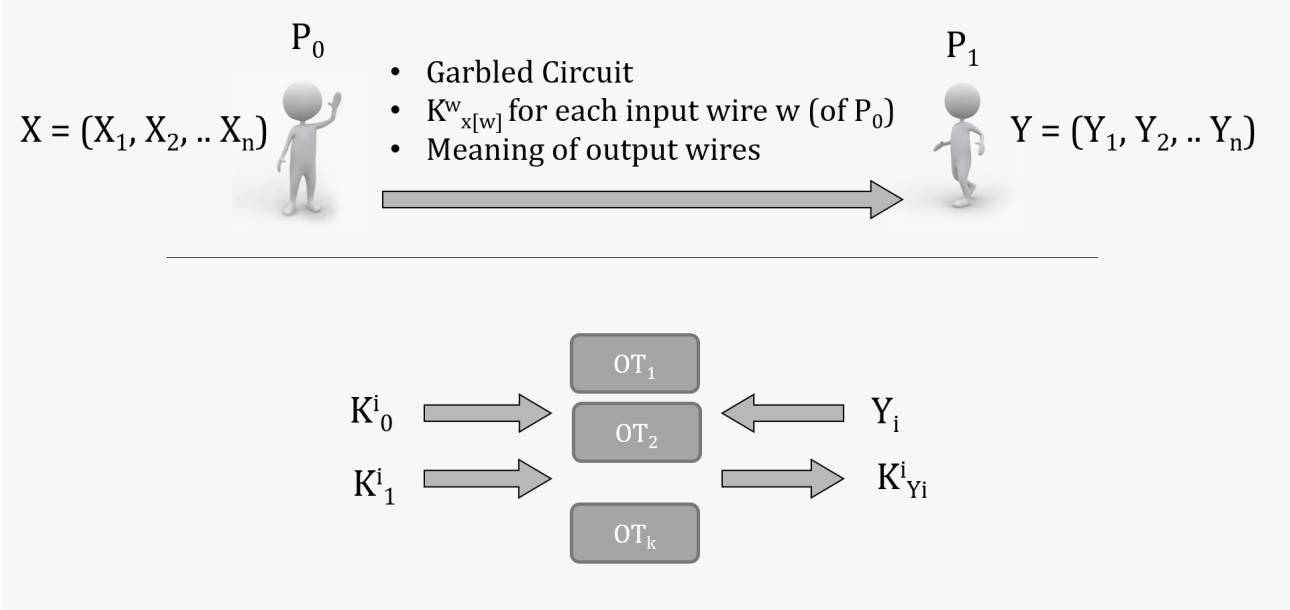

Figure 2.2: Yao's protocol

Upon receiving P_0's circuit with all the input keys, the circuit-evaluation is just local computation for P_1. After the circuit evaluation, P_1 will be left with the keys in the output wires. P_1 decrypts the output bits with the decryption table provided for output wires. It is optional (rather, depending on the specification of the problem), P_1 sends the output to P_0. The security

[1]For example, if the function is to find maximum bidder among P_0 and P_1, and P_1 can run circuit many times with same X but different Y, then P_1 can perform a binary search circuit evaluation to find out what is X.

lies in the security of OT as well as the indistinguishability of keys. Yao's protocol is proven secure in semi-honest setting.

2.2 BMR Protocol

In 1990, Beaver et al. [1] extended the Yao's protocol for generic MPC function evaluation, to n parties. Unlike Yao's protocol, BMR demands the construction of the garbled circuit *collaboratively*. Once constructed, a garbled circuit can be evaluated locally by the parties. The garbled circuit construction is done in an *offline phase* also called as *setup phase*, where the parties may communicate with the apriori knowledge of the logic circuit, but not the inputs. This phase will be followed by an *online phase* where the garbled circuit from offline phase is used to compute the underlying Boolean circuit. This offline-online paradigm is a very useful and common approach to shift the burden of heavy computations to the time where parties may be present, but they may not have decided their inputs.

2.2.1 Offline Phase of BMR protocol

In the offline phase of BMR, all the parties will collaboratively garble the circuit as follows. Now, each party will choose two random keys of length κ, for each wire. Without loss of generality, we name those keys $k^0_{w,i}$ and $k^1_{w,i}$ for each party P_i, for each wire w. Now we define *super-keys* for a wire w, represented as K^0_w and K^1_w are defined as follows:

$$K^0_w = k^0_{w,1}||k^0_{w,2}||\cdots||k^0_{w,i}||\cdots||k^0_{w,n}$$
$$K^1_w = k^1_{w,1}||k^1_{w,2}||\cdots||k^1_{w,i}||\cdots||k^1_{w,n}$$

These two super-keys will act as two different keys, and bear the meaning of the bits; just like two keys for each wire in Yao's protocol. But here, the super-key K^b_w will not represent the bit b. If it does so then, during the garbled circuit evaluation, a corrupted party will get to know the bits associated with every wire in the circuit via the associated super-key and the key it has contributed to the super-key. Namely, if the super-key for a wire contains $k^0_{w,i}$, then P_i knows that the super-key represents 0 and therefore the wire is assigned the bit 0. This will reveal the bits assigned to the intermediate wires. This is prevented by introducing an additional random bit to each wire which determine the meaning of the super-keys and which remain unknown to all the parties. Specifically, K^0_w represent the bit π_w and K^1_w represents the bit $\overline{\pi_w}$ where π_w is a random bit associated with wire w and is unknown to the adversary. These π bits are generated as follows. For each wire w, all parties will collectively decide upon a random bit π_w, which is (n, n) shared among n parties (only with the shares of all parties,

anybody can reconstruct the secret)[1]. For output wires, all parties collaboratively decide upon π values, which are random but, known to everybody. The next step is to reconstruct π_w to party P_i if w is an input wire of P_i. So every party will know the π values of their input wires, secret shares of π_w for any intermediate wire w, and π_z for output wire z.

Now we discuss how a gate is garbled in BMR protocol. Like Yao protocol, every two input gate will be associated with four ciphertexts. We need two PRGs G^1 and G^2, both maps $\{0,1\}^k \to \{0,1\}^{nk}$. Let a and b be the input wires of the gate g, and c be output wire of g. The ciphertexts corresponding to a gate are constructed as given below:

$$\begin{aligned} C_1 := &\left(G^1(k^0_{a,1}) \oplus G^1(k^0_{a,2}) \oplus \cdots \oplus G^1(k^0_{a,n})\right) \\ &\oplus \left(G^1(k^0_{b,1}) \oplus G^1(k^0_{b,2}) \oplus \cdots \oplus G^1(k^0_{b,n})\right) \oplus K^{r_1}_c \end{aligned}$$

$$\begin{aligned} C_2 := &\left(G^2(k^0_{a,1}) \oplus G^2(k^0_{a,2}) \oplus \cdots \oplus G^2(k^0_{a,n})\right) \\ &\oplus \left(G^1(k^1_{b,1}) \oplus G^1(k^1_{b,2}) \oplus \cdots \oplus G^1(k^1_{b,n})\right) \oplus K^{r_2}_c \end{aligned}$$

$$\begin{aligned} C_3 := &\left(G^1(k^1_{a,1}) \oplus G^1(k^1_{a,2}) \oplus \cdots \oplus G^1(k^1_{a,n})\right) \\ &\oplus \left(G^2(k^0_{b,1}) \oplus G^2(k^0_{b,2}) \oplus \cdots \oplus G^2(k^0_{b,n})\right) \oplus K^{r_3}_c \end{aligned}$$

$$\begin{aligned} C_4 := &\left(G^2(k^1_{a,1}) \oplus G^2(k^1_{a,2}) \oplus \cdots \oplus G^2(k^1_{a,n})\right) \\ &\oplus \left(G^2(k^1_{b,1}) \oplus G^2(k^1_{b,2}) \oplus \cdots \oplus G^2(k^1_{b,n})\right) \oplus K^{r_4}_c \end{aligned}$$

Where,

$$\begin{aligned} r_1 &= g(\pi_a, \pi_b) \oplus \pi_c \\ r_2 &= g(\pi_a, \overline{\pi_b}) \oplus \pi_c \\ r_3 &= g(\overline{\pi_a}, \pi_b) \oplus \pi_c \\ r_4 &= g(\overline{\pi_a}, \overline{\pi_b}) \oplus \pi_c \end{aligned}$$

Intuitively, the r_i values are balancing bits, to maintain relation $(0 \to \pi_c, 1 \to \overline{\pi_c})$. We can observe that $r_i = \pi_c$ whenever the gate outputs 0, and $r_i = \overline{\pi_c}$ otherwise. So the relation

[1] A simple way to achieve (n,n)-sharing is to split the secret into n random shares subject to the condition that they sum up to the secret and distribute the ith share to party P_i

holds good if the output of g is computed upon the actual bit values of the input wires. In the correctness proof of BMR, we will show that we always computes the boolean gate g upon the actual bits of input wires.

The ciphertexts C_1, C_2, C_3 and C_4 should be a computed only via a generic MPC protocol. We emphasize that, if each party just sends across the PRG outputs of both keys to all the other parties for the computation of these 4 ciphertexts, then the $\mathcal{A}$ who might have corrupted t parties, can gain extra information about the messages in the ciphertexts, that are not supposed to open in a particular execution (For example: if $t = n - 1$, and without loss of generality P_1 is the only honest party, then $G^1(k^0_{a,1})), G^2(k^0_{a,1})), G^1(k^1_{a,1})), G^2(k^1_{a,1}))$ are leaked to $\mathcal{A}$, and he can decrypt all the ciphertexts). If that is the case, $\mathcal{A}$, who does not know the key(s) of honest party, will see the ciphertexts as one-time padded messages with PRG outputs. In the end of Offline phase, every party P_i knows,

- 2 keys of each wires
- 4 ciphertexts for each gate g
- Secret share of π_j, for all wires j
- π_z, for all output wire z
- π_m, for each input wire m, if m belongs to P_i

2.2.2 Online Phase of BMR protocol

Each party will evaluate the circuit as a local computation, in this phase. Parties need to know about their inputs, only at the beginning of this phase. So we start this phase with the assumption that every party has fixed their inputs. We define a new bit called *signal bit* of a wire w denoted by λ_w, as follows:

$$\lambda_w = v_w \oplus \pi_w \tag{2.1}$$

Where v_w is the actual bit in a wire, in a specific evaluation. The λ_w value of the input wire w of P_i can be calculated by the party P_i only; because only P_i knows π_w. But the circuit has been constructed in a way that, everybody needs λ to evaluate the circuit. So the owner of the input wire(s) will broadcast the λ values of his input wires. After every party repeats this step, all parties will get to know the λ values of all the input wires in the circuit. For decrypting one logic gate, we need exactly one super-key for each input wire. So every party will send across

λ_a	λ_b	Decryptable Ciphertext
0	0	C_1
0	1	C_2
1	0	C_3
1	1	C_4

their keys $k_{w,i}^{\lambda_w}$ so that every party will get exactly one super-key corresponding to λ_w for each input wire w. With this information, parties can decrypt exactly one ciphertext as follows:

Using the super-keys $K_a^{\lambda_a}$ and $K_b^{\lambda_b}$, all parties can decrypt the ciphertext and get the super-key of the output wire, and so on. But its not clear how to get λ value of the new (gate output) wire. But every party knows their keys on all the wires, and super-key is constructed by appending all the keys of parties. So every party can locally check the key bits corresponding to that party ($(i-1)\cdot k+1$ to $i\cdot k$ for party P_i) in the super-key. If that happened to be equal to $k_{c,i}^0$ then the λ_c is 0; Otherwise it will be 1.

In short, after computing and sharing λ_w values and super-keys $K_w^{\lambda_w}$ of all input wires w, no other communication between the parties are required. The entire circuit evaluation is done as local computation. Note that, λ values are found and propagated from input wires to output wires. As far as π values are chosen random, by disclosing λ values, it guarantees perfect security to the v value.

2.2.3 Correctness of BMR

Consider the case when $\lambda_a = \lambda_b = 0$. We have to prove that λ_c and $K_c^{\lambda_c}$ are correctly computed, so that they satisfy equation 2.1. When both λ_a and λ_b are 0, we can prove the correctness as follows:

$$\lambda_a = 0 \implies v_a = \pi_a$$
$$\lambda_b = 0 \implies v_b = \pi_b$$

As per the table of decryptable ciphertexts, we will decrypt C_1.

- If $g(\pi_a, \pi_b) = \pi_c$, that means $g(v_a, v_b) = \pi_c$ and we will be encrypting $S_{c,0}$ as message, in C_1 (see the construction). We can see that λ_c must be 0, to become consistent with the definition of λ. In this case,

$$g(v_a, v_b) = \pi_c \tag{2.2}$$

$$g(v_a, v_b) = v_c \tag{2.3}$$

$$\begin{aligned} (2.2)\&(2.3) &\implies \pi_c = v_c \\ &\implies \lambda_c = \pi_c \oplus v_c \\ &\implies \lambda_c = 0 \end{aligned}$$

- If $g(\pi_a, \pi_b) = \overline{\pi_c}$, then $g(v_a, v_b) = \overline{\pi_c}$ and C_1 will be an encryption of K_c^1. Here, λ_c must be 1, to become consistent with the definition of λ. In this case,

$$g(v_a, v_b) = \overline{\pi_c} \tag{2.4}$$

$$g(v_a, v_b) = v_c \tag{2.5}$$

$$\begin{aligned} (2.4)\&(2.5) &\implies \pi_c = \overline{v_c} \\ &\implies \lambda_c = \pi_c \oplus v_c \\ &\implies \lambda_c = 1 \end{aligned}$$

When both λ_a and λ_b are 0, super-key and λ_c calculated are shown to be correct. All the other ciphertexts, we can prove with similar argument.

At the end of circuit evaluation, all parties will get to know the λ values of output wires. Note that, Parties already know the π values of output wires. Hence we can compute v_c using the equation (2.1) for each output wire c.

2.3 Free-XOR: An Optimization for Yao's protocol

In 2008, Kolesnikov et al. introduced an optimization to Yao's garbled circuit [9] which makes the evaluation of XOR gates, *free of cost*. Their approach relies upon Random Oracle (RO) assumption. Free-XOR is proven secure under semi-honest setting. Let's first look at how the Yao's circuit is modified here.

2.3.1 Free-XOR circuit construction

Party P_0 (who constructs circuit) will first sort the gates of the circuit topologically, and then chooses keys for garbled circuit. The key selection mechanism in Free-XOR is as follows: P_0 will selects a random string R, only once for the circuit. If a wire w is *not* the output of an XOR gate, then the key k_w^0 is randomly chosen, and $k_w^1 := k_w^0 \oplus R$. On the other hand, if wire c is the output wire of an XOR gate of input wires, say a and b, then $k_c^0 := k_a^0 \oplus k_b^0$ and $k_c^1 := k_c^0 \oplus R$.

2.3.2 Garbling of XOR gates

In this construction, we don't need ciphertexts for XOR gate. P_1 can get the output key by performing XOR upon input keys of XOR gate. We are about to show correctness of this approach with respect to XOR gate. Assume a, b are input wires and c is the output wire. We know that the following equations hold for the keys of a, b and c.

$$k_c^0 = k_a^0 \oplus k_b^0 \tag{2.6}$$

$$\begin{aligned} k_c^1 &= k_c^0 \oplus R && (2.7)\\ &= k_a^0 \oplus k_b^0 \oplus R && (2.8) \end{aligned}$$

Let B_a and B_b be the bit values on the wires a and b respectively. Then the keys that P_1 (evaluator) possess will be $k_a^{B_a}$ and $k_b^{B_b}$. According to the Free-XOR protocol, we will get the key for wire c as $k_a^{B_a} \oplus k_b^{B_b}$. We now need to prove this expression evaluates to $k_c^{B_a \oplus B_b}$. We can observe that,

$$k_w^{B_w} = k_w^0 \oplus (B_w \cdot R) \tag{2.9}$$

for any wire w. Therefore,

$$\begin{aligned} k_a^{B_a} \oplus k_b^{B_b} &= \big(k_a^0 \oplus (B_a \cdot R)\big) \oplus \big(k_b^0 \oplus (B_b \cdot R)\big) \\ &= k_a^0 \oplus k_b^0 \oplus R \cdot (B_a \oplus B_b) \\ &= k_c^0 \oplus R \cdot (B_a \oplus B_b) \\ &= k_c^{B_a \oplus B_b} && \text{from (2.9)} \end{aligned}$$

2.3.3 Garbling of non-XOR gates

Note that in Yao's protocol both the keys corresponding to a wire are chosen uniformly at random and independent of each other. Whereas Free-XOR technique demands that the keys are related as discussed above. To negate the effect of the dependency, garbling of other gates are done by masking the output wire key by the output of Random Oracle, parameterized with input keys. Recall that RO implements a function that is chosen uniformly at random from the set of functions (for a given domain and co-domain) and therefore it's output distribution is uniform over the co-domain.

For garbling non-XOR gates, we perform the following steps. First choose the key corresponding to bit 0 in output wire (k_c^0), randomly and we set $k_c^1 := k_c^0 \oplus R$. Using the keys of input wires a and b, we should be able to decrypt $k_c^{g(v_a,v_b)}$ for gate g (which will be an integer representing topological position of the gate). In the construction of ciphertexts, we use a *Random Oracle* $H : \{0,1\}^* \to \{0,1\}^l$ if $l = |k_w^b|$ for all wire w and bit b.

$$C_{v_a||v_b} := H(k_a^{v_a}||k_b^{v_b}||g) \oplus k_c^{g(v_a,v_b)} \tag{2.10}$$

In Yao's protocol, they had an optimization technique called *point and permute* to limit the attempts to decrypt the ciphertext from 4 to 1. Point and permute technique will assign a random bit to one key, and the complement bit to the other key, on a wire. The permutation bit is disclosed along with the key, so that $\langle p_a^{v_a}, p_b^{v_b} \rangle$ will be the position of ciphertext corresponding to keys $k_a^{v_a}$ and $k_b^{v_b}$ (p_w^β is the permutation bit for k_w^β, and $p_w^\beta = (p_w^{\beta^c})^c$). Point and permute is a necessity here, as the encryption is just XOR operation[1]. Here we have random permutation bits for keys that are chosen at random, and permutation bit for the output key from an XOR gate will be $p_a^{v_a} \oplus p_b^{v_b}$, a and b being input wires. Therefore, the equation (2.10) will become,

$$C_{p_a^{v_a}||p_b^{v_b}} := H(k_a^{v_a}||k_b^{v_b}||g) \oplus (k_c^{g(v_a,v_b)}||p_c^{g(v_a,v_b)}) \tag{2.11}$$

and the Random Oracle must be modified as $H : \{0,1\}^* \to \{0,1\}^{l+1}$ The circuit is constructed and given to the party P_1 who evaluates the circuit. The keys corresponding to P_0 is given along with the circuit, and keys for P_1 is transfered through OT similar to figure 2.2

Free-XOR circuit evaluation. P_1, with exactly one key for each input wire, can evaluate the circuit till the output wires. In Yao's protocol, we used to give a decryption table which maps $key \to bit$ which can't be done here. If so, then P_1 can be a passive adversary and he

[1]We can't assume the failed decryption attempts will results in $\perp$

can learn about R by performing XOR on the two keys of output wire. Even if we give just one key to bit mapping (say, always $k_z^0 \to 0$. If key doesn't match then bit is 1), still it may compromise the privacy of P_0 if the circuit outputs k_z^1, as he can still perform XOR with the key from circuit with key from decryption table. So it is essential to encrypt the output bit, using the keys.

$$C_{p_z^{v_z}} := H(k_z^{v_z} || \text{“}out\text{”} || g) \oplus v_z \tag{2.12}$$

This gives an additional couple of ciphertexts for every output gate. By decrypting with the only key, he gains no information about the other key, and hence R will not be revealed.

2.4 Almost Free-XOR

In this section, we briefly explain the Almost Free-XOR optimization from the paper GLPN15 [8], which enables the efficient evaluation of XOR gates in garbled circuit. This is a an alternative optimization to Free-XOR (section 2.3). Although this requires one ciphertext for XOR evaluation unlike Free-XOR, this is based on standard assumption (Pseudo Random Function Assumption).

Almost Free-XOR demands *point and permute* optimization in Yao's protocol. For each wire w of Yao's garbled circuit, a random bit called permutation bit π_w is chosen, and that bit will be associated with k_w^0, whereas $\overline{\pi}_w$ will be associated with k_w^1. Therefore while evaluation, a bit will be exposed along with the key which we call signal bit λ_w. Note that $\lambda_w = \pi_w$ if the key in the evaluation is k_w^0, and $\lambda_w = \overline{\pi}_w$ if the key is k_w^1. Let v_w be the actual bit on the wire w. So in general, it holds that $\lambda_w = \pi_w \oplus v_w$. Since π_w is random, this reveals nothing about the value v_w. Almost Free-XOR tries to mimic the Free-XOR, but standing within standard assumptions. But the basic idea of obtaining output key by taking XOR of the input keys stays the same.

2.4.1 Garbling

Let the XOR gate possess the id g and have input wires a, b and output wire c and F be a pseudo random function. The garbling of XOR gate according to the optimization works in the following way.

- **translate input keys on wire** a: Let $D_{a^0} := F_{k_a^0}(g)$, $D_{a^1} := F_{k_a^1}(g)$ be computed and called as derived keys.

- **set offset of output wire** c: Compute their XOR of the derived keys of wire a and call as Δ. i.e, $\Delta := D_{a^0} \oplus D_{a^1}$. This will be the offset of the two output keys on the wire c.

- **translate input keys on wire** b: Derived keys are computed on wire b as follows. $D_b^{\pi_b} := F_{k_b^{\pi_b}}(g)$ and $D_b^{\overline{\pi}_b} := D_b^{\pi_b} \oplus \Delta$. The two derived keys are computed differently to keep the bitwise distance (XOR) of D_b^0 and D_b^1 as Δ so that, we can perform XOR evaluation as XOR of the keys itself.

- **compute the output keys on wire** c: We set the output wire keys as $k_c^0 = D_a^0 \oplus D_b^0$ and $k_c^1 = k_c^0 \oplus \Delta$. We can observe that $D_a^{v_a} \oplus D_b^{v_b} = k_c^{v_a \oplus v_b}$, because of the bit difference Δ within each pair of keys on the input wires.

- **set the ciphertext**: We need it as finding $D_b^{\overline{\pi}_b}$ with $k_b^{\overline{\pi}_b}$ seems impossible, as it is not a function of that key. So we set a cipher text so that, only with the $k_b^{\overline{\pi}_b}$, the corresponding derived key can be obtained. So the cipher text $T := F_{k_b^{\overline{\pi}_b}} \oplus D_b^{\overline{\pi}_b}$.

2.4.2 Evaluation

In order to evaluate a XOR gate g with ciphertext T and the keys k_a and k_b on the wires a and b respectively, the evaluator needs to compute both the derived keys. They are $D_a = F_{k_a}(g)$ and $D_b = F_{k_b}(g)$ or $D_b = F_{k_b}(g) \oplus T$ depends on the signal bit λ_b. It is left to find the output wire key as $k_c = D_a \oplus D_b$.

As we are using PRF outputs with unique gate id's as seeds, for both computing ciphertexts as well as determining output wire keys, the security of this optimization is implied by PRF assumption. As PRF assumption holds, the output of PRF are pseudorandom and indistinguishable from a perfect random string for any polynomial time adversary. This makes it as good as choosing the keys of output wires as random. Also without $k_b^{\overline{\pi}_b}$, no information about $D_b^{\overline{\pi}_b}$ is leaked from T, as it is padded with pseudo-random string.

Chapter 3

Our Contributions

3.1 Cheap-XOR for BMR

Our contribution is an approach called *Cheap-XOR* that enables to evaluate the XOR gates with no cost in BMR's protocol and is equivalent to the Free-XOR technique in Yao's protocol. However, besides the key acquisition of output wire by performing XOR on two input wire keys, we do not adapt any techniques from the work of Kolesnikov et al. [9] and thereby preventing copyright issues regarding intellectual properties. Cheap XOR can be proven secure under semi-honest setting, with an underlying assumption on the existence of Random Oracle. We will be discussing Cheap-XOR protocol in the following sections.

We will be following the notations used in BMR's protocol, discussed in Section 2.2. Cheap-XOR protocol will structurally resembles BMR's protocol. Therefore, we will be having an offline phase and an online phase followed by local computation of circuit evaluation.

3.1.1 Cheap-XOR: Offline phase

In offline phase of Cheap-XOR, all parties will collectively decide upon π values and shared (like BMR's protocol), as long as the wire is *not* an output of an XOR gate. If wire c is output of an XOR gate, then $\pi_c := \pi_a \oplus \pi_b$. As the π_a and π_b are shared, one party may only possess π_a^i and π_b^i. So output wire c's π share is $\pi_c^i := \pi_a^i \oplus \pi_b^i$ [1]. Please note that, we may need reconstruction of π_z and broadcast of the same for each output wire z, if z is the output wire of some XOR gate. The structure of super-key is also similar to BMR, except the way in which the parties choose keys for wires differ from BMR's approach. For super-key calculation, each party P_i

[1] If (n,n) sharings are $\underset{i}{\oplus}(\pi_a^i) = \pi_a$, $\underset{i}{\oplus}(\pi_b^i) = \pi_b$, then being $\pi_c^i = \pi_a^i \oplus \pi_b^i$ means $\underset{i}{\oplus}(\pi_c^i) = \pi_a \oplus \pi_b$

randomly choose a κ length random string r^i, only once. Then they topologically sort all the gates (and wires), then choose key for a wire as follows:

1. If wire w is not the output wire of any XOR Gate,
   (a) Choose a random key $k^0_{w,i} \in_R \{0,1\}^\kappa$
   (b) $k^1_{w,i} := k^0_{w,i} \oplus r^i$
2. Else (Let a, b be input wires of the XOR gate)
   (a) $k^0_{w,i} := k^0_{a,i} \oplus k^0_{b,i}$
   (b) $k^1_{w,i} := k^0_{w,i} \oplus r^i$

The super-key computation is given below.

$$\begin{aligned}
\mathcal{SK}^0_w &= k^0_{w,1}||k^0_{w,2}||\cdots||k^0_{w,n} \\
\mathcal{SK}^1_w &= k^1_{w,1}||k^1_{w,2}||\cdots||k^1_{w,n} \\
&= k^0_{w,1} \oplus r^1||k^0_{w,2} \oplus r^2||\cdots||k^0_{w,n} \oplus r^n \\
&= \mathcal{SK}^0_w \oplus (r^1||r^2||\cdots||r^n) \\
&= \mathcal{SK}^0_w \oplus \Gamma
\end{aligned}$$

Where Γ is common for all wires, as all parties will use same r^i in all wires (assuming semi-honest behaviour). We can see that performing XOR on input keys will result in the output wire key, for XOR gates as similar to Section 2.3.2.

3.1.1.1 Garbling of Other Gates

We use two Random Oracles, namely $H^1, H^2 : \{0,1\}^\kappa \to \{0,1\}^{n\kappa}$ for encrypting cipher boxes of non-XOR gates. The Garbling of non-XOR gates can be done as follows.

$$\begin{aligned}
C_1 := &\left(H^1(k^0_{a,1}) \oplus H^1(k^0_{a,2}) \oplus \cdots \oplus H^1(k^0_{a,n})\right) \\
&\oplus \left(H^1(k^0_{b,1}) \oplus H^1(k^0_{b,2}) \oplus \cdots \oplus H^1(k^0_{b,n})\right) \oplus K^{r_1}_c
\end{aligned}$$

$$\begin{aligned}
C_2 := &\left(H^2(k^0_{a,1}) \oplus H^2(k^0_{a,2}) \oplus \cdots \oplus H^2(k^0_{a,n})\right) \\
&\oplus \left(H^1(k^1_{b,1}) \oplus H^1(k^1_{b,2}) \oplus \cdots \oplus H^1(k^1_{b,n})\right) \oplus K^{r_2}_c
\end{aligned}$$

$$C_3 := \left(H^1(k^1_{a,1}) \oplus H^1(k^1_{a,2}) \oplus \cdots \oplus H^1(k^1_{a,n})\right) \\ \oplus \left(H^2(k^0_{b,1}) \oplus H^2(k^0_{b,2}) \oplus \cdots \oplus H^2(k^0_{b,n})\right) \oplus K_c^{r_3}$$

$$C_4 := \left(H^2(k^1_{a,1}) \oplus H^2(k^1_{a,2}) \oplus \cdots \oplus H^2(k^1_{a,n})\right) \\ \oplus \left(H^2(k^1_{b,1}) \oplus H^2(k^1_{b,2}) \oplus \cdots \oplus H^2(k^1_{b,n})\right) \oplus K_c^{r_4}$$

Where,

$$r_1 = g(\pi_a, \pi_b) \oplus \pi_c$$
$$r_2 = g(\pi_a, \overline{\pi_b}) \oplus \pi_c$$
$$r_3 = g(\overline{\pi_a}, \pi_b) \oplus \pi_c$$
$$r_4 = g(\overline{\pi_a}, \overline{\pi_b}) \oplus \pi_c$$

3.1.2 Cheap-XOR: Online Phase

Cheap-XOR Online phase is same as that of BMR's protocol in Section 2.2, where λ values and $K_w^{\lambda_w}$ values are shared for each input wire w, followed by the evaluation of the circuit as a local computation. λ is propagated from input wires to output wires, and finally every party finds out bit value by computing $\pi_z \oplus \lambda_z$.

3.1.3 Correctness

Correctness in non-XOR gates directly follows from Section 2.2.3. For proving correctness in XOR gates, we will be using the following claim.

Claim 3.1 *If λ_a and λ_b are the computed values of λ on two input wires of an XOR gate g, then the output wire c will be associated with $\lambda_c = \lambda_a \oplus \lambda_b$.*

In correctness, we have to prove that the correlation between π, λ and v values are consistent with the output wire of XOR. Please note that, for an XOR gate with input wires a and b and output wire c, we chose π values in such a way that $\pi_c = \pi_a \oplus \pi_b$.

$$\begin{aligned} v_a \oplus v_b &= (\lambda_a \oplus \pi_a) \oplus (\lambda_b \oplus \pi_b) \\ &= (\lambda_a \oplus \lambda_b) \oplus (\pi_a \oplus \pi_b) \\ &= \lambda_c \oplus \pi_c \qquad \text{(follows from claim 1)} \\ &= v_c \end{aligned}$$

Proof: From the construction of super-keys, we can see that the following equation holds for all wires.

$$\mathcal{SK}_w^{\lambda_w} = \mathcal{SK}_w^0 \oplus (\lambda_w \cdot \Gamma) \tag{3.1}$$

If c is the output wire of XOR gate g with input wires a and b, then the super-keys of c must have been calculated from the super-keys of a and b, as follows.

$$\begin{aligned}
\mathcal{SK}_c^{\lambda_c} &= \mathcal{SK}_a^{\lambda_a} \oplus \mathcal{SK}_b^{\lambda_b} \\
&= \mathcal{SK}_a^0 \oplus (\lambda_a \cdot \Gamma) \oplus \mathcal{SK}_b^0 \oplus (\lambda_b \cdot \Gamma) \\
&= \mathcal{SK}_a^0 \oplus \mathcal{SK}_b^0 \oplus ((\lambda_a \oplus \lambda_b) \cdot \Gamma) \\
&= \mathcal{SK}_c^0 \oplus ((\lambda_a \oplus \lambda_b) \cdot \Gamma)
\end{aligned} \tag{3.2}$$

Applying Equation(3.1) for wire c,

$$\mathcal{SK}_c^{\lambda_c} = \mathcal{SK}_c^0 \oplus (\lambda_c \cdot \Gamma) \tag{3.3}$$

Comparing Equations(3.2) and (3.3),

$$(3.2)\&(3.3) \implies \lambda_c = (\lambda_a \oplus \lambda_b)$$

Hence we proved that correctness holds for both XOR and Non-XOR gates. ■

3.1.4 Security of Cheap-XOR

The Cheap-XOR differs from actual BMR protocol in the following areas:

- Construction of super-key $\mathcal{SK}_w^1$
- Choosing both super-keys of XOR output wires
- Encryption of cipher boxes (Non-XOR gates)
- Choosing λ values of XOR output wires.

Without loss of generality, we can assume P_1 is honest, but the rest are corrupted parties (worst case scenario). Here, $\mathcal{A}$ can get to know all r^i, except r^1. So $\mathcal{A}$ can predict $\mathcal{SK}_w^{1-\lambda_w}$ from $\mathcal{SK}_w^{\lambda_w}$ only with probability $1/2^k$, as they differ by Γ and it contains r^1. Therefore choosing $\mathcal{SK}_w^1 = \mathcal{SK}_w^0 \oplus \Gamma$ is secure. If $\mathcal{SK}_a^{\lambda_a}, \mathcal{SK}_b^{\lambda_b}$ are the input wire keys of an XOR gate, $\mathcal{SK}_c^{\lambda_c}$ will

be the XOR of the above two. Like we have shown in the previous case, $\mathcal{A}$ will have only a negligible chance to predict the $\mathcal{SK}_c^{1-\lambda_c}$ without the knowledge of r^1. In the encryptions of Non-XOR gates, we use Random Oracle outputs to mask the message. Even though the inputs may be correlated ($k_{a,i}^1 = k_{a,i}^0 \oplus r^i$) as long as they are different, Random Oracle will be returning perfect random strings and that hides messages. To know what exact information is leaked from bringing the correlation of λ values, let us analyse in the following way. Let's assume c is some output wire of an XOR gate g, whose input wires are a and b.

- **Case 1:** *Both a and b are input wires of corrupted parties:* Here, we can see that $\mathcal{A}$ may learn π_c which is $\pi_a \oplus \pi_b$. Our claim is that, even if we choose π_c randomly, $\mathcal{A}$ can compute it. Here's is how $\mathcal{A}$ can learn π_c: Compute $v_c = v_a \oplus v_b$, and compute $\pi_c = \lambda_c \oplus v_c$ (All parties will λ_c as the circuit execution proceeds).

- **Case 2:** *a, b are input wires, and only one wire (say, a) or neither of the wires are possessed by corrupted parites:* $\mathcal{A}$ will be missing one share of π_b (In our case, π_b^1), hence π_b is hidden. As $\pi_c = \pi_a \oplus \pi_b$, π_c is also hidden from $\mathcal{A}$.

- **Case 3:** *From wire c till output wire z, there is at least 1 non-XOR gate:* Assuming a and b are intermediate wires, $\mathcal{A}$ will not have any knowledge about π_a and π_b. Therefore, $\mathcal{A}$ only knows the relation $\pi_c = \pi_a \oplus \pi_b$. In the path from c to z, if there is a non-XOR gate, then π_c will be hidden from $\mathcal{A}$, as there is no correlation between π values of input and output wires of a non-XOR gate.

- **Case 4:** *From wire c till output wire z, all gates are XOR gates:* We take $w_1, w_2, \cdots w_n$ be the set of intermediate wires which are only connected with themselves using XOR gates with the final output wire is z. In other words, $v_z = v_{w_1} \oplus \cdots v_{w_n}$. All parties (and $\mathcal{A}$) know the value of π_z. Due to our construction, $\mathcal{A}$ may learn the value $(\pi_{w_1} \oplus \pi_{w_2} \cdots \pi_{w_n})$. However, we claim that this information $\mathcal{A}$ can learn even if we had chosen π values randomly. It is obvious to see that $\mathcal{A}$ knows $\lambda_{w_1}, \cdots \lambda_{w_n}$. As $v_z = \bigoplus_i (\lambda_{w_i} \oplus \pi_{w_i})$, $\mathcal{A}$ can compute $(\pi_{w_1} \oplus \pi_{w_2} \cdots \pi_{w_n})$. This implies $\mathcal{A}$ doesn't learn anything, apart from what he already knows.

3.2 Generic Distributed Garbling Scheme Definitions

We define a generic distributed garbling scheme which acts as a framework for any multi-party garbling protocol like BMR could fit in. Along with this, an optimization of XOR gates and NOT gates to BMR sums up our second contribution. The generic distributed garbling

scheme enables us to define properties privacy, obliviousness and authenticity through code-based games. These are experiments where we expect no polynomial time adversary can win in the game with non-negligible probability. We will claim that with the presence of an adversary who can wins the experiment(s), we can solve a hard problem in polynomial time, which is believed to be not true. In the following sections, we will first show that the BMR protocol fits into this definition, then define our optimization Almost Free-XOR in BMR, followed by showing if BMR with our optimizations fails any experiment, we can solve a hard problem called 2PRF, thereby proving the properties.

Syntax. We follow the circuit conventions defined in [3]. A circuit is a 6-tuple $f = (u, m, q, A, B, G)$. Here $u \geq 2$ is the number of inputs, $m \geq 1$ is the number of outputs, and $q \geq 1$ is the number of gates. We let $r = u + q$ be the number of wires. We let $\mathsf{Inputs} = \{1, \cdots, u\}$, $\mathsf{Wires} = \{1, \cdots, u+q\}$, $\mathsf{OutputWires} = \{u+q-m+1, \cdots, u+q\}$, and $\mathsf{Gates} = \{u+1, \cdots, u+q\}$. Every gate in a circuit has two incoming wires, and one outgoing wire (by which it is indexed). Then $A : \mathsf{Gates} \mapsto \mathsf{Wires} \backslash \mathsf{OutputWires}$ is a function to identify each gate's first incoming wire and $B : \mathsf{Gates} \mapsto \mathsf{Wires} \backslash \mathsf{OutputWires}$ is a function to identify each gate's second incoming wire. Finally $G : \mathsf{Gates} \times \{0,1\}^2 \mapsto \{0,1\}$ is a function that determines the functionality of each gate. We require $A(g) < B(g) < g$ for all $g \in \mathsf{Gates}$.

3.2.1 Syntax

Our garbling scheme uses a slight variant of the notation of Bellare et al. [3] in which a garbling scheme consists of 4 algorithms:

– $\mathsf{Garble}(1^n, c) \to (C, \Xi, E, d)$ is an algorithm that takes as input a security parameter 1^n and a description of a boolean circuit c, and returns a tuple (C, Ξ, E, d), where $C = (c_1, c_2, \cdots, c_q)$ represents the garbled circuit, $\Xi = \xi_1 || \cdots || \xi_i || \cdots || \xi_n$ represents the evaluation information, $E = (e_1, e_2, \cdots, e_u)$ represents input encoding information and d represents output decoding information. The output decoding information d is public.

– $\mathsf{Encode}(E, x) \to X$ is a function that takes as input encoding information E and input $x = (x_1, x_2, \cdots, x_u)$ and returns garbled input $X = (X_1, X_2, \cdots, X_u)$. The garbled input X is public and the encoding is necessarily projective. [1]

[1] Projective garbling schemes, as defined by [3], adhere to the rule that the garbled input X can be parsed as tokens $X_1, \cdots, X_u$, and that any token X_i depends only on input bit x_i. ie. Encoding inputs $x = x_1, \cdots, x_u$ and $x' = x'_1, \cdots, x'_u$ with the same encoding information e will produce garbled inputs $X = (X_1, \cdots, X_u)$, $X' = (X'_1, \cdots, X'_u)$ with $X_i = X'_i$ if $x_i = x'_i$

– $\mathsf{Eval}(C, X, \xi_i) \rightarrow Y$ is a function that takes as input a garbled circuit C, garbled input X, and a single party's evaluation information ξ_i and returns garbled output Y (corresponding to the concrete output $y = c(x)$).

– $\mathsf{Decode}(Y, d) \rightarrow y$ is a function that takes as input decoding information d and garbled output Y and returns the clear output y of the circuit.

3.2.2 Code-based Experiments

Syntax for the experiment: For the ease of representing the collective information held by the corrupted parties, we define the set of indices of corrupted parties as $I = \{i_1, i_2, \cdots, i_t\}$. The inputs provided by the corrupted parties are denoted $\mathcal{X}_I = (\mathcal{X}_{i_1}, \mathcal{X}_{i_2}, \cdots, \mathcal{X}_{i_t})$, and the shares of the encoding information available to the corrupted parties are $\mathcal{E}_I = (e_{i_1}, e_{i_2}, \cdots, e_{i_t})$ respectively. Then, $x \setminus \mathcal{X}_I$ denotes the input of honest parties. The evaluation information known to the corrupted parties is denoted ξ_I.

We define three security notions for garbling schemes adhering to the specification of 3.2.1, along the lines of [3] as follows:

–**Privacy:** The tuple $(C, X, \xi_I, \mathcal{E}_I, d)$ should not reveal any information about $x \setminus \mathcal{X}_I$ (input of honest parties) that cannot be learned directly from $c(x)$ and $\mathcal{X}_I$. More formally, there exists a simulator $\mathcal{S}$ that receives input $(1^\kappa, c, c(x), \mathcal{X}_I)$ and outputs a simulated garbled circuit, encoded input, decoding information, along with the corrupted parties' shares of encoding information $(\mathcal{E}_I)$ and evaluation information ξ_I that is indistinguishable from $(C, X, \xi_I, \mathcal{E}_I, d)$ legitimately generated using the real garbling functions $\mathsf{Garble}(1^n, c)$ and $\mathsf{Encode}(E, x)$. Observe that $\mathcal{S}$ knows the output $c(x)$ and corrupted parties' inputs $\mathcal{X}_I$, but does not know the input $x \setminus \mathcal{X}_I$.

–**Obliviousness:** $(C, X, \xi_I, \mathcal{E}_I)$ should not reveal any information about x. i.e., there exists a simulator $\mathcal{S}$ that receives input $(1^\kappa, c, \mathcal{X}_I)$ and outputs a simulated garbled circuit and garbled input, along with the corrupted parties' shares of encoding information $(\mathcal{E}_I)$ and evaluation information ξ_I that is indistinguishable from $(C, X, \xi_I, \mathcal{E}_I)$ legitimately generated using the real garbling functions $\mathsf{Garble}(1^\kappa, c)$ and $\mathsf{Encode}(E, x)$. Observe that $\mathcal{S}$ here is not even given the output.

–**Authenticity:** Given $(C, X, \xi_I, \mathcal{E}_I)$ as input, no adversary should be able to forge a *different* garbled output $\widetilde{Y}$ that is not obtained as a direct output of $\mathsf{Eval}(C, X)$. More formally, a probabilistic-polynomial time adversary should be able to output $\widetilde{Y} \neq \mathsf{Eval}(C, X)$ such that $\mathsf{Decode}(\widetilde{Y}, d) \neq \perp$, with only negligible probability.

For each security definition we define an experiment that formalizes the adversary's task. In the following, G denotes a garbling scheme that consists of the 4 algorithms stated in 3.2.1, and $\mathcal{S}$ denotes a simulator.

The basic non-triviality requirement for a garbling scheme, called correctness, is that for every circuit c and input $x \in \{0,1\}^{poly(n)}$, it holds that $\mathsf{Decode}(\mathsf{Eval}(C, \mathsf{Encode}(E, x), \xi_i), d) = c(x)$ except with negligible probability, where $(C, E, \Xi, d) \leftarrow \mathsf{Garble}(1^n, c)$, and ξ_i is parsed from Ξ (correctness should hold for every ξ_i parsed from Ξ).

We define the privacy, obliviousness and authenticity experiments as $\mathsf{Expt}^{\mathsf{priv}}_{G,\mathcal{A},\mathcal{S}}(\kappa)$, $\mathsf{Expt}^{\mathsf{auth}}_{G,\mathcal{A}}(\kappa)$ and $\mathsf{Expt}^{\mathsf{auth}}_{G,\mathcal{A}}(\kappa)$ respectively. For any scheme to be secure w.r.t. to the definitions, no poly-time adversary should win the experiments with a non-negligible probability.

The privacy experiment $\mathsf{Expt}^{\mathsf{priv}}_{G,\mathcal{A},\mathcal{S}}(\kappa)$

1 Invoke adversary $\mathcal{A}$: compute $(c, \mathcal{X}_I, I) \leftarrow \mathcal{A}(1^\kappa)$
2 Choose the input of honest parties $x \setminus \mathcal{X}_I$.
3 Choose a random bit $\beta \in \{0,1\}$.
4 **if** $\beta = 0$ **then** Compute $(C, E, \Xi, d) \leftarrow \mathsf{Garble}(1^\kappa, c)$ and $X \leftarrow \mathsf{Encode}(E, x)$; $\xi_I \leftarrow \Xi[I]$;
5 **else** Compute $(C, X, \xi_I, \mathcal{E}_I, d) \leftarrow \mathcal{S}(1^\kappa, c, c(x), \mathcal{X}_I)$;
6 Give $\mathcal{A}$ the challenge $(C, X, \xi_I, \mathcal{E}_I, d)$ and obtain its guess: $\beta' \leftarrow \mathcal{A}(C, X, \xi_I, \mathcal{E}_I, d)$
7 Output 1 if and only if $\beta' = \beta$.

A distributed garbling scheme G achieves the property privacy if for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ and a negligible function $\mu(\kappa)$ such that, $\forall \kappa \in \mathbb{N}$:

$$\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{G},\mathcal{A},\mathcal{S}}(\kappa) = 1\right] \leq 1/2 + \mu(\kappa)$$

The obliviousness experiment $\mathsf{Expt}^{\mathsf{oblv}}_{G,\mathcal{A},\mathcal{S}}(\kappa)$

1 Invoke adversary $\mathcal{A}$: compute $(c, \mathcal{X}_I, I) \leftarrow \mathcal{A}(1^\kappa)$
2 Choose the input of honest parties $x \setminus \mathcal{X}_I$.
3 Choose a random bit $\beta \in \{0,1\}$.
4 **if** $\beta = 0$ **then** compute $(C, E, \Xi, d) \leftarrow \mathsf{Garble}(1^\kappa, c)$ and $X \leftarrow \mathsf{Encode}(E, x)$; $\xi_I \leftarrow \Xi[I]$;
5 **else** compute $(C, X, \xi_I, \mathcal{E}_I) \leftarrow \mathcal{S}(1^\kappa, c, \mathcal{X}_I)$;
6 Give $\mathcal{A}$ the challenge $(C, X, \xi_I, \mathcal{E}_I)$ and obtain its guess: $\beta' \leftarrow \mathcal{A}(C, X, \xi_I, \mathcal{E}_I)$
7 Output 1 if and only if $\beta' = \beta$.

A distributed garbling scheme G achieves the property obliviousness if for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ and a negligible function $\mu(\kappa)$ such that, $\forall \kappa \in \mathbb{N}$:

$$\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{oblv}}_{\mathsf{G},\mathcal{A},\mathcal{S}}(\kappa) = 1\right] \leq 1/2 + \mu(\kappa)$$

The authenticity experiment $\mathsf{Expt}^{\mathsf{auth}}_{G,\mathcal{A}}(\kappa)$

1 Invoke adversary $\mathcal{A}$: compute $(c, \mathcal{X}_I, I) \leftarrow \mathcal{A}(1^\kappa)$
2 Choose the input of honest parties $x \setminus \mathcal{X}_I$.
3 Compute $(C, E, d) \leftarrow \mathsf{Garble}(1^\kappa, c)$ and $X \leftarrow \mathsf{Encode}(E, x)$
4 Give $\mathcal{A}$ the challenge $(C, X, \mathcal{E}_I)$ and obtain its output: $\widetilde{Y} \leftarrow \mathcal{A}(C, X, \mathcal{E}_I)$
5 Output 1 if and only if $\mathsf{Decode}(\widetilde{Y}, d) \notin \{\perp, c(x)\}$

A distributed garbling scheme G achieves the property authenticity if for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mu(\kappa)$ such that, $\forall \kappa \in \mathbb{N}$:

$$\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{auth}}_{\mathsf{G},\mathcal{A}}(\kappa) = 1\right] \leq \mu(\kappa)$$

3.3 BMR as a distributed garbling scheme

We define the 4 functions $\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}$ and Decode of the distributed garbling scheme which resembles the BMR protocol. The intension is to show that BMR protocol fits in to the generic scheme for distributed garbling.

Garble($1^n, c$)

$C = \Xi = E = d = \Phi$
for $i \in [1, n], w \in [1, u]$ **do**
  Choose random $\mathcal{K}^0_{w,i}$, $\mathcal{K}^1_{w,i}$, $\pi_{w,i}$.
  $e_i = \{\mathcal{K}^0_{w,i}, \mathcal{K}^1_{w,i}, \pi_{w,i}\}$; $E = E \cup e_i$
  $\mathcal{SK}^b_w = \mathcal{K}^b_{w,1} || \cdots || \mathcal{K}^b_{w,n} \forall b \in \{0, 1\}$
end
for $i \in [1, n], \forall w$ *the wires* P_i *owns* **do**
  $e_i = e_i \cup \{\pi_w = \oplus_{i=1}^n \pi_{w,i}\}$
end
for $i \in [1, n], w \in [u + 1, u + q]$ **do**
  Choose random $\mathcal{K}^0_{w,i}$, $\mathcal{K}^1_{w,i}$.
  $\Xi = \Xi \cup \{\mathcal{K}^0_{w,i}, \mathcal{K}^1_{w,i}\}$
  $\mathcal{SK}^b_w = \mathcal{K}^b_{w,1} || \cdots || \mathcal{K}^b_{w,n} \forall b \in \{0, 1\}$
end
for $g \in [1, q]$ **do**
  $C = Gb\_g(\mathcal{SK}^0_a, \mathcal{SK}^1_a, \mathcal{SK}^0_b, \mathcal{SK}^1_b, \mathcal{SK}^0_w, \mathcal{SK}^1_w, g)$ (where a,b are the input wires, w the output wire)
end
for $w \in$ OutputWires **do**
  $d_w[0] := F_{\mathcal{SK}^{\pi_w}_w}(w || \pi_w)$
  $d_w[1] := F_{\mathcal{SK}^{\overline{\pi}_w}_w}(w || \overline{\pi}_w)$
end
return C, Ξ, E, d

Encode(E, x)

$X = \Phi$
for $i \in [1, u]$ **do**
  $\mathcal{SK}_i := \$$
  $\pi_i := e_r[\pi_i]$ where r is the owner of wire i
  for $j \in [1, n]$ **do**
    $\mathcal{SK}_i = \mathcal{SK}_i || \left(e_j[i][x_i \oplus \pi_i]\right)$
  end
  $X = X \cup \mathcal{SK}_i$
end
return X

Eval(C, X, ξ_i)

Parse the $\mathcal{SK}_w$ for all input wire w from X.
Identify λ_w ($\lambda_w = b$ if $\mathcal{K}_{w,i} = \mathcal{K}^b_{w,i}$ from Ξ_i)
Obtain $\mathcal{SK}_w$ of output wire w by decoding $C_{(\lambda_a \times 2 + \lambda_b)}$ of gate $g(a, b) \to w$.
Repeat steps 2, 3 in topological order of gates.
On output wires out_1 to out_m, $Y := \left(F_{\mathcal{SK}_{out_1}}(out_1 || \lambda_{out_1}), \cdots, F_{\mathcal{SK}_{out_m}}(out_m || \lambda_{out_m})\right)$
return Y

Decode(Y, d)

for $w \in$ OutputWires **do**
  if $d_w[0] = Y_w$ **then**
    $y_w := 0$
  else if $d_w[1] = Y_w$ **then**
    $y_w := 1$
  else
    return $\perp$
end
return $y = y_1||\cdots||y_m$

$Gb\_g(\mathcal{SK}_a^0, \mathcal{SK}_a^1, \mathcal{SK}_b^0, \mathcal{SK}_b^1, \mathcal{SK}_w^0, \mathcal{SK}_w^1, g)$

$\gamma = type(g)$ (where $type(g)$ returns AND/OR/XOR, etc.)
Parse the input superkeys into partial keys: $\mathcal{K}_{a,1}^0||...||\mathcal{K}_{a,i}^0||...||\mathcal{K}_{a,n}^0 = \mathcal{SK}_a^0$ $\mathcal{K}_{a,1}^1||...||\mathcal{K}_{a,i}^1||...||\mathcal{K}_{a,n}^1 = \mathcal{SK}_a^1$ $\mathcal{K}_{b,1}^0||...||\mathcal{K}_{b,i}^0||...||\mathcal{K}_{b,n}^0 = \mathcal{SK}_b^0$ $\mathcal{K}_{b,1}^1||...||\mathcal{K}_{b,i}^1||...||\mathcal{K}_{b,n}^1 = \mathcal{SK}_b^1$
Generate the ciphertexts such that the ciphertext at position $(\pi_a \oplus \beta_a, \pi_b \oplus \beta_b)$ will contain the key corresponding to $\pi_g \oplus \beta_g$
$C_{(\beta_1 \times 2 + \beta_2)} = \bigoplus_{i=1}^{n} F_{\mathcal{K}_{a,i}^{\beta_1}}(g||\beta_1||\beta_2) \oplus \bigoplus_{i=1}^{n} F_{\mathcal{K}_{b,i}^{\beta_2}}(g||\beta_1||\beta_2) \oplus \mathcal{SK}_g^{\pi_g \oplus \gamma(\pi_a, \pi_b)}$ (where $\beta_1, \beta_2 \in \{0,1\}$. Same as how it's done in BMR.)
return C_0, C_1, C_2, C_3

3.4 Almost Free-XOR in BMR

As we have seen, Almost Free-XOR garbling scheme produces one ciphertext per XOR gate, and requires only three PRF invocations per XOR gate in the two party Yao-style setting. Here, we adapt this optimization for distributed garbling in an n-party setting.

We assume a [1] style superkey structure; the superkey for bit b on a wire w is of the form

$$\mathcal{SK}_w^b = \mathcal{K}_{w,1}^b||\mathcal{K}_{w,2}^b||...||\mathcal{K}_{w,i}^b||...||\mathcal{K}_{w,n}^b$$

where $\mathcal{K}_{w,i}^b$ is the partial key contributed by party P_i.

It can be observed that the Almost Free-XOR optimization essentially produces keys for the output wire as a function of derivations of the input wire keys. Along these lines, we adapt this optimization (to the distributed setting) by making each partial key on the output wire a function of derivations of the corresponding partial input wire keys. We will later see that this allows the XOR gates to be garbled locally with no communication whatsoever before the

online phase; each κ-bit block of the ciphertext can be generated independently of the parts of the input and output keys at different positions.

The outline of the garbling scheme is as follows: in an n-party setting, each party locally, independently executes an instance of the Almost Free-XOR garbling for an XOR gate, with its partial keys on the input and output wires. As per the original scheme, there is one ciphertext which encrypts the derived key corresponding to bit value 1 on the right input wire, with the original 1 key on that wire. In the n-party setting, each party locally encrypts the partial derived 1 key with the original partial 1 key on the right input wire. A simple concatenation of every party's ciphertext will comprise the entire ciphertext for the gate. Hence, the only communication required to garble an XOR gate in this scheme is one broadcast round, to construct the ciphertext for the gate. This is a significant improvement over using the generic gate garbling described in [1], which requires a secure MPC to generate four ciphertexts for each gate. The output permutation bit is computed as the XOR of the input permutation bits.

Consider the left and right incoming wires of an XOR gate to be labeled a and b respectively. w. Even though the label of outgoing wire of gate g will be g itself, we will use w as the label for the output wire, to avoid confusion. The outline of the distributed Almost Free garbling of XOR gates is as follows:

1. Each party P_i locally computes their partial derived keys on wire a as $D^0_{a,i} = F_{\mathcal{K}^0_{a,i}}(g)$ and $D^1_{a,i} = F_{\mathcal{K}^1_{a,i}}(g)$
2. Each party P_i computes its partial offset $\Delta_{g,i} = D^0_{a,i} \oplus D^1_{a,i}$
3. The partial key corresponding to value 0 on wire b is derived for this gate as $D^0_{b,i} = F_{\mathcal{K}^0_{b,i}}(g)$
4. The partial derived key corresponding to 1 on wire b is set such that it is offset from the partial derived key corresponding to 0 on the same wire by $\Delta_{g,i}$; ie. $D^1_{b,i} = D^0_{b,i} \oplus \Delta_{g,i}$
5. The partial output wire keys are set such that XORing the available partial input wire keys will produce the appropriate output wire key. $\mathcal{K}^0_{g,i} := D^0_{a,i} \oplus D^0_{b,i} = D^1_{a,i} \oplus D^1_{b,i}$ and $\mathcal{K}^1_{g,i} := D^1_{a,i} \oplus D^0_{b,i} = D^0_{a,i} \oplus D^1_{b,i}$
6. As $D^1_{b,i}$ can not be directly derived from $\mathcal{K}^1_{b,i}$ like the other derived keys, P_i publishes a ciphertext encrypting $D^1_{b,i}$ using $\mathcal{K}^1_{b,i}$ as follows: $T_{g,i} := F_{\mathcal{K}^1_{b,i}}(g) \oplus D^1_{b,i}$

7. Functionality wise, along the lines of the [1] superkey, concatenating the partial elements belonging to each party will provide the corresponding whole element. Specifically,

$$D_a^0 = D_{a,1}^0 || \cdots || D_{a,i}^0 || \cdots || D_{a,n}^0$$

The same relation holds for all the derived keys D_a^1, D_b^1, D_b^0. Also worth noting is that the gate offset Δ_g, even though never explicitly computed in its whole form by any party, adheres to the same format.

$$\Delta_g = \Delta_{g,1} || \cdots || \Delta_{g,i} || \cdots || \Delta_{g,n}$$

Each party broadcasts its locally computed ciphertext from the previous step, so that the overall gate ciphertext may be constructed as follows:

$$T_g = T_{g,1} || \cdots || T_{g,i} || \cdots || T_{g,n}$$

Formally, we define the distributed version of this scheme in Algorithm 3.4.3.

The intuition for security is that every party's contribution (the derived key $D_{b,i}^1$) is masked by a PRF keyed with a value known only to that party.

3.4.1 Privacy

Privacy of the permutation bit π_w is preserved, as can be shown by examining all possible cases for π_a and π_b. Consider $\mathcal{S} = P \setminus \{P_i\}$ to be an abstract party representing every party except P_i.

1. $\mathcal{S}$ owns only wire a, and hence knows only π_a. In this case, π_b is known only to party P_i. Hence, π_w is masked information theoretically by π_b to $\mathcal{S}$, and by π_a to P_i (as $\pi_w = \pi_a \oplus \pi_b$).

2. $\mathcal{S}$ owns only wire b, and P_i owns a. The same argument as above applies.

3. $\mathcal{S}$ owns both wires, a as well as b. In this case, wire w is effectively owned by $\mathcal{S}$ in the security proof. Clearly, P_i knows nothing beyond her own share of π_a, π_b and π_w.

4. P_i owns both wires, a as well as b. In this case, wire w is effectively owned by P_i in the security proof. Clearly, $\mathcal{S}$ knows nothing beyond her own share of π_a, π_b and π_w.

3.4.2 Correctness

The intuition behind correctness of the scheme is that the n-party version of the Almost Free-XOR scheme that is presented above is merely a concatenation of n independent instances of the 2-party Almost Free-XOR scheme from [8].

The relation between the keys of the input and output wires is as follows:

$$\mathcal{K}^0_{w,i} = D^0_{a,i} \oplus D^0_{b,i} = D^1_{a,i} \oplus D^1_{b,i}$$

$$\begin{aligned}\mathcal{K}^1_{w,i} = \mathcal{K}^0_{w,i} \oplus \Delta_{g,i} &= D^0_{a,i} \oplus D^0_{b,i} \oplus \Delta_{g,i} \\ &= D^0_{a,i} \oplus D^1_{b,i} = D^0_{a,i} \oplus D^1_{b,i}\end{aligned}$$

Correctness for the mapping $\{\mathcal{K}_{a,i}\} \times \{\mathcal{K}_{b,i}\} \times \{C_i[g]\} \mapsto \{\mathcal{K}_{w,i}\}$ follows from [8], as each party locally executes that exact protocol to generate $C_i[g]$, $\mathcal{K}^0_{w,i}$, $\mathcal{K}^1_{w,i}$ from $\mathcal{K}^0_{a,i}, \mathcal{K}^1_{a,i}, \mathcal{K}^0_{b,i}, \mathcal{K}^1_{b,i}$. The zero and one superkeys for wire w are concatenations of $\mathcal{K}^0_{w,i}, \forall i$ and $\mathcal{K}^1_{w,i}, \forall i$ respectively. Hence, as each of the n blocks of k bits of the superkeys of the input wires correctly map to the corresponding k bit block of the output wire superkey using the (public) ciphertext, correctness is preserved upon concatenation.

3.4.3 Signal and Permutation bit Invariant

Both [1] and [8] maintain the following invariant across every wire w : $\pi_w \oplus \lambda_w = v_w$, where v_w is the actual bit value on the wire during evaluation. It can be seen that the *Distributed_GbXOR* and *Eval_XOR* procedures preserve this invariant.

For an XOR gate g, consider the invariant to be true for incoming wires a and b. Hence we have $\pi_a \oplus \lambda_a = v_a$ and $\pi_b \oplus \lambda_b = v_b$. By virtue of the gate functionality, we know that $v_w = v_a \oplus v_b$. While garbling, the *Local_GbXOR* procedure ensures that $\pi_w = \pi_a \oplus \pi_b$. During evaluation, *Eval_XOR* sets $\lambda_w = \lambda_a \oplus \lambda_b$.

Hence we have:

$$\begin{aligned}\pi_w \oplus \lambda_w =&(\pi_a \oplus \pi_b) \oplus (\lambda_a \oplus \lambda_b) \\ =&(\pi_a \oplus \lambda_a) \oplus (\pi_b \oplus \lambda_b) \\ =&(v_a) \oplus (v_b) \\ =&v_w\end{aligned} \tag{3.4}$$

Therefore, the invariant $\pi_w \oplus \lambda_w = v_w$ is maintained for the output wire of an XOR gate garbled and evaluated with Algorithms $\mathsf{Gb_{XOR}}()$ and $\mathsf{Ev_{XOR}}()$ respectively, given that the input wires maintain the same invariant.

It is left for us to define the garbling and evaluation sub-procedures specific for XOR gates such that the distributed garbling scheme for BMR can be modified. Unlike the normal gate-specific garbling function (Gb_g), this determines the keys of output wire too. This means there is a slight difference in Garble algorithm (3.3) too, where the output keys are determined by the Gb_XOR algorithm.

$Gb\_XOR(\mathcal{SK}_a^0, \mathcal{SK}_a^1, \mathcal{SK}_b^0, \mathcal{SK}_b^1, g)$

Parse the superkeys into partial keys:
$\cdots ||\mathcal{K}_{b,i}^0|| \cdots \leftarrow \mathcal{SK}_b^0$
$\cdots ||\mathcal{K}_{b,i}^1|| \cdots \leftarrow \mathcal{SK}_b^1$
Compute the gate offset $\Delta_g := F_{\mathcal{K}_{a,i}^0}(g) \oplus F_{\mathcal{K}_{a,i}^1}(g)$
Set the output keys: $\mathcal{SK}_w^0 := F_{\mathcal{K}_{a,i}^0}(g) \oplus F_{\mathcal{K}_{b,i}^0}(g)$
$\mathcal{SK}_w^1 := \mathcal{SK}_w^0 \oplus \Delta_g$
Compute the derived keys for wire b: $D_{b,i}^0 := \mathcal{K}_{b,i}^0$; $D_{b,i}^1 := D_{b,i}^0 \oplus \Delta_g$
Compute the ciphertext for XOR gate g: $C_g = F_{\mathcal{K}_{b,1}^1}(g) \oplus D_{b,1}^1 || ... || F_{\mathcal{K}_{b,i}^1}(g) \oplus D_{b,i}^1 || ... || F_{\mathcal{K}_{b,n}^1}(g) \oplus D_{b,n}^1$
// Note that this can be accomplished in one round by each party P_i broadcasting $F_{\mathcal{K}_{b,i}^1}(g) \oplus D_{b,i}^1$
return $C_g, \mathcal{SK}_w^0, \mathcal{SK}_w^1$

The Eval algorithm for BMR protocol needs to be updated by incorporating the Ev_XOR whenever XOR gates are evaluated.

$Ev\_XOR(\mathcal{SK}_a, \mathcal{SK}_b, \lambda_a, \lambda_b, C, g)$

Parse the superkeys:
$\mathcal{K}_{a,1}||...||\mathcal{K}_{a,i}||...||\mathcal{K}_{a,n} = \mathcal{SK}_a$
$\mathcal{K}_{b,1}||...||\mathcal{K}_{b,i}||...||\mathcal{K}_{b,n} = \mathcal{SK}_b$
Derive key for wire a:
$D_a = F_{\mathcal{K}_{a,1}}(g)||...||F_{\mathcal{K}_{a,i}}(g)...||F_{\mathcal{K}_{a,n}}(g)$
Invoke and concatenate PRFs using parsed superkey of wire b:
$D'_b = F_{\mathcal{K}_{b,1}}(g)||...||F_{\mathcal{K}_{b,i}}(g)...||F_{\mathcal{K}_{b,n}}(g)$
if $\lambda_b = 0$ **then**
  $D_b = D'_b$
else
  $D_b = C \oplus D'_b$
end
Compute output superkey for wire g: $\mathcal{SK}_g = D_a \oplus D_b$
return $\mathcal{SK}_w$

Proving our scheme achieves Privacy and Obliviousness, we will first define a Simulator $\mathcal{S}$ which can construct the garbled circuit only with the information provided by the game. We then claim that the garbled circuits constructed in the legitimate way is indifferent from the one created by $\mathcal{S}$. If otherwise, we can solve a hard problem called 2PRF [8] with the adversary $\mathcal{A}$. Here, we will prove the security of our optimization through the construction of Simulator $\mathcal{S}$ and the reduction to 2PRF.

3.4.4 Security of Almost Free-XOR in BMR

priliminaries

We define the experiment $\mathsf{Expt}^{\mathsf{2PRF}}_{F,\mathcal{A}}(n,\sigma)$ [8] briefly here. Here, the adversary $\mathcal{A}$ is given access to 4 oracle services $(\mathcal{O}^1(.),\mathcal{O}^2(.),\mathcal{O}^3(.),\mathcal{O}^4(.))$. The experiment starts with challenger chooses two random keys k_1,k_2, two truly random functions f^1,f^2 and a random bit σ. If $\sigma=0$, challenger provides the oracles as $(\mathcal{O}^1(.),\mathcal{O}^2(.),\mathcal{O}^3(.),\mathcal{O}^4(.)) = (F_{K_1}(.),F_{K_1}(.),F_{K_2}(.),F_{K_2}(.))$ and if $\sigma=1$, $(\mathcal{O}^1(.),\mathcal{O}^2(.),\mathcal{O}^3(.),\mathcal{O}^4(.)) = (f^1,F_{K_1}(.),f^2,F_{K_2}(.))$. Clearly, if $\mathcal{A}$ makes two queries of same value, say, x to both $\mathcal{O}^1(.)$ and $\mathcal{O}^2(.)$ then the σ can be determined. The hardness of this experiment exists because the same query is not allowed twice to the groups $(\mathcal{O}^1(.),\mathcal{O}^2(.))$, $(\mathcal{O}^3(.),\mathcal{O}^4(.))$. It is intuitive to see the hardness as F is PRF because, it's output is indistinguishable from a truly random function. The only possibility to distinguish is to see both the outputs of same query from $(\mathcal{O}^1(.),\mathcal{O}^2(.))$, or from $(\mathcal{O}^3(.),\mathcal{O}^4(.))$ which is prevented by the game itself. $\mathcal{A}$ should try to predict σ, and the game outputs the same bit.

Claim 3.2 *For any PPT adversary $\mathcal{A}$, any negligible function $\mu(n)$ and for any pseudo random function F*

$$\left|Pr\left[\mathsf{Expt}^{\mathsf{2PRF}}_{F,\mathcal{A}}(n,0)=1\right] - Pr\left[\mathsf{Expt}^{\mathsf{2PRF}}_{F,\mathcal{A}}(n,1)=1\right]\right| \leq \mu(n)$$

The formal proof of this claim is in the paper GLNP15 [8]. For proving the security of our scheme, we need a special PRF denoted as $\hat{F}:\{0,1\}^{nk}\times\{0,1\}^*\rightarrow\{0,1\}^{nk}$. Let $F:\{0,1\}^k\times\{0,1\}^*\rightarrow\{0,1\}^k$. Now let $\hat{F}$ be defined as follows:

$$\hat{F}_{\mathcal{SK}}(.) := F_{_1}(.)||F_{_2}(.)||\cdots||F_{_n}(.)$$

where $\mathcal{SK} :=_1||\cdots||_n$

Claim 3.3 *if F is a PRF, then $\hat{F}$ is a PRF.*

Proof: Proof of this claim with goes in a hybrid approach. We define a series of functions as follows.

$$
\begin{aligned}
H_{0,\mathcal{SK}}(.) &= F_1(.) \,||\, F_2(.) \,||\, \cdots \,||\, F_n(.) \\
H_{1,\mathcal{SK}}(.) &= \; r_1 \;||\, F_2(.) \,||\, \cdots \,||\, F_n(.) \\
H_{2,\mathcal{SK}}(.) &= \; r_1 \;||\; r_2 \;||\, F_3(.) \,||\, \cdots \,||\, F_n(.) \\
&\vdots \\
H_{i,\mathcal{SK}}(.) &= \; r_1 \;||\; r_2 \;||\, \cdots \,||\; r_i \;||\, F_{i+1}(.) \,||\, \cdots \,||\, F_n(.) \\
&\vdots \\
H_{2,\mathcal{SK}}(.) &= \; r_1 \;||\; r_2 \;||\, \cdots \,||\; r_n
\end{aligned}
$$

where $r_i \in_R \{0,1\}^k$. It can be observed that $H_{0,\mathcal{SK}}$ is $\hat{F}$ itself, and $H_{n,\mathcal{SK}}$ return completely random string. We can assume the contradiction of the claim i.e., $\exists$ a PPT adversary $\mathcal{A}$ who can distinguish $H_{0,\mathcal{SK}}$ and $H_{n,\mathcal{SK}}$ which is the reason $\hat{F}$ is not a PRF. This implies $\exists \mathcal{A}_i$ which can distinguish $H_{i,\mathcal{SK}}$ and $H_{i-1,\mathcal{SK}}$. Therefore we can construct an efficient adversary $\mathcal{A}^*$ for the PRF game of F, using the $\mathcal{A}_i$.

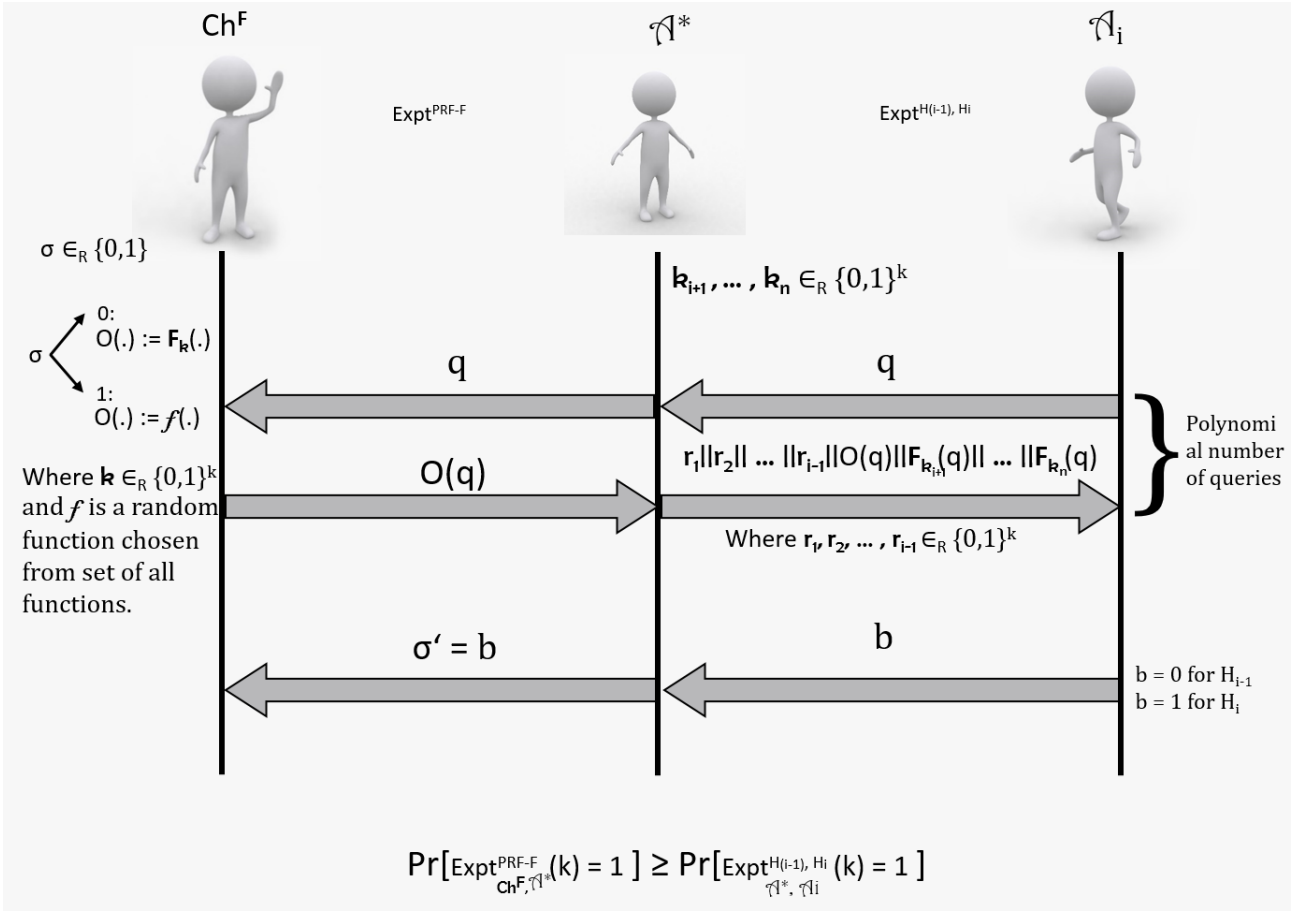

Figure 3.1: Proof of Claim 3

If there us an adversary $\mathcal{A}_i$ which has non-negligible advantage on distinguishing $H_{i,\mathcal{SK}}$ and $H_{i-1,\mathcal{SK}}$, we can construct another adversary $\mathcal{A}^*$ which has the a non-negligible advantage in winning PRF game of F, which is a contradiction as F is a PRF. Hence the claim is true.

■

3.4.5 Simulation

We are going to simulate the Garble according to our new scheme of garbling. As per the privacy experiment, Simulator $\mathcal{S}$ is provided with only the circuit $c, \mathcal{X}_I$ and the output $c(x)$, and $\mathcal{S}$ is expected to return $(C, X, \xi_I, \mathcal{E}_I, d)$ where C is garbled circuit, X is garbled input, ξ_I is the necessary information needed by the corrupted parties to evaluate the circuit, $\mathcal{E}_I$ is the encoding information of corrupted parties, and d is the decoding information.

The simulation will create the garbled circuit with only one active (super)key per wire. We

choose superkeys randomly, except when the wire being output of an XOR gate. We will show that if there is a poly-time adversary who can win the experiment $\mathsf{Expt}^{\mathsf{priv}}_{G,\mathcal{A},\mathcal{S}}(k)$, then we can break the 2PRF game using $\mathcal{A}$ which leads to a contradiction to the fact that $\hat{F}$ is a PRF.

$\mathcal{S}(1^n, c, c(x), \mathcal{X}_I)$

$C = X = \Xi = E = d = \Phi$
for $i \in [1,n], w \in [1,u]$ **do**
  Choose active key $\mathcal{K}_{w,i} \in_R \{0,1\}^k$
  if *w owned by a corrupted party from I* **then**
    Choose random shares $\pi_{w,i} \in_R \{0,1\}$ and define $\pi_w := \pi_{w,1} \oplus \cdots \oplus \pi_{w,n}$
    Choose signal bit $\lambda_w = x_w \oplus \pi_w$
  else
    Choose signal bit $\lambda_w \in_R \{0,1\}$ and $\pi_{w,i} \in_R \{0,1\}$
  $\mathcal{E}_i = \{\mathcal{K}^{\lambda_w}_{w,i} = \mathcal{K}_{w,i}, \mathcal{K}^{\overline{\lambda}_w}_{w,i} \in_R \{0,1\}^k, \pi_{w,i}\}$;
  $E = \Xi \cup \mathcal{E}_i$
  $\mathcal{SK}_w = \mathcal{K}_{w,1} || \cdots || \mathcal{K}_{w,n}$
  $X = X \cup \{\mathcal{SK}_w\}$
end
In topological order, **for** $i \in [1,n], w \in [u+1, u+q]$ **do**
  Choose random $\mathcal{K}_{w,i}, \lambda_w$.
  $\Xi = \Xi \cup \xi_i = \{\mathcal{K}^{\lambda_w}_{w,i} = \mathcal{K}_{w,i}, \mathcal{K}^{\overline{\lambda}_w}_{w,i} \in_R \{0,1\}^k\}$
  $\mathcal{SK}_w = \mathcal{K}_{w,1} || \cdots || \mathcal{K}_{w,n}$
end
for $g \in [1,q]$ **do**
  Let a,b be input wires, and w be output wire.
  if *g is XOR* **then**
    $D_a := \hat{F}_{\mathcal{SK}_a}(g)$
    if $\lambda_b = 0$ **then**
      $D_b := \hat{F}_{\mathcal{SK}_b}(g)$
      $C_g \in_R \{0,1\}^{nk}$
    else
      $D_b \in_R \{0,1\}^{nk}$
      $C_g = \hat{F}_{\mathcal{SK}_b}(g) \oplus D_b$
    Update $\mathcal{SK}_w = D_a \oplus D_b$
  else
    $C_g[2 \times \lambda_a + \lambda_b] := \bigoplus_{i=1}^{n} F_{\mathcal{K}_{a,i}}(g||\lambda_a||\lambda_b) \oplus \bigoplus_{i=1}^{n} F_{\mathcal{K}_{b,i}}(g||\lambda_a||\lambda_b) \oplus \mathcal{SK}_w$
    $C_g[i] \in_R \{0,1\}^{nk}$,
    $\forall i \in \{0,1,2,3\} \setminus \{2 \times \lambda_a + \lambda_b\}$
end
for $w \in$ OutputWires **do**
  $d_w[c(x)_w] := \hat{F}_{\mathcal{SK}_w}(w||\lambda_w)$
  $d_w[\overline{c(x)}_w] \in_R \{0,1\}^{nk}$
end
$\xi_I := \bigcup_{i \in I} \xi_i$; $\mathcal{E}_I := \bigcup_{i \in I} \mathcal{E}_i$
return $C, X, \xi_I, \mathcal{E}_I, d$

Claim 3.4 *The garbling scheme G achieves privacy as for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ and a negligible function $\mu(\kappa)$ such that, $\forall\kappa \in \mathbb{N}$:*

$$\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{G},\mathcal{A},\mathcal{S}}(\kappa) = 1\right] \leq 1/2 + \mu(\kappa)$$

Proof: We prove this claim using the proof of contradiction. Therefore first we assume $\exists$ an adversary $\mathcal{A}$ which can win the privacy experiment of G. We introduce a number of special garbling schemes called $G^0, G^1, \cdots, G^q$ where G^0 resembles our garbling scheme G and G^q resembles completely simulated garbling by $\mathcal{S}$. G^1 through G^{q-1} represents the hybrid garbling schemes in between legitimate garbling and simulated garbling. Formally, G^i garble every gate exactly same like in G^{i-1} except the gate i. In G^{i-1}, the garbling of gate i would be legitimate garbling whereas that of G^i is simulated garbling. In each G^i, for each wire l, do the following:

- **If l is an input wire to g:**
  - **If $g \leq i$:** Choose the active key, and a random string and add to $\mathcal{E}_l$ or ξ_l, as per Simulator algorithm $\mathcal{S}()$.
  - **Else:** Choose both the keys and add to $\mathcal{E}_l$ or ξ_l, as per $\mathsf{Garble}()$ of BMR.
- **Else If l is an output wire of circuit, from gate g:**
  - **If $g \leq i$:** Prepare d_l according to $\mathcal{S}()$, where only one superkey can be decoded as the other entry is random.
  - **Else:** Prepare d_l according to BMR's $\mathsf{Garble}()$, where it contains decoding information of both superkeys.

We can easily claim that $\exists$ at least one adversary $\mathcal{A}^i$ who can distinguish G^i from G^{i-1} with non-negligible probability. Otherwise, if no adversary exists who can distinguish (G^i, G^{i-1}) for any i, then $\mathcal{A}$ can't distinguish G^0 from G^q also, which is not possible. So we assume

$$\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{Dist}(\mathsf{G}^{\mathsf{i}-1},\mathsf{G}^{\mathsf{i}})}_{\mathsf{Ch},\mathcal{A}^{\mathsf{i}}}(\kappa) = 1\right] \geq \frac{1}{2} + \frac{1}{p(\kappa)}$$

where $p(\kappa)$ is some polynomial in κ. In $\mathsf{Expt}^{Dist}_{G^{i-1},G^i,\mathcal{A}^i}(\kappa, 0)$, challenger garbles using G^{i-1} and in $\mathsf{Expt}^{Dist}_{G^{i-1},G^i,\mathcal{A}^i}(\kappa, 1)$, challenger garbles using G^i.

Now we construct an efficient adversary $\mathcal{A}^*$ for 2PRF game, using the responses of $\mathcal{A}_i$ by playing distinguishability experiment of (G_{i-1}, G_i) with him. This will contradict the fact that

2PRF is a hard problem, which is not true.

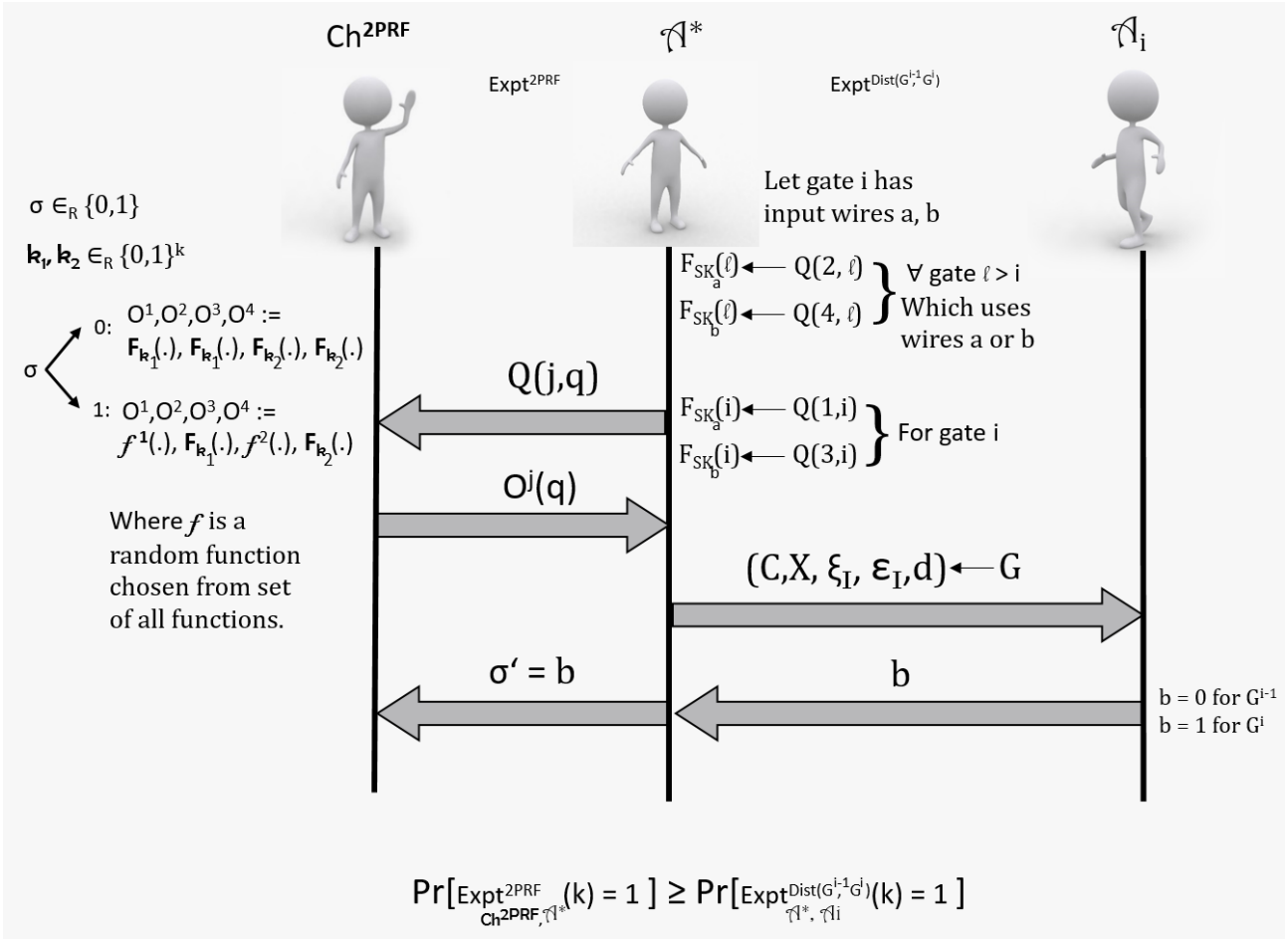

Figure 3.2: Reduction based proof of privacy

From the figure, we can claim that $\mathcal{A}*$ can win experiment 2PRF with non-negligible probability, as when $\sigma = 0$, $\mathcal{A}^*$ uses the garbling scheme G^{i-1} and when $\sigma = 1$, it is G^i. Please note that we take care of the other gates $l > i$ which uses the wires a and/or b. We make sure those gates are garbled legitimately by using the oracles $\mathcal{O}^1$ and $\mathcal{O}^3$, which are always PRF outputs. The same query will not be asked to both $(\mathcal{O}^1, \mathcal{O}^2)$, or to $(\mathcal{O}^3, \mathcal{O}^4)$ because of the unique gate id's. Therefore for all polynomial $p(\kappa)$,

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{Dist}(\mathsf{G}^{i-1},\mathsf{G}^i)}_{\mathcal{A}^*,\mathcal{A}^i}(\kappa) = 1\right] \leq & \mathsf{Pr}\left[\mathsf{Expt}^{\mathsf{2PRF}}_{\mathsf{Ch}^{\mathsf{2PRF}},\mathcal{A}^*}(\kappa) = 1\right] \\ & < \frac{1}{2} + \frac{1}{p(\kappa)}\end{aligned}$$

which contradicts our assumption of A^i. Hence privacy is proved.

■

For proving obliviousness, we can use the same hybrid argument with a small change in garbling. As the Simulator is not provided with the output of the circuit $c(x)$, $\mathcal{S}$ can't make legitimate decoding information d. However, as it is obliousness experiment, $\mathcal{S}$ doesn't need to construct d and will not be provided to $\mathcal{A}$. Therefore the proof goes same way, except the absence of d.

We can prove the authenticity of our scheme by a proof of contradiction. If there is a poly-time adversary $\mathcal{A}$ for $\mathsf{Expt}^{\mathsf{oblv}}_{G,\mathcal{A},\mathcal{S}}(\kappa)$ with non-negligible winning probability, that means it can produce a garbled output which doesn't decode to $c(x)$. Note that, we give completely random string to the d for the bits $\overline{c(x)}_j$ when we simulate the garbling. This implies $\mathcal{A}$ can distinguish some G^i from G^{i-1} by verifying if $\mathcal{A}$ can win obliviousness experiment. But we proved that such an adversary $\mathcal{A}$ doesn't exists so, no poly-time adversary can win obliviousness experiment unless with a negligible probability.

3.5 NOT gate with no ciphertext

This NOT gate optimization is a variant of Almost Free-XOR where we assume one arbitrary wire Z for the entire circuit, which has only one superkey $\mathcal{SK}_Z$, permutation bit $\pi_Z = 1$, and bit value $v_Z = 1$. This means the signal bit $\lambda_Z = 0$. We convert all the NOT gates to XOR gates with Z as the second input. Note that, $(v_w \oplus 1 = \overline{v}_w)$. As λ_z is always 1, we never need ciphertext for these gates. The security of this optimization reduces to the security of Almost Free-XOR.

Chapter 4

Conclusions and Future Work

We introduced two optimizations called *Cheap-XOR* and *Almost Free-XOR* to BMR's protocol, which enables cheap computation of XOR gates. This is proven (informally) to be secure against semi-honest adversary, even with a dishonest majority of $(n-1)$ out of n. We also introduced a generic distributed garbling scheme definition. Finally we optimized NOT gate, where garbling is possible without any ciphertexts. We look forward to apply other optimizations (like Garbled Row Reduction, FleXOR technique, etc.) to BMR protocol, which are currently known to only Yao's protocol.

Bibliography

[1] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM. ISBN 0-89791-361-2. doi: 10.1145/100216.100287. URL http://doi.acm.org/10.1145/100216.100287. 3, 5, 9, 28, 29, 30, 31

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 134–153, 2012. 2

[3] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012. 2, 23, 24

[4] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. Tinylego: An interactive garbling scheme for maliciously secure two-party computation. *IACR Cryptology ePrint Archive*, 2015:309, 2015. URL http://eprint.iacr.org/2015/309. 2

[5] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 191–219, 2015. 2

[6] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO*

2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings, pages 465–482, 2010. 2

[7] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: 10.1145/28395.28420. URL http://doi.acm.org/10.1145/28395.28420. 1, 3, 5

[8] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 567–578, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813619. URL http://doi.acm.org/10.1145/2810103.2813619. ii, 2, 3, 5, 16, 31, 33

[9] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In Luca Aceto, Ivan Damgrd, LeslieAnn Goldberg, MagnsM. Halldrsson, Anna Inglfsdttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-70582-6. doi: 10.1007/978-3-540-70583-3_40. URL http://dx.doi.org/10.1007/978-3-540-70583-3_40. 2, 5, 13, 18

[10] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44380-4. doi: 10.1007/978-3-662-44381-1_25. URL http://dx.doi.org/10.1007/978-3-662-44381-1_25. 2, 3, 4

[11] AndrewY. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, *Topics in Cryptology CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 117–132. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00861-0. doi: 10.1007/978-3-642-00862-7_8. URL http://dx.doi.org/10.1007/978-3-642-00862-7_8. 6

[12] Yehuda Lindell, Benny Pinkas, NigelP. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining bmr and spdz. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9216 of *Lecture Notes*

in Computer Science, pages 319–338. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-47999-5. doi: 10.1007/978-3-662-48000-7_16. URL http://dx.doi.org/10.1007/978-3-662-48000-7_16. 3

[13] Jesper Buus Nielsen and Samuel Ranellucci. Foundations of reactive garbling schemes. *IACR Cryptology ePrint Archive*, 2015:693, 2015. URL http://eprint.iacr.org/2015/693. 2

[14] Benny Pinkas, Thomas Schneider, NigelP. Smart, and StephenC. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-10365-0. doi: 10.1007/978-3-642-10366-7_15. URL http://dx.doi.org/10.1007/978-3-642-10366-7_15. 2, 4

[15] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society. doi: 10.1109/SFCS.1982.88. URL http://dx.doi.org/10.1109/SFCS.1982.88. 5

[16] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167, Oct 1986. doi: 10.1109/SFCS.1986.25. 1, 5