UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Lucas Piske

Shared OT and Efficient Protocols for Secure Integer Equality, Comparison and Bit-Decomposition

Belo Horizonte
2022

Lucas Piske

Shared OT and Efficient Protocols for Secure Integer Equality, Comparison and Bit-Decomposition

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Jeroen van de Graaf
Co-Advisor: Anderson C. A. Nascimento

Belo Horizonte
2022

Piske, Lucas.

P677s Shared OT and efficient protocols for secure integer equality, comparison and bit-decomposition [manuscrito] / Lucas Piske. — 2022.
60 f. il.

Orientador: Jeroen Antonius Maria Van de Graaf
Coorientador: Anderson Clayton Alves Nascimento
Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências 57-60.

1. Ciência da Computação – Teses. 2. Criptografia de dados (Computação) - Teses 3. Redes de computadores – Protocolos – Teses. I. Graaf, Jeroen Antonius Maria Van de. II. Nascimento, Anderson Clayton Alves. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*43(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz, CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Shared OT and Efficient Protocols for Secure Integer Equality, Comparison and Bit-Decomposition

LUCAS PISKE

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Jeroen Antonius Maria van de Graaf - Orientador
Departamento de Ciência da Computação - UFMG

Vinicius Fernandes dos Santos
Prof. Vinícius Fernandes dos Santos
Departamento de Ciência da Computação - UFMG

Anderson C. A. Nascimento
Prof. Anderson Clayton Alves Nascimento
Departamento de Engenharia Elétrica - UFMG

[illegible]
Profa. Ni Trieu
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University

Belo Horizonte, 19 de agosto de 2022.

Resumo

O desenvolvimento de protocolos criptográficos para a realização de computações conjuntas sobre entradas secretas entre múltiplas partes pode ser uma tarefa difícil, especialmente quando eficiência é uma qualidade necessária. Uma importante área de pesquisa que tende a usar tais protocolos é a de *privacy-presenving machine learning*, uma área que tem como objetivo o treinamento e a consulta de modelos de *machine learning* sem que seja necessário sacrificar a privacidade dos dados de usuários.

A fim de fazer com que a construção deste tipo de protocolos seja mais fácil, projetistas geralmente fazem uso de outros protocolos previamente propostos dentro do novo protocolo sendo construído. Isso faz com que a criação de protocolos eficientes que possam ser facilmente reusados em diferentes contextos seja uma tarefa importante, já que esses protocolos tem a possibilidade de ter um alto impacto ao serem utilizados em vários projetos.

Neste trabalho, propomos nos protocolos eficientes para três tarefas altamente pesquisadas e que são muito usadas na construção de novos protocolos criptográficos, sendo essas: teste de igualdade de elementos, comparação binária e decomposição em bits. Nós comparamos essas novas soluções a propostas previamente publicadas e mostramos que nossos protocolos se mostram favoráveis no quesito eficiência, especialmente no número de *rounds* que precisam ser realizados entre as partes envolvidas na execução do protocolo.

Ao definir os protocolos criados para as três tarefas especificadas, nós fazemos o uso de uma nova primitiva criptográfica também proposta neste trabalho, chamada de *Shared Oblivious Transfer* e abreviada como *Shared OT*. Essa nova primitiva pode ser interpretada como uma extensão da altamente conhecida primitiva criptográfica *Oblivious Transfer*. Mostramos que essa nova primitiva criptográfica pode ser reduzida a somente uma execução da primitiva original *Oblivious Transfer*, com muito pouca computação extra. Dada a eficiência e versatilidade desta nova primitiva, acreditamos que esta pode ser de interesse independente.

Palavras-chave: segurança incondicional, comparação segura, decomposição binária, teste de igualdade, transferência oblívia

Abstract

Developing cryptographic protocols for performing joint computations over several parties' secret inputs can be a difficult task, especially when an efficient solution is required. One important area of research that tends to require such protocols is privacy-preserving machine learning, an area that has as its focus training and querying machine learning models without having to sacrifice user data privacy.

In order to make constructing these types of protocols more manageable, designers usually use other previously created protocols as building blocks. This makes creating efficient protocols that can be reused in many different contexts an important effort, as they might have a lot of impact.

In this work, we propose new efficient protocols for three highly researched tasks used as building blocks in many different solutions: elementwise equality, bitwise comparison, and bit-decomposition. We compare these new solutions to previously published works and show that our protocols compare favorably efficiency-wise, especially in the number of communication rounds that need to be performed between parties.

When defining the protocols for these three tasks, we make use of a new cryptographic primitive that is also being proposed in this work, called Shared Oblivious Transfer, abbreviated as Shared OT. This new primitive can be seen as an extension of the widely known Oblious Transfer cryptographic primitive. We show that this new primitive can be reduced to a single execution of the original Oblivious Transfer primitive with very low overhead. Given its efficiency and versatility, we believe that Shared OTs can be of independent interest.

Keywords: unconditional security, secure comparison, bit-decomposition, equality test, oblivious transfer

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Secure two-party computation aims to enable a group of 2 parties, called Alice and Bob, to evaluate a function f over their secret inputs x_A, x_B without revealing anything beyond possibly the function output $f(x_A, x_B)$ in the presence of a party that may behave adversarially. This problem was introduced by Yao in his seminal paper [20], which proposed a method to compute any function f securely. Subsequently, other general constructions for implementing secure two-party computations and extensions to several parties appeared in the literature [2, 6, 8].

General solutions for secure two-party (and multiparty) computation are usually not efficient. Thus, the research community has focused on finding efficient methods to evaluate specific functions. This work proposes new efficient protocols for three highly researched functions: integer equality test, integer comparison, and bit-decomposition.

In the case of the integer equality test protocol, the two inputs x_A and x_B are obfuscated integers modulo $M \geq 2$ and the function f outputs a single obfuscated bit which indicates if $x_A = x_B$. In the case of the integer comparison protocol, the two inputs, x_A and x_B, are obfuscated bitwise representations of two integers a and b and the function f outputs a single obfuscated bit which indicates if $a < b$.

The bit-decomposition protocol input and output structure differs from the two previous protocols. While the other two took two inputs, this third protocol only takes a single obfuscated integer as input. Also, while the the other protocols output a single obsfuscated bit, this protocol outputs a sequence of obscuted bits that are the binary representation of the input integer.

After presenting and proving the security and correctness of all the new protocols proposed in this work we proceed to show their efficiency and compare them to other previously published works that were proposed for the same or similar settings. We are interested in comparing computational complexity, the number of bits exchanged between Alice and Bob during the protocol, and the required amount of rounds to complete the protocol.

As previously stated, secure two-party computation solutions must be resistant to parties acting adversarially. In this work, we assume that adversarial parties are unconditional, static, and semi-honest. Unconditional means that no assumptions are made about the computational power of the adversary, static means that the corrupt parties must be defined before starting executing the protocol and semi-honest means that the adversary follows the protocol but can try to discover extra information.

1.1 Motivation

Since 1982, when Yao introduced the secure two-party computation problem, multiple general solutions that allows for the two parties to evaluate any function f have been proposed throughout the years. This means that given any computable function f, we can choose one of these previously proposed solutions to run a protocol that securely computes f.

The ideas behind these schemes that securely evaluate arbitrary computable functions require reducing the computable functions to boolean or arithmetic circuits. This results in protocols that have poor efficiency and thus makes it so they have a small range of possible applicability in the 'real world'.

This forces protocol designers to look for different solutions from these generic ones when they are required to build efficient protocols for 'real world' problems. This, however, leads to an increase in the complexity of designing the necessary solution. This complexity can materialize itself in different ways, for example, in a higher difficulty writing security and correctness proofs for these protocols.

In order to deal with these efficiency and design complexity problems, the research community focused its efforts on finding efficient protocols for specific functionalities that are useful in many different scenarios and contexts. Some examples of such functionalities that have been highly researched are: dot product computation, set intersection, integer equality test, integer comparison test and bit-decomposition.

Protocols that need to execute these functionalities can make use of previously proposed effcient sub-protocols that implement them. This tends to decrease the complexity a designer needs to tackle when designing a new protocol. This is similar to the process of developing and publishing programming language libraries that can later be used in many different scenarios.

One such functionality that can be useful in many different case is the integer comparison test. This functionality takes two integers a and b as input and outputs a single bit which indicates if $a < b$. All of this without revealing any extra information about a and b other than the protocol's desired output.

Given that the integer comparison functionality is such a basic building block of computations, many different computations depend on it. This makes building an efficient secure protocol that can compute it really important, as improving its performance means improving the performance of many different protocols that depend on it. This is true for any other functionality that is as basic as this one, such as integer equality test and bit-composition, functionalities that also have new protocols proposed in this work.

A field where integer comparison shows itself really useful is privacy-preserving machine learning. The focus of this field is to 'teach' machines to perform certain tasks by generalizing data provided through learning algorithms without sacrificing data privacy in the process.

An interesting work in this field, which makes use of the integer comparison functionality, is the one by [3]. In this work, they propose a protocol for executing the k-means clustering algorithm, which allows two parties, each possessing a private data set, to compute a k-means clustering of their combined data. All of this without having to share their individual data points with one another.

Another interesting work in the field of privacy-preserving machine learning, that also performs integer comparison testing, is the one by [12], where Erkin proposed a recommendation system that can provide recommendations based on user encrypted data. This lets users receive relevant recommendations without having to relegate their privacy.

Based on the reasoning and examples presented in this section we believe that the importance of developing efficient protocols for functionalities, such as the ones that are the focus of this work, is evident. We also believe that this importance will keep growing, since more protocols that require these basic building blocks will probablly be proposed in the future.

1.2 Contributions

This work proposes novel solutions to privately evaluating the integer equality test, integer comparison, and bit-decomposition functions. The setting consists of two parties, Alice and Bob, and an adversary with unconditional computation power but behaves in a semi-honest manner. All the protocols presented in this paper to evaluate the previously enumerated functions achieve perfect security against such an adversary.

In order to construct the protocols proposed in this paper, we make use of a new primitive we call Shared Oblivious Transfer, a variation of the well-known Oblivious Transfer cryptographic primitive. The Oblivious Transfer primitive is explained in section 2.5. We show that we can implement shared oblivious transfer using one single instance of 1-out-of-N Oblivious Transfer over elements modulo M. This means that the efficiency of this new primitive is the same as the protocol chosen to implement the Oblivious Transfer, except for a negligible computational overhead.

The two new protocols for integer equality testing presented in this work take two additively shared ℓ-bit elements as input and output a shared bit, which indicates if the two elements are the same. What it means for an element to be additively shared is explained in section 2.3. We denote one of the equality protocols by $\Pi_{\texttt{EEQ}}$, and the other by $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$. Both protocols require 2 online rounds to be executed, but while $\Pi_{\texttt{EEQ}}$ has online computation and communication complexity equal to $O(\ell \log(\ell))$, protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$ has online computation and communication complexity equal to $O(\ell)$. However, protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$ requires one pre-processing round and the pre-processing phase has computation and communication complexity of $O(\ell \log(\ell))$, while $\Pi_{\texttt{EEQ}}$ does not require pre-processing.

We also present two new protocols for integer comparison in this work. Both protocols take the binary representation of two ℓ-bit elements, a and b, additively shared modulo 2 as input and outputs one additively shared bit, where this output bit indicates if $a < b$. We denote these two protocols by $\Pi_{\texttt{BLT}}$ and $\Pi^{\mathcal{P}}_{\texttt{BLT}}$. Both protocols require 3 online rounds to be performed, but while $\Pi_{\texttt{BLT}}$ has online computational and communication complexity equal to $O(\ell \log(\ell) \log(\log(\ell)))$, $\Pi^{\mathcal{P}}_{\texttt{BLT}}$ has online computational and communication complexity equal to $O(\ell \log(\ell))$. However, protocol $\Pi^{\mathcal{P}}_{\texttt{BLT}}$ requires one pre-processing round and the pre-processing phase has $O(\ell \log(\ell) \log(\log(\ell)))$ computational and communication complexity, while $\Pi_{\texttt{BLT}}$ does not require pre-processing.

Lastly, we present two protocols for the bit-decomposition function. Both protocols take a single additively shared ℓ-bit element β as input and outputs the binary representation of β additively shared modulo 2. We denote these two protocols by $\Pi_{\texttt{BD}}$ and $\Pi'_{\texttt{BD}}$. While protocol $\Pi_{\texttt{BD}}$ requires 2 overall rounds to be performed and has overall computational and communication complexity equal to $O(\ell^3)$, protocol $\Pi'_{\texttt{BD}}$ requires 3 overall rounds to be performed and has overall computational and communication complexity equal to $O(\ell^2 \log(\ell))$.

1.3 Outline

In this chapter, we briefly described the problems we focus on in this work, the solutions we were able to build, their main properties, and also discussed our motivations for researching this topic. The remaining chapters of this dissertation are organized in the following way:

Chapter 2 - Background. We start this chapter by presenting the Universally Composable framework, which is the theoretical framework we use to analyze and prove the security of all the protocols proposed in this work. We then proceed to describe the Commodity-Based Cryptography paradigm and introduce the cryptographic primitives Oblivious Transfer and Additive Secret Sharing. We finish this chapter by introducing notation and conventions deemed useful in making the new ideas expressed in this dissertation easier to comprehend.

Chapter 3 - Shared Oblivious Transfer and Applications. This chapter starts with a section that defines and proves the security for the new Shared Oblivious Transfer primitive. The next 4 sections of this chapter show how to construct private protocols for elementwise equality, bitwise comparison, and bit-decomposition using the Shared Oblivious Transfer primitive. This chapter ends with a section dedicated to comparing the efficiency of the protocols built using Shared OT to other previously published protocols that implement the same functionalities.

Chapter 4 - Conclusions. We present the conclusions reached through the development of this work.

Chapter 2

Backgroud

In this chapter, we explore the main ideas, concepts, and previous works upon which the results contained in this dissertation are based. We start the chapter by presenting notation and conventions used throughout this work. In section 2.2, we explore the Universally Composable Framework, the security framework we use to analyze and prove the security of the new protocols proposed in this work. In section 2.3, we explore what an Additive Secret Sharing scheme is and its properties. Next, we discuss the Commodity-Based Cryptography paradigm of developing protocols in section 2.4. We then conclude the chapter by delving into the important cryptographic primitive Oblivious Transfer in section 2.5.

2.1 Notation and Conventions

We now introduce notation, conventions and some important mathematical functions.

In all the protocols described in this work we output, and receive as input, additively shared elements modulo M, where $M \in \mathbb{Z}^{\geq 2}$. Let $a \in \mathbb{Z}_M$. We use $[\![a]\!]_M$ to denote the additive sharing modulo M of a by $[\![a]\!]_M$ and we use $[\![a]\!]_M^A \in \mathbb{Z}_M$ and $[\![a]\!]_M^B \in \mathbb{Z}_M$ to denote Alice's and Bob's respective shares of $[\![a]\!]_M$. Note that we only propose two-party protocols in this work and we always refer to the parties by Alice and Bob, so we only need notation to refer to these two parties shares.

The first convention we introduce is to only explicitly display an expression's modulo if it is not clear from the context. For example, if $a, b \in \mathbb{Z}_M$, then $a + b$, $a - b$, $a \cdot b$ are meant to be interpreted as $a + b \pmod{M}$, $a - b \pmod{M}$ and $a \cdot b \pmod{M}$, respectively. Another convention is how we index vectors and binary expansions. Let $a \in \mathbb{Z}_M$ and $\vec{a} \in \mathbb{Z}_2^{\lceil \log_2(N) \rceil}$, where $\vec{a}$ is the binary expasion of a. We index the vectors in this paper starting from 0 and the LSB of a binary expansion $\vec{a}$ is $\vec{a}_0$. Another convention is the meaning of adding a scalar modulo M to a vector of elements modulo M. Let $a \in \mathbb{Z}_M$

and $\vec{v} \in \mathbb{Z}_M^N$, where $N \in \mathbb{Z}^{\geq 1}$. In this paper, when we write $\vec{v}' = \vec{v} + u$, we mean that $\vec{v}'_i = \vec{v}_i + u \pmod{M}$, for $i \in \{0, 1, \ldots, N-1\}$.

One of the first two functions we need to define is $\mathsf{One}_n(i, a)$, where $0 \leq i \leq n-1$ and $0 \leq a \leq n - i$. This function outputs a vector $\vec{v} \in \mathbb{Z}_2^n$ containing a 1s, starting from position i, and containing 0s in all the remaining positions. For example, $\mathsf{One}_5(2, 3) = (0, 0, 1, 1, 1)$ and $\mathsf{One}_4(0, 1) = (1, 0, 0, 0)$. Formally, if $\vec{v} = \mathsf{One}_N(i, a)$, then

$$\vec{v}_j = \begin{cases} 1, & \text{if } i \leq j < i + a \\ 0, & \text{otherwise} \end{cases}, \text{ for } j \in \{0, 1, \ldots, N-1\}$$

The second function is $\mathsf{cshift}_N(\vec{v}, x)$, where $\vec{v} \in \mathbb{Z}^N$ and $x \in \mathbb{Z}_N$. The function cshift outputs a vector $\vec{v}'$, where $\vec{v}'$ is the vector $\vec{v}$ with its values shifted x positions, from position 0 towards position $N-1$. For example, if $\vec{v} = (1, 2, 3, 4)$, then $\mathsf{cshift}_4(\vec{v}, 2) = (3, 4, 1, 2)$. Formally, if $\vec{v}' = \mathsf{cshift}(\vec{v}, x)$, then $\vec{v}'_i = \vec{v}_{i-x \pmod{N}}$, for $i \in \{0, 1, \ldots, N-1\}$.

2.2 Universally Composable Framework

When proposing new cryptographic protocols it is important to provide a rigorous security proof that attests to its security claims, otherwise, any security property stated by its designers is simply a wish. To write such security proofs, we must first define a model and then define a security definition for the cryptographic task in question. It is only then that we can write a proof showing that a protocol fulfills the security definition.

This makes elaborating a model and a security definition important, but doing so is a non-trivial task. Mistakes made when defining a model or security can lead to security issues in protocols that have their security proven using these. For example, if a model does not consider certain properties that an adversary can have, this restricts the cases in which a protocol is secure from the start. Thus, when proposing a definition, we must strive to cover any adversarial behavior, which again, is not an easy task.

Another important aspect that should be taken in consideration is that a protocol should maintain security not only when in run by it self but should also do so when is part of another larger system, as is usually the case in ‘real world’ scenarios.

In this work, we make use of the model and definitions that form the Universally Composable framework, first presented in [5]. The Universally Composable framework, sometimes abbreviated as UC framework, is considered by a large part of the cryptographic community as the current strongest security model a protocol can fulfill and can be used to show that a protocol maintains security, even when composed with other protocols, or copies of itself. The rest of this chapter focuses on exploring the main ideas involved in the UC framework, starting with an intuitive overview.

In the UC framework, we assume the existence of an ideal world which contains a magic black-box that securely provides the functionality f to two parties P_A and P_B. More specifically, each party provides their secret inputs to the box without revealing the inputs to the other party, the box performs internal hidden computations and then delivers to the parties their respective outputs. In the UC framework, we call such a world the Ideal-world and we call such a magic box the ideal functionality $\mathcal{F}$, where $\mathcal{F}$ exists in the Ideal-world.

Also in the UC framework, there exists another world separate from the Ideal-world, called Real-world. In the Real-world, like the real world, there is no mechanism magic box. In the Real-world, the two parties P_A and P_B must perform a protocol Π in order to compute the functionality f. Since no magic box exists, an adversary might have the ability to interfere in the protocol's execution and cause some damage, like leaking some information not intended to be leaked. This adversary that exists in the Real-world is denoted by $\mathcal{A}$.

Intuitively, a protocol is secure in the UC framework, also called UC secure, if the protocol Π behaves in a way that makes the Real-world be an emulation of the Ideal-world. But to better understand what it means to say that a world emulates another, we must further understand the two worlds in question. In the next two sections we better detail how these worlds are composed, their differences and similarities, and then better define emulation.

2.2.1 Real-world

The Real-world contains the following 3 components: Two parties P_A and P_B, the adversary $\mathcal{A}$ and the environment $\mathcal{E}$, to which the two parties are connected by a communication medium that they can use to exchange messages.

The goal of the two parties P_A and P_B is to compute the functionality f by performing Π. They do this by following their respective instructions contained in the description for Π. These instructions may require them to perform internal computations or send messages through the communication medium they have access to. Some protocols may require a subset of parties to provide inputs, which in this case, the inputs used by these parties will be provided by $\mathcal{E}$. The execution of the protocol may also result in outputs for a subset of the parties involved, which in this case, these parties will deliver their outputs to $\mathcal{E}$.

The adversary $\mathcal{A}$ exists with the main purpose of modeling the corruption of parties in the Real-world. For this purpose, the adversary can corrupt a subset of parties, arbitrarily drop and delay messages exchanged between the two parties, and communicate with the environment $\mathcal{E}$ as it wishes. When an adversary is able to corrupt a party and what it can do to that party once it corrupts it, varies depending on the adversarial model considered.

If we suppose that an adversary can only corrupt a party at the beginning of the protocol's execution, we say that our adversarial model assumes static corruption. But if the adversary can corrupt parties at any time, we say that our adversarial model assumes adaptative corruption. In this work, we only consider adversarial models that assume static corruption.

Another important aspect of the adversarial model is how the adversary can influence parties once they are corrupted. We say that an adversary is semi-honest if it only has the ability to read the internal state of corrupted parties, and we say that an adversary is malicious if it can also cause the corrupted party to send arbitrary messages. In this work, we have proofs that consider only semi-honest adversaries and proofs that also consider the malicious case.

The last component of the Real-world is the environment $\mathcal{E}$. The two main goals of $\mathcal{E}$ is to model everything external to the execution of the protocol Π and to output a distinguisher bit. In order to achieve these goals, $\mathcal{E}$ can communicate with $\mathcal{A}$, provides the inputs that it whishes for P_A and P_B to use when executing Π and receives the outputs that P_A and P_B may produce as the result of executing Π.

The environment's distinguisher bit plays an integral role in the formalization of the emulation that must be achieved between the Real-world and Ideal-world. This bit's role will be clearer after section 2.2.3, where the formalization of world emulation is discussed further. We denote the distribution ensemble that describes the environment's output bit, when protocol Π is run with adversary $\mathcal{A}$ in the Real-world, by $\text{REAL}_{\Pi,\mathcal{A},\mathcal{E}}$.

2.2.2 Ideal-world

The components of the Ideal-world are really similar to the ones present in the Real-world. In the Ideal-world we have the following 4 components: two dummy parties P_A and P_B, the ideal adversary called simulator $\mathcal{S}$, the ideal functionality $\mathcal{F}$ and the environment $\mathcal{E}$.

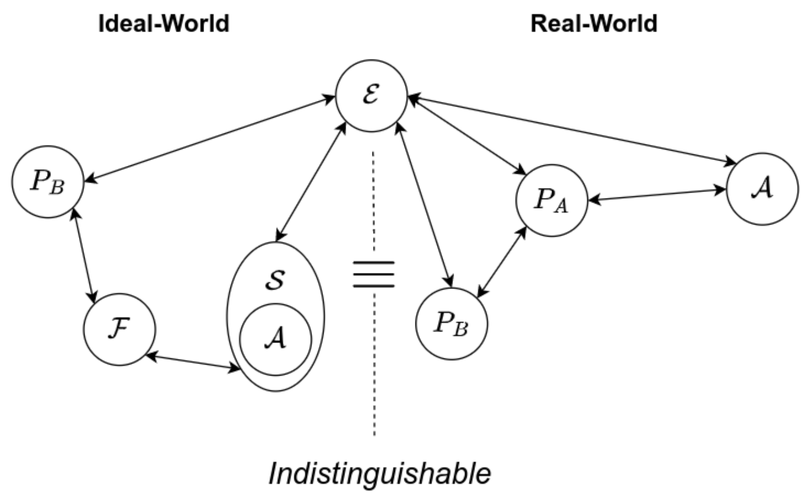
The ideal functionality $\mathcal{F}$ can be seen as a magical black-box that securely provides the functionality f to the parties that use it. The parties that use $\mathcal{F}$ can provide their secret inputs to $\mathcal{F}$ without revealing them to other parties and can also receive their respective output from $\mathcal{F}$ without having them revealead to others. Any computation necessary to provide the functionality f is done internally and in secret by $\mathcal{F}$.

As in the Real-world, the parties P_A and P_B also have the goal of executing the functionality f in the Ideal-world. But in the Ideal-world, the parties do not follow the set instructions contained in the description of Π, they simply provide their inputs to $\mathcal{F}$ and wait for their respective output. This simpler behaviour is the reason they are called dummy parties. Also like in the Real-world, the inputs used by the parties are chosen by the environment and the parties relay the output they receive from $\mathcal{F}$ to the environment.

The ideal adversary $\mathcal{S}$, usually called simulator, has a different role compared to its Real-world counterpart. The simulator exists in the Ideal-world for the purpose of simulating the impact that $\mathcal{A}$ can have over the protocol Π while only having access to the ideal functionality $\mathcal{F}$. The existence of such a simulator means that the same impact an adversary $\mathcal{A}$ has over Π in the Real-world can also be achieved in the Ideal-world against $\mathcal{F}$.

Similar to the Real-world, the environment $\mathcal{E}$ exists in the Ideal-world with the objective of modeling everything external to the protocol execution of $\mathcal{F}$ and to output a distinguisher bit. For this purpose, the environment can communicate with $\mathcal{S}$, it chooses the inputs that P_A and P_B provide to $\mathcal{F}$ and receives the outputs of $\mathcal{F}$. We denote the distribution ensemble that describes the environment's output bit, when ideal functionality $\mathcal{F}$ is run with simulator $\mathcal{S}$ in the Ideal-world, by $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}$.

Figure 2.1: Diagram of the Ideal-World and Real-World executions for a corrupted party P_A.



Source: Created by author.

2.2.3 Formalizing the Security Definition

Now that we have explored the details of protocol executions in the ideal and real worlds, we can go back to rigorously proving the security of a protocol Π. As already intuitively explained, a protocol Π UC securely provides a functionality f if it emulates an ideal functionality $\mathcal{F}$ that provides the functionality f.

In the UC framework, a protocol Π emulates an ideal functionality $\mathcal{F}$ if no environment $\mathcal{E}$ can tell them both apart. In other words, if no environment can tell if it is currently a part of the real world or a part of the ideal world, then Π emulates $\mathcal{F}$. Note that the environment's only way to distinguish between the two worlds is by interacting with the other components as described in previous sections.

Since an environment $\mathcal{E}$ can interact with an adversary $\mathcal{A}$ in the Real-world, the existence of an $\mathcal{E}$ that can distinguish between the two possible worlds depends on a simulator $\mathcal{S}$ being able to simulate the interaction that happens between $\mathcal{E}$ and $\mathcal{A}$. It turns out that a $\mathcal{S}$ that can simulate the interactions between $\mathcal{A}$ and $\mathcal{E}$ in the Real-world may not exist.

This means that a scheme for constructing a simulator $\mathcal{S}$ for each $\mathcal{A}$ that simulates all necessary interactions that happen in the Real-world must be shown as part of the security proof of a protocol Π. Having understood the different components involved in proving the security of a protocol Π in the Universally Security framework, next we present the formal definition of UC security.

Definition 1 (Perfect UC-Security). *Let Π be a protocol and $\mathcal{F}$ be the ideal functionality to be simulated by Π. The protocol Π perfectly UC-realizes $\mathcal{F}$ if for every Real-world adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that, for all environments $\mathcal{E}$, the distribution ensembles $REAL_{\Pi,\mathcal{A},\mathcal{E}}$ and $IDEAL_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are the same.*

Notice the "perfect" qualifier in the definition's name. This qualifier points to the requirement that both distribution ensembles must be identical, which might not be the case in some variants of UC-Security definitions. By proving the security of a protocol according to this definition we do not make any assumptions about an adversary's computational power, which is what we do when proving the security of protocols proposed in this work.

2.2.4 Hybrid-world

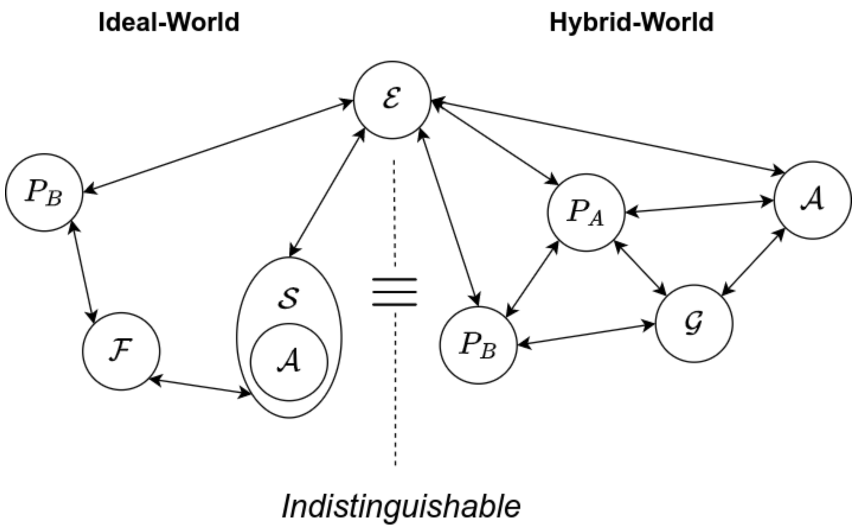
Proving the security of more complex and detailed protocols in the UC framework can lead to more intricate and complicated security proofs, which can get much harder to write and verify. A natural way to tackle this task is by making more modular constructions and analyzing these constructions piece by piece. The UC framework helps us in this approach by providing important theoretical tools, which revolve around a third world called Hybrid-world.

The Hybrid-world is really similar to the previously presented Real-world. It contains all the components contained in the Real-world and also allows all the interactions that are allowed in the Real-world, but it also contains an extra component, which is an ideal functionality $\mathcal{G}$.

In this third world, the two parties involved not only have access to a communication medium with which they can communicate by exchanging messages but they can also interact through the use of ideal functionality $\mathcal{G}$. But the parties are not the components that can interact with $\mathcal{G}$, the adversary $\mathcal{A}$ also is allowed to do so. The adversary can corrupt this functionality, which will behave as described in the Ideal-world when this happens.

Now, let $\Pi_{\mathcal{G}}$ be a protocol that perfectly UC-realizes $\mathcal{G}$ in the Real-world and $\Pi^{\mathcal{G}}$ be a protocol that replaces every call to $\mathcal{G}$ by a call to $\Pi_{\mathcal{G}}$. Based on the Universal Composability theorem, presented in [4], we know that if Π perfectly UC-realizes an ideal functionality $\mathcal{F}$ in the Hybrid-world, then $\Pi^{\mathcal{G}}$ perfectly UC-realizes $\mathcal{F}$ in the Real-world. This means that by analysing and proving the security in a modular way, through the Hybrid-world, we know that the protocol will also be secure in the Real-world.

Figure 2.2: Diagram of the Ideal-World and Hybrid-World executions for a corrupted party P_A and auxiliary functionality $\mathcal{G}$.



Source: Created by author.

2.3 Additive Secret Sharing

Intuitively speaking, a perfectly private t-out-of-n secret sharing scheme is a method that lets a dealer D split a secret s into n pieces in a way that makes it possible to reconstruct s from $t \leq n$ pieces while not allowing any information about s to be computed from less than t pieces. These pieces can then be distributed to a set of parties which can use t pieces to reconstruct s without interacting with D. When discussing secret sharing schemes, the pieces of s are usually called shares of s.

Formally, we can define a perfectly private t-out-of-n secret sharing scheme in the following way:

Definition 2 (Perfectly private t-out-of-n secret sharing scheme)**.** *A t-out-of-n secret sharing scheme over message space $\mathcal{M}$ is a pair of algorithms (*Share*,* Reconstruct*) such that*

- Share*: Is a probabilistic polynomial-time algorithm that takes any message $m \in \mathcal{M}$ as input and outputs n shares $(s_0, s_1, \ldots, s_{n-1})$.*
- Reconstruct*: Is a deterministic polynomial-time algorithm that takes t secret shares and outputs a message $m \in \mathcal{M}$.*

While satisfying the following correctness and perfect privacy requirements:

- *Correctness:* $\forall m \in \mathcal{M}, \forall S = \{i_0, i_1, \ldots, i_{t-1}\} \subseteq \{0, 1, \ldots, n-1\}$ *where* $|S| = t$*:*

$$\underset{(s_0,\ldots,s_{n-1})\leftarrow\mathsf{Share}(m)}{Pr}[\mathsf{Reconstruct}(s_{i_0}, \ldots, s_{i_{t-1}}) = m] = 1$$

- *Perfect Privacy:* $\forall m, m' \in \mathcal{M}, \forall S \subseteq \{0, \ldots, n\}$ *where* $|S| < t$*, the following distributions are identical:*

$$\{(s_i | i \in S) \colon (s_0, \ldots, s_{n-1}) \leftarrow \mathsf{Share}(m)\}$$

$$\{(s'_i | i \in S) \colon (s'_0, \ldots, s'_{n-1}) \leftarrow \mathsf{Share}(m')\}$$

In the work contained in this dissertation, we only propose new perfectly secure protocols executed by two parties. Therefore, we are restricted to only using perfectly private 2-out-of-2 secret sharing schemes when developing these protocols. More specifically, we only make use of the 2-out-of-2 additive secret sharing scheme. Next, we formally describe the 2-out-of-2 additive secret sharing scheme:

Definition 3 (2-out-of-2 additive secret sharing scheme)**.** *Let* $M \geq 2$ *and the message space* $\mathcal{M}$ *be* $\mathbb{Z}_M$*. The algorithms* Share *and* Reconstruct *are defined in the following way for this scheme:*

- Share*: First, the dealer* D *samples an element* $[\![s]\!]^A_M \in_R \mathbb{Z}_M$ *modulo* M*. After sampling* $[\![s]\!]^A_M$*, the dealer then computes the element* $[\![s]\!]^B_M = s + [\![s]\!]^A_M \pmod M$ *modulo* M*. After that, the dealer outputs the two shares* $([\![s]\!]^A_M, [\![s]\!]^B_M)$*, with* $[\![s]\!]^A_M$ *intended to be Alice's share of* s *and* $[\![s]\!]^B_M$ *intended to be Bob's share of* s*.*

- Reconstruct*: Let* $[\![s]\!]^A_M \in \mathbb{Z}_M$ *and* $[\![s]\!]^B_M \in \mathbb{Z}_M$ *be Alice's and Bob's repective shares of* s*. A party can reconstruct* s *by computing* $s = [\![s]\!]^B_M - [\![s]\!]^A_M \pmod M$*.*

It is easy to see from definition 3 that the 2-out-of-2 additive secret sharing scheme is a type of perfectly private t-out-of-n secret sharing scheme, as its two algorithms respect the descriptions given in definition 2 and that they also fulfill the correctness and perfect privacy properties described in definition 2.

Another important fact about the 2-out-of-2 additive secret sharing scheme is that the shares can not only be used to reconstruct the original secrets but can also be used to compute shares of related secrets. This type of operation over secret shares is used extensively in the Multi-party computation literature and is also used extensively in the work presented in this dissertation. More precisely, we make use of three such operations, where all of the three do not require any interaction between the two parties to be performed.

Suppose that Alice and Bob each have a share of two secrets $s_0, s_1 \in \mathbb{Z}_M$. In other words, Alice has $[\![s_0]\!]^A_M, [\![s_1]\!]^A_M$ and Bob has $[\![s_0]\!]^B_M, [\![s_1]\!]^B_M$. The first of the three operations over secret shares allows Alice and Bob to compute $[\![s']\!]^A_M$ and $[\![s']\!]^B_M$, respectively, where

$s' = c + s_0 \pmod{M}$ and $c \in \mathbb{Z}_M$ is a publicly known constant. The second operation allows for Alice and Bob to compute $[\![s']\!]_M^A$ and $[\![s']\!]_M^B$, respectively, where $s' = c \cdot s_0$ (mod M) and $c \in \mathbb{Z}_M$ is also a publicly known constant. The last of the three operations makes it possible for Alice and Bob to compute the shares $[\![s']\!]_M^A$ and $[\![s']\!]_M^B$, respectively, where $s' = s_0 + s_1 \pmod{M}$.

2.4 Commodity-Based Cryptography

First proposed by Beaver in [1], Commodity-Based Cryptography is a paradigm for designing efficient secure multi-party computation protocols. In this paradigm we have a set of servers and a set of clients, where the servers exist for the purpose of aiding the clients in executing cryptographyic primitives. Some type of corruption of a subset of servers may be admissable, but this may vary between two protocols. The same is true about the set of clients.

The Commodity-Based paradigm not only defines this set of players, but also restrains what information these players have about each other and how they interact. This is what separates this paradigm from other client-server models. First, a server should not have any information about any other server, including whether other servers exist or not. Second, any server-client pair must interact in a request-response manner where the client sends the request. Third, any response sent to the client must be independent of the client's input and of any previous communication between the client and the server.

By having this restrictions in place, the paradigm presents some advantageous properties. There is no need for the client to provide sensitive data to the server, which minimizes the trust that clients need to have on servers. At same time, the paradigm is scalable since many servers can be employed at the same time. Having many servers also helps to increase confidence that at least a fraction of the material provided by the servers is secure and correct.

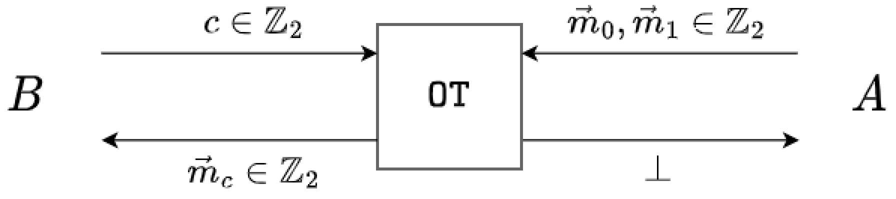
Since first introducted, many protocols have been proposed in this paradigm, where the first two were defined in the paper that defined the paradigm. In [1], two protocols for $\binom{1}{2}$-OT are apresented. One protocol that is perfectly secure against honest-but-curious adversaries and another one that is statistically secure against malicious adversaries. Later in [18], Rivest proposed a different protocol for $\binom{1}{2}$-OT that is perfectly secure against malicious adversaries, along with a protocol for bit commitment. Protocols for secret sharing, such as the one contained in [14], have also been proposed, among other primitives.

2.5 Oblivious Transfer

Ever since first proposed by Michael O. Rabin, many interesting and useful variants of the Oblivious Transfer primitive have been constructed and even more uses have been found for them. The most commonly used version of this primitive is called 1-out-of-2 Oblivious Transfer of bits. This version of the primitive is so common that researches usually refer to it as Oblivious Transfer only.

In a 1-out-of-2 Oblivious Transfer of bits, one party, usually, Alice, has two secret bits $\vec{m}_0, \vec{m}_1 \in \mathbb{Z}_2$ and she wishes to let another party, usually, Bob, learn the value of one of the two bits without herself learning which bit Bob chose to learn the value of. In other words, when executing this primitive, Alice inputs two secret bits $\vec{m}_0, \vec{m}_1 \in \mathbb{Z}_2$ and Bob inputs one secret bit $\vec{c} \in \mathbb{Z}_2$. After finishing executing this primitive, Bob will end up with $\vec{m}_c$, and Alice will not learn any new information. Next, we have a diagram showing the input and output structures of this primitive.

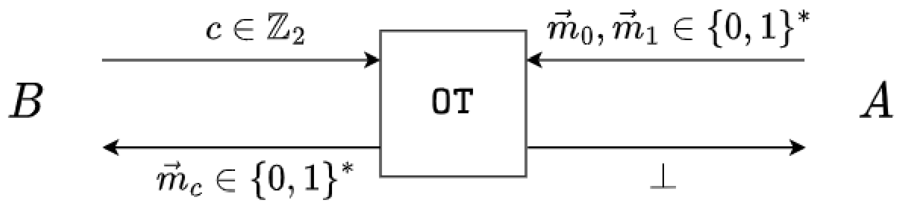
Figure 2.3: Input and output structures of 1-out-of-2 Oblivious Transfer of bits.


$c \in \mathbb{Z}_2$
$\vec{m}_0, \vec{m}_1 \in \mathbb{Z}_2$
B
OT
A
$\vec{m}_c \in \mathbb{Z}_2$
$\perp$

Source: Created by author.

Probably the next most commonly used variant of Oblivious Transfer is the 1-out-of-2 OT of strings. As the names suggests, this other variant is really similar to the previous version of OT described, with the only difference being that in this case $\vec{m}_0$ and $\vec{m}_1$ are strings of bits, instead of single bits. Next we have a diagram showing the input and output structures of this OT variant.

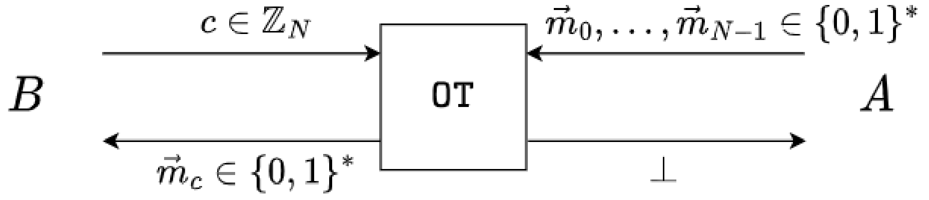
Figure 2.4: Input and output structures of 1-out-of-2 Oblivious Transfer of bit strings.


$c \in \mathbb{Z}_2$
$\vec{m}_0, \vec{m}_1 \in \{0,1\}^*$
B
OT
A
$\vec{m}_c \in \{0,1\}^*$
$\perp$

Source: Created by author.

Another common generalization of Oblivious Transfer is the 1-out-of-N Oblivious Transfer of strings. Like the previously described variant, the elements of $\vec{m}$ are strings of bits, but the different is that the length of vector $\vec{m}$ is now N instead of 2. Next we have a diagram showing the input and output structures of this flavor of Oblivious Transfer.

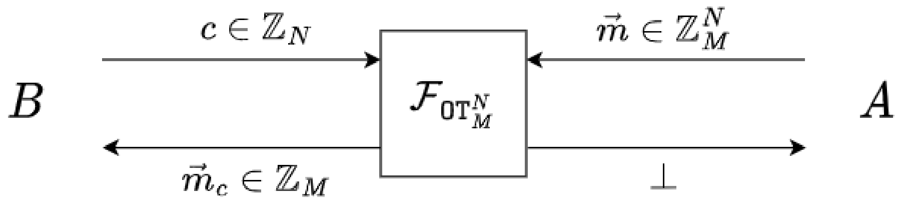
Figure 2.5: Input and output structures of 1-out-of-N Oblivious Transfer of bit strings.


$c \in \mathbb{Z}_N$
$\vec{m}_0, \ldots, \vec{m}_{N-1} \in \{0,1\}^*$
B
OT
A
$\vec{m}_c \in \{0,1\}^*$
$\perp$

Source: Created by author.

In this dissertation we are only concerned with one single type of OT, the 1-out-of-N Obvlious Transfer of elements modulo M. Like the last described OT, the vector $\vec{m}$ has size N, but now the elements of $\vec{m}$ are all elements modulo M instead of bit strings. Next we have a diagram showing the input and output structures of this OT variant and the formal definition of this OT's functionality.

Figure 2.6: Input and output structures of 1-out-of-N Oblivious Transfer of elements modulo M.


$c \in \mathbb{Z}_N$
$\vec{m} \in \mathbb{Z}_M^N$
B
$\mathcal{F}_{\mathtt{OT}_M^N}$
A
$\vec{m}_c \in \mathbb{Z}_M$
$\perp$

Source: Created by author.

Functionality $\mathcal{F}_{\mathtt{OT}_M^N}$

- Upon receiving a message (`choose`, c) from Bob: Ignore any subsequent (`choose`, c) messages. If $c \notin \mathbb{Z}_N$, then send (`invalid input`) to both parties and halt. Otherwise, store c internally and send the public delayed message (`chosen`) to Alice.

- Upon receiving a message (`propose`, $\vec{m}$) from Alice: Ignore any subsequent (`propose`, $\vec{m}$) messages. If it isn't the case that $\vec{m} \in \mathbb{Z}_M^N$ and c is currently internally stored, send (`invalid input`) to both parties and halt. Otherwise, send $\vec{m}_c$ to Bob.

By generalizing the Oblivious Transfer protocol proposed by Rivest in [18] in a straightforward manner, it is possible to implement a protocol that fulfills the description for $\mathcal{F}_{\mathtt{OT}^N_M}$ while providing perfect security in the malicious setting, in the Commodity-Based paradigm. This protocol can be performed in one single round and the amount of bits transferred between the two parties and the computation required to be performed by the two parties are both equal to $O(\log(N) + N \cdot \log(M))$. The amount of commodity data in bits required and the amount of computation required to generate this data are both equal to $O(\log(N) + N \cdot \log(M))$.

Chapter 3

Shared Oblivious Transfer and Applications

We start this chapter with a section that presents the new Shared Oblivious Transfer cryptographic primitive. In this section, we first give an intuitive explanation of its functionality and then proceed to formally define it. With Shared OT formally defined, we go on to prove its security and correctness against a static malicious adversary with unconditional computational power.

The next 4 sections are dedicated to constructing protocols for different tasks, one section for each task. These sections have a similar structure to the first one. In each section, we start by giving an intuitive explanation of the task at hand, then proceed to formally define it and end the section by proving the security and correctness of the protocol against a static semi-honest adversary with unconditional computational power.

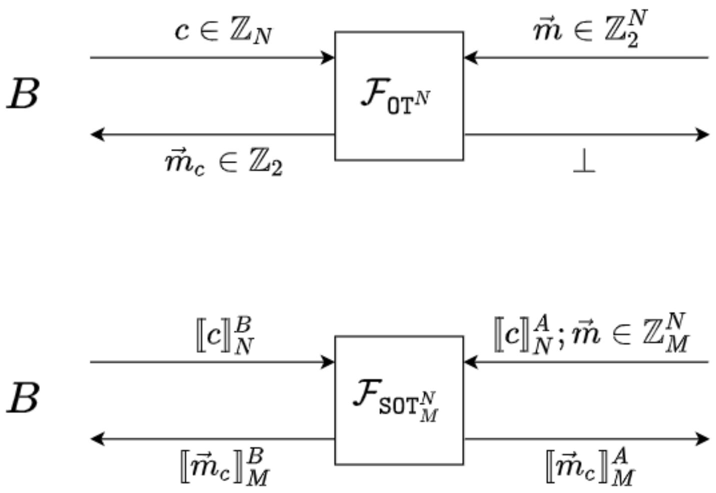
This chapter ends with a section that compares the efficiency of the proposed protocols for elementwise equality, bitwise comparison, and bit-decomposition to similar previously published protocols.

3.1 Shared Oblivious Transfer

We now introduce a simple, new variant of oblivious transfer. Our new primitive Shared Oblivious Transfer is a simple extension of 1-out-of-N Oblivious Transfer over elements modulo M. Our extension differs from OT in two significant ways: (1) the selection index input is additively shared by the parties, Alice and Bob; and (2) the output is also additively secret shared between the parties.

Figure 3.1 shows the difference between the input and output structures of the functionalities 1-out-of-N Bit OT and the 1-out-of-N SOT over elements modulo M. In Shared OT, Alice inputs the options vector $\vec{m}$, containing N elements modulo M, and Alice and Bob input their respective shares of an index c modulo N. Likewise, SOT outputs additive shares modulo M of $\vec{m}_c$ to Alice and Bob, and neither party gains any information to which it is not entitled. Note that the options vector $\vec{m}$ is not shared by both parties; only Alice knows its value.

Figure 3.1: Difference between input and output structure of 1-out-of-N binary OT and 1-out-of-N SOT over elements modulo M.


$c \in \mathbb{Z}_N$
$\vec{m} \in \mathbb{Z}_2^N$
B
$\mathcal{F}_{\mathtt{OT}^N}$
A
$\vec{m}_c \in \mathbb{Z}_2$
$\perp$
$[\![c]\!]_N^B$
$[\![c]\!]_N^A; \vec{m} \in \mathbb{Z}_M^N$
B
$\mathcal{F}_{\mathtt{SOT}_M^N}$
A
$[\![\vec{m}_c]\!]_M^B$
$[\![\vec{m}_c]\!]_M^A$

Source: Created by author.

We formally define the functionality of SOT in the following way:

Functionality $\mathcal{F}_{\mathtt{SOT}_M^N}$

- Upon receiving a message $(\mathtt{choose}, [\![c]\!]_N^B)$ from Bob: Ignore any subsequent $(\mathtt{choose}, [\![c]\!]_N^B)$ messages. If $[\![c]\!]_N^B \notin \mathbb{Z}_N$, then send $(\mathtt{invalid\ input})$ to both parties and halt. Store $[\![c]\!]_N^B$ and send the public delayed message $(\mathtt{chosen})$ to Alice.

- Upon receiving a message $(\mathtt{sample\ share})$ from Alice: Ignore any subsequent messages $(\mathtt{sample\ share})$. Sample $[\![\vec{m}_c]\!]_M^A \in_R \mathbb{Z}_M$, store it internally and send it to Alice.

- Upon receiving a message $(\mathtt{propose}, [\![c]\!]_N^A, \vec{m})$ from Alice: Ignore any subsequent $(\mathtt{propose}, [\![c]\!]_N^A, \vec{m})$ messages. If it is not the case that $\vec{m} \in \mathbb{Z}_M^N$, $[\![c]\!]_N^A \in \mathbb{Z}_N$ and $[\![\vec{m}_c]\!]_M^A$ is currently stored, send $(\mathtt{invalid\ input})$ to both parties and halt. If it is the case, send $[\![\vec{m}_c]\!]_M^B = \vec{m}_c + [\![\vec{m}_c]\!]_M^A \pmod M$ to Bob.

We are able to build a protocol, which we denote by $\Pi_{\mathtt{SOT}}$, that implements this functionality using a single instance of $\mathcal{F}_{\mathtt{OT}_M^N}$ and only performs elementary local operations (such as cyclic shift of the vector m, sampling and addition modulo some integer) over the protocol's inputs. In order to perform this protocol, Alice and Bob execute an instance of $\mathcal{F}_{\mathtt{OT}^N_M}$ with Bob providing $[\![c]\!]^B_N$ as the choice index and Alice providing $\vec{m}'$ as the options vector, where $\vec{m}' = \mathsf{cshift}_N(\vec{m}, [\![c]\!]^A_N) + u$ and $u \in_R \mathbb{Z}_M$ is sampled by Alice. As a result of the execution of $\mathcal{F}_{\mathtt{OT}^N_M}$, Bob ends up with $\vec{m'}_{[\![c]\!]^B_N}$ and Alice ends up with u, since she sampled it. These two values will be their respective outputs for $\Pi_{\mathtt{SOT}}$.

From this brief description, we can explain the main argument behind the correctness and security of $\Pi_{\mathtt{SOT}}$. Regarding correctness: Given the inputs provided to $\mathcal{F}_{\mathtt{OT}}$, during the execution of $\Pi_{\mathtt{SOT}}$, Bob will receive $\vec{m'}_{[\![c]\!]^B_N}$, which, based on the definition of cshift_N and how $\vec{m}'$ is built, implies Bob receives $\vec{m'}_{[\![c]\!]^B_N} = \vec{m}_{[\![c]\!]^B_N - [\![c]\!]^A_N \pmod N} + u = \vec{m}_c + u \pmod M$. Since Bob receives $\vec{m}_c + u \pmod M$ as the output of $\mathcal{F}_{\mathtt{OT}}$ and Alice sampled u in step 1 of the protocol, both Bob and Alice end up with an additive share modulo M of $\vec{m}_c$ when they finish executing $\Pi_{\mathtt{SOT}}$.

Assuming the existence of a protocol that successfully implements $\mathcal{F}_{\mathtt{OT}}$ in the malicious security setting, we now explain why the protocol $\Pi_{\mathtt{SOT}}$ implements the functionality $\mathcal{F}_{\mathtt{SOT}}$ in the malicious security setting. The security of $\mathcal{F}_{\mathtt{SOT}}$ comes from the ability of the simulator to read the inputs provided by the adversary to $\mathcal{F}_{\mathtt{OT}}$ and its other ability to map these inputs into $\mathcal{F}_{\mathtt{SOT}}$ inputs that make $\mathcal{F}_{\mathtt{SOT}}$ behave as an $\mathcal{F}_{\mathtt{OT}}$ that received the inputs chosen by the adversary. A description for how the mapping between the two types of inputs can be performed is found in the security proof for $\Pi_{\mathtt{SOT}}$.

Now we present the complete and formal description for the protocol $\Pi_{\mathtt{SOT}}$.

Protocol $\Pi_{\mathtt{SOT}^N_M}$

Inputs:

- Bob inputs $[\![c]\!]^B_N$.
- Alice inputs $\vec{m} \in \mathbb{Z}^N_M$ and $[\![c]\!]^A_N$.

Protocol Steps:

1. Alice locally samples $u \in_R \mathbb{Z}_M$.
2. Alice locally computes $\vec{m}' = \mathsf{cshift}_N(\vec{m}, [\![c]\!]^A_N) + u$, where $\mathsf{cshift}_N(\vec{m}, x)$ denotes a cyclic shift of x positions of $\vec{m}$'s elements.
3. The parties execute $\vec{m}_c + u, \perp \leftarrow \mathcal{F}_{\mathtt{OT}^N_M}(\vec{m}', [\![c]\!]^B_N)$
4. Output $[\![\vec{m}_c]\!]^A_M = u$ to Alice and $[\![\vec{m}_c]\!]^B_M = \vec{m}_c + u$ to Bob.

By analyzing this protocol's description and assuming its security proof (which is detailed in the next subsection) is correct, we can see that even though Shared OT is a more flexible primitive, it is as efficient as an Oblivious Transfer, while also being secure in the malicious setting. Protocol $\Pi_{\mathtt{SOT}}$ requires the same amount of rounds and transfers the same amount of bits between the two parties as the protocol that implements $\mathcal{F}_{\mathtt{OT}_M^N}$, while the computational overhead of $\Pi_{\mathtt{SOT}}$ is negligible. It is also interesting to note that our primitive can be pre-computed in the trusted initializer model as proposed by Rivest [18].

3.1.1 Security Proof

In this subsection, we present a proof of security for the SOT protocol. The proof is relatively simple, but we write it in detail here because it will make it easier to explain security proofs of our protocols for secure equality, comparison, and bit-decomposition - our main research contributions.

We follow the universal composability framework as introduced by Canetti in [5]. We refer to the original paper [5] for basic definitions and notations.

Our goal for this subsection is to prove the security of the SOT protocol in the $\mathcal{F}_{\mathtt{OT}}$-hybrid model. In other words, we aim to show that, in a hybrid world, where the parties have access to $\mathcal{F}_{\mathtt{OT}}$, the execution of $\Pi_{\mathtt{SOT}}$ perfectly simulates the ideal functionality $\mathcal{F}_{\mathtt{SOT}}$ even when the adversary $\mathcal{A}$ behaves maliciously. Mathematically:

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{E} \colon \mathrm{HYBRID}^{\mathcal{F}_{\mathtt{OT}}}_{\Pi_{\mathtt{SOT}},\mathcal{A},\mathcal{E}} \equiv \mathrm{IDEAL}_{\mathcal{F}_{\mathtt{SOT}},\mathcal{S},\mathcal{E}}$$

From now on, the variables in the simulated scenario will be written with a prime symbol ($'$).

Simulation: Alice Corrupted and Bob Honest

In this scenario, Alice is corrupted, which means that the simulator $\mathcal{S}$ can read hear her inputs ($[\![c]\!]_N^A$ and $\vec{m} \in \mathbb{Z}_M^N$) and her internal state. The simulator $\mathcal{S}$ runs an internal copy $\mathcal{A}'$ of the hybrid-world adversary $\mathcal{A}$, where all the interations between $\mathcal{S}$ and $\mathcal{A}'$ are those that Alice has with other parties($\mathcal{F}_{\mathtt{OT}}$ and $\mathcal{E}$). The behaviour of the simulator is described next.

Simulation Description

1. The environment $\mathcal{E}$ delivers the inputs $[\![c]\!]_N^A$ and $\vec{m}$ to the simulator $\mathcal{S}$, which activates $\mathcal{S}$. Upon its activation, $\mathcal{S}$ performs two actions. First, $\mathcal{S}$ delivers $[\![c]\!]_N^A$ and $\vec{m}$

to $\mathcal{A}'$. Second, $\mathcal{S}$ sends a message (`sample share`) to $\mathcal{F}_{\texttt{SOT}}$, awaits for the response $[\![\vec{m}_c]\!]^A_M$ and stores it internally.

2. Upon receiving a message (`chosen`) or (`invalid input`) from $\mathcal{F}_{\texttt{SOT}}$, relay the message to $\mathcal{A}'$ as if $\mathcal{F}_{\texttt{OT}}$ had sent it.

3. Upon receiving a message $(\texttt{propose}, \vec{v})$ from $\mathcal{A}'$, where $\vec{v} \notin \mathbb{Z}^N_M$, send $(\texttt{propose}, 0, \vec{v})$ to $\mathcal{F}_{\texttt{SOT}}$, causing $\mathcal{F}_{\texttt{SOT}}$ to send (`invalid input`) messages to both parties and halt.

4. Upon receiving a message $(\texttt{propose}, \vec{v})$ from $\mathcal{A}'$, where $\vec{v} \in \mathbb{Z}^N_M$, $\mathcal{S}$ computes $\vec{v}' = \vec{v} - [\![\vec{m}_c]\!]^A_M \pmod M$ and sends $(\texttt{propose}, 0, \vec{v}')$ to $\mathcal{F}_{\texttt{SOT}}$. Note that this causes Bob to receive $\vec{v}_{[\![c]\!]^B_N}$ as output from $\mathcal{F}_{\texttt{SOT}}$, which is the behaviour of $\mathcal{F}_{\texttt{OT}}$.

5. Upon receiving Alice's output from $\mathcal{F}_{\texttt{SOT}}$, $\mathcal{S}$ doesn't deliver it.

Indistinguishability

We now prove that no environment is able to distinguish between hybrid and ideal executions. We divide this proof in two parts. First, we show that the simulator succeeds in simulating the protocol, and second, we show that the messages exchanged during the hybrid and ideal executions are indistinguishable.

Part I: On the Simulation

- The adversary $\mathcal{A}'$ can misbehave in three ways. The first one is to send a message $(\texttt{propose}, \vec{v})$ before receiving a message (`chosen`), which causes both parties to receive (`invalid input`) messages in both worlds (hybrid and ideal). The second one is to send a message $(\texttt{propose}, \vec{v})$ after receiving a message (`chosen`), but where $\vec{v} \notin \mathbb{Z}^N_M$, which again causes both parties to receive (`invalid input`) messages in both worlds. The third is to send a message that does not follows the template $(\texttt{propose}, \vec{v})$, which simply does not cause any effect in both worlds.

- The adversary can also interact with the $\mathcal{F}_{\texttt{OT}}$ as expected that is, by sending a message $(\texttt{propose}, \vec{v})$ after receiving a message (`chosen`), where $\vec{v} \in \mathbb{Z}^N_M$. In the hybrid world, this will cause Alice and Bob to execute an $\mathcal{F}_{\texttt{OT}}$ where the selection index is $[\![c]\!]^B_N$ and the options vector is $\vec{v}$. But in the ideal world, $\mathcal{S}$ maps $\vec{v}$ to $\vec{v}'$ and executes $\mathcal{F}_{\texttt{SOT}}$ over the inputs $[\![c]\!]^B_N$, $[\![c]\!]^A_N := 0$ and $\vec{v}'$. This input mapping is made in order to make the $\mathcal{F}_{\texttt{SOT}}$ behave as the $\mathcal{F}_{\texttt{OT}}$ does in the hybrid world.

Part II: On the Probability Distributions

- First, we demostrate that the (`chosen`) message is delivered to Alice if and only if Bob has sent the message $(\texttt{choose}, u)$, where $u \in \mathbb{Z}_N$. This obviously happens, because $\mathcal{S}$ relays the message (`chosen`) if and only if it received (`chosen`) from $\mathcal{F}_{\texttt{SOT}}$.

- Second, we demostrate that Bob's output follows the same distribution regardless of the world in question, hybrid or ideal. Let it be the case that Bob and Alice sent the messages $(\texttt{choose}, u)$ and $(\texttt{propose}, \vec{v})$, respectively, where $u \in \mathbb{Z}_N$ and $\vec{v} \in \mathbb{Z}_M^N$. This means that in the hybrid world, Bob will receive the output $\vec{v}_u$ of $\mathcal{F}_{\texttt{OT}}(u, \vec{v})$. This also means that in the ideal world, Bob receives the output $[\![\vec{v}'_c]\!]_M^B$ of $\mathcal{F}_{\texttt{SOT}}([\![c]\!]_N, \vec{v}')$, where $[\![c]\!]_N^A = 0$, $[\![c]\!]_N^B = u$ and $\vec{v}' = \vec{v} - [\![\vec{v}'_c]\!]_M^A \pmod M$. But based on how the shares of c and the vector $\vec{v}'$ are constructed, we know that $c = u$ and $[\![\vec{v}'_c]\!]_M^B = \vec{v}_c$, which implies that Bob also receives $\vec{v}_u$ in the ideal world.

Simulation: Alice Honest and Bob Corrupted

In this scenario, Bob is corrupted, which means that the simulator $\mathcal{S}$ can read his input $[\![c]\!]_N^B$ and his internal state. Like in the last simulation case, $\mathcal{S}$ runs an internal copy $\mathcal{A}'$ of the hybrid-world adversary $\mathcal{A}$, where all the interations between $\mathcal{S}$ and $\mathcal{A}'$ are those that Bob has with other parties($\mathcal{F}_{\texttt{OT}}$ and $\mathcal{E}$). The behaviour of $\mathcal{S}$ is described next.

<u>Simulation Description</u>

1. The environment $\mathcal{E}$ delivers the input $[\![c]\!]_N^B$ to the simulator $\mathcal{S}$, this action activates $\mathcal{S}$. Upon its activation, $\mathcal{S}$ delivers $[\![c]\!]_N^B$ to $\mathcal{A}'$.
2. Upon receiving a message $(\texttt{invalid input})$ from $\mathcal{F}_{\texttt{SOT}}$, relay the message to $\mathcal{A}'$ as if $\mathcal{F}_{\texttt{OT}}$ had sent it.
3. Upon receiving a message $(\texttt{choose}, u)$ from $\mathcal{A}'$, relay the message to $\mathcal{F}_{\texttt{SOT}}$.
4. Upon receiving the output $[\![\vec{m}_c]\!]_M^B$ from $\mathcal{F}_{\texttt{SOT}}$, relay the message to $\mathcal{A}'$ as if $\mathcal{F}_{\texttt{OT}}$ had sent it.

<u>Indistinguishability</u>

We now prove that no environment is able to distinguish between hybrid and ideal executions in this simulation case. We structure the proof for this simulation case like we did for the last one.

<u>Part I: On the Simulation</u>

- The adversary $\mathcal{A}'$ can misbehave in two ways. The first one is by sending messages that do not match the pattern $(\texttt{choose}, u)$, which in both worlds (hybrid and ideal) does not cause any effect. The second one is by sending a message $(\texttt{choose}, u)$ where $u \notin \mathbb{Z}_N$, which in both worlds causes both parties to receive $(\texttt{invalid input})$ messages.

- The adversary can also interact with $\mathcal{F}_{\texttt{OT}}$ as expected, by sending a message $(\texttt{choose}, u)$ where $u \in \mathbb{Z}_N$. By simply relaying $(\texttt{choose}, u)$ to $\mathcal{F}_{\texttt{SOT}}$, the simulator $\mathcal{S}$ makes the ideal execution behave exactly the same as the hybrid one.

Part II: On the Probability Distributions

- In the case where the adversary $\mathcal{A}'$ sends a message $(\texttt{choose}, u)$, where $u \in \mathbb{Z}_N$, the behaviour of the protocol will be the same as if $\mathcal{A}'$ had acted honestly and the environment $\mathcal{E}$ had given $\mathcal{A}'$ the input u. By simply relaying the message $(\texttt{choose}, u)$ to $\mathcal{F}_{\texttt{SOT}}$, the simulator $\mathcal{S}$ is simulating the behavior of $\mathcal{E}$ delivering u to $\mathcal{A}'$ and $\mathcal{A}'$ acting honestly. Based on this, we can see that $\mathcal{S}$ simulates all the probability distributions perfectly.

Simulation: Both Alice and Bob are Honest

In this simulation case, the simulator $\mathcal{S}$ doesn't have access to the parties inputs or internal information. In order to simulate the message transcript, $\mathcal{S}$ needs only to run an internal copy of the protocol using arbitrary inputs. Regarding the outputs, $\mathcal{S}$ just lets the ideal functionality $\mathcal{F}_{\texttt{SOT}}$ deliver the prescribed outputs to Alice and Bob. By acting in this way, $\mathcal{S}$ makes hybrid and ideal executions perfectly indistinguishable from each other, when the adversary acts passively.

Simulation: Both Alice and Bob are Corrupted

In this scenario both parties are corrupted, which means that the simulator $\mathcal{S}$ has access to the internal state of both parties, including their input and randomness. This implies that $\mathcal{S}$ can completely and perfectly simulate the protocol. Therefore, no envionment $\mathcal{E}$ will be able to distinguish between the hybrid and ideal executions.

3.2 Elementwise Equality*

Let $a, b \in \mathbb{Z}_N$ be inputs shared by Alice and Bob, that is, Alice has $[\![a]\!]_N^A$ and $[\![b]\!]_N^A$ and Bob has $[\![a]\!]_N^B$ and $[\![b]\!]_N^B$. In this section we define two slightly different functions for determining equality:

$$\texttt{EEQ}(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \qquad \text{and} \qquad \texttt{EEQ}^*(a, b) = \begin{cases} 0 & \text{if } a = b \\ i \neq 0 & \text{if } a \neq b \end{cases}$$

Here i is some integer, $0 < i < M$, with M some protocol parameter. We first present a protocol for evaluating $\texttt{EEQ}^*$, whereas in the next section we will show how $\texttt{EEQ}$ can be obtained from $\texttt{EEQ}^*$ in a straightforward way. Below we present the functionality corresponding to computing the function $\texttt{EEQ}^*$. Please note that the modulus of the shared inputs and of the shared output are different.

Functionality: $(\llbracket c \rrbracket_M^A, \llbracket c \rrbracket_M^B) \leftarrow \mathcal{F}_{\texttt{EEQ}^*_{N,M}}(\llbracket a \rrbracket_N^A, \llbracket b \rrbracket_N^A)(\llbracket a \rrbracket_N^B, \llbracket b \rrbracket_N^B)$

Let $N \geq 2$ and $M > \lceil \log_2(N) \rceil$ be integers. The functionality $\mathcal{F}_{\texttt{EEQ}^*_\ell}$ runs with the parties Alice and Bob, and is parameterized by N and M.

- **Input**: Upon receiving a message from a party containing its shares of $\llbracket a \rrbracket_N$ and $\llbracket b \rrbracket_N$, check if both shares belong to $\mathbb{Z}_N$. If one of them does not belong, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other parties about the receival.
- **Output**: Upon receiving the shares of both parties, compute $\llbracket d \rrbracket_N$, where $\llbracket d \rrbracket_N = \llbracket a \rrbracket_N - \llbracket b \rrbracket_N$. After computing $\llbracket d \rrbracket_N$, set c as the Hamming distance between $\llbracket d \rrbracket_N^A$ and $\llbracket d \rrbracket_N^B$. Then, return to Alice and Bob their respective shares of $\llbracket c \rrbracket_M$. Note that $c = 0$ if $a = b$ and $1 \leq c \leq \lceil \log_2(N) \rceil$, otherwise.

With $\mathcal{F}_{\texttt{EEQ}^*_{N,M}}$ now formally defined, we show that building a protocol for $\texttt{EEQ}^*$ is quite easy when using Shared OTs and elementary local operation over shared elements. The intuition behind the protocol is the following. Let $\llbracket d \rrbracket_N = \llbracket a \rrbracket_N - \llbracket b \rrbracket_N$. Since $d = 0$ iff $\llbracket d \rrbracket_N^A = \llbracket d \rrbracket_N^B$, we can just privately compute h, the Hamming distance between the binary representations of $\llbracket d \rrbracket_N^A$ and $\llbracket d \rrbracket_N^B$ to obtain the desired output as specified by the functionality, given that $d = 0$ iff $a = b$ and that $h = 0$ iff $\llbracket d \rrbracket_N^A = \llbracket d \rrbracket_N^B$. The value of h is obtained by computing the weight of the bitwise xor of $\llbracket d \rrbracket_N^A$ and $\llbracket d \rrbracket_N^B$, which implies that the underlying modulus must be changed from 2 to $M > \lceil \lg N \rceil$ to perform this addition.

Protocol: $(\llbracket c \rrbracket_M^A, \llbracket c \rrbracket_M^B) \leftarrow \Pi_{\texttt{EEQ}^*_{N,M}}(\llbracket a \rrbracket_N^A, \llbracket b \rrbracket_N^A)(\llbracket a \rrbracket_N^B, \llbracket b \rrbracket_N^B)$

Set $\ell = \lceil \lg N \rceil$.

1. Party $X \in \{A, B\}$ locally computes $\llbracket d \rrbracket_N^X = \llbracket a \rrbracket_N^X - \llbracket b \rrbracket_N^X \pmod N$
2. Alice locally computes the binary expansion $\vec{u} \in \mathbb{Z}_2^\ell$ of $\llbracket d \rrbracket_N^A$.
3. Bob locally computes the binary expansion $\vec{v} \in \mathbb{Z}_2^\ell$ of $\llbracket d \rrbracket_N^B$.

4. Party $X \in \{A, B\}$ locally computes $[\![\vec{x}_i]\!]_2^X = [\![\vec{u}_i]\!]_2^X \oplus [\![\vec{v}_i]\!]_2^X$, for $0 \leq i \leq \ell - 1$.
5. Execute $[\![\vec{x}_i]\!]_M \leftarrow \mathcal{F}_{\mathtt{SOT}^2_M}((0,1), [\![\vec{x}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. (Converts $[\![\vec{x}_i]\!]_2$ to $[\![\vec{x}_i]\!]_M$)
6. Party $X \in \{A, B\}$ locally computes $[\![c]\!]_M^X = \sum_{i=0}^{\ell-1} [\![x_i]\!]_M^X \pmod M$. ($c$ is the Hamming distance between $[\![d]\!]_N^A$ and $[\![d]\!]_N^B$)

Next, in order to help the reader assimilate the protocol just presented, we give an example of a particular execution of $\Pi_{\mathtt{EEQ}^*_{N,M}}$. We go through every step of the protocol and show what values would be attributed to each variable involved in its execution.

Example: $([\![1]\!]_5^A, [\![1]\!]_5^B) \leftarrow \Pi_{\mathtt{EEQ}^*_{10,5}}(5,2)(2,5)$

Let $N = 10$, $M = 5$ and, $([\![a]\!]_{10}^A, [\![b]\!]_{10}^A) = (5,2)$ and $([\![a]\!]_{10}^B, [\![b]\!]_{10}^B) = (2,5)$ be Alice and Bob's inputs, respectively. Note that this implies $a = [\![a]\!]_{10}^B - [\![a]\!]_{10}^A = 7 \pmod{10}, b = [\![b]\!]_{10}^B - [\![b]\!]_{10}^A = 3 \pmod{10}$.

1. Alice and Bob locally compute $[\![d]\!]_{10}^A = [\![4]\!]_{10}^A = 3$ and $[\![d]\!]_{10}^B = [\![4]\!]_{10}^B = 7$, respectively.
2. Alice locally computes the binary expansion $\vec{u} = (0,0,1,1)$ of $[\![d]\!]_{10}^A = 3$.
3. Bob locally computes the binary expansion $\vec{v} = (0,1,1,1)$ of $[\![d]\!]_{10}^B = 7$.
4. Alice and Bob locally compute $[\![\vec{x}]\!]_2^A = \vec{u} = (0,0,1,1)$ and $[\![\vec{x}]\!]_2^B = \vec{v} = (0,1,1,1)$, respectively. This implies that $\vec{x} = (0,1,0,0)$.
5. Alice and Bob perform the following 4 SOTs:
   - $[\![\vec{x}_0]\!]_5 \leftarrow \mathcal{F}_{\mathtt{SOT}^2_5}((\mathbf{0},1), [\![\vec{x}_0]\!]_2)$, where $\vec{x}_0 = 0$.
   - $[\![\vec{x}_1]\!]_5 \leftarrow \mathcal{F}_{\mathtt{SOT}^2_5}((0,\mathbf{1}), [\![\vec{x}_1]\!]_2)$, where $\vec{x}_1 = 1$.
   - $[\![\vec{x}_2]\!]_5 \leftarrow \mathcal{F}_{\mathtt{SOT}^2_5}((\mathbf{0},1), [\![\vec{x}_2]\!]_2)$, where $\vec{x}_2 = 0$.
   - $[\![\vec{x}_3]\!]_5 \leftarrow \mathcal{F}_{\mathtt{SOT}^2_5}((\mathbf{0},1), [\![\vec{x}_3]\!]_2)$, where $\vec{x}_3 = 0$.
6. Alice and Bob locally compute shares of $[\![c]\!]_5 = \sum_{i=0}^{3} [\![\vec{x}_i]\!]_5 = [\![1]\!]_5$.

Theorem 1. *Protocol $\Pi_{\mathtt{EEQ}^*_{N,M}}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{EEQ}^*_{N,M}}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** From the definition of $\Pi_{\texttt{EEQ}^*}$, we know that $\vec{u}$ and $\vec{v}$ are the binary expansion of $[\![d]\!]_N^A$ and $[\![d]\!]_N^A$, respectively, and that $\vec{x}_i = \vec{u}_i \oplus \vec{v}_i$ for $0 \leq i \leq \ell - 1$. Based on this, we have that the value of c, computed on step 6, is the Hamming distance between $[\![d]\!]_N^A$ and $[\![d]\!]_N^B$. Thus, $\Pi_{\texttt{EEQ}^*}$ is correct.

Security: The simulation is very simple and proceeds as follows. The simulator $\mathcal{S}$ runs internally a copy of the adversary $\mathcal{A}$ and reproduces the real world protocol execution perfectly for $\mathcal{A}$. In order to do this, $\mathcal{S}$ simulates the protocol execution with dummy inputs for the uncorrupted parties. The simulator's leverage over $\mathcal{A}$ and $\mathcal{E}$ is the fact that $\mathcal{S}$ can perfectly simulate the outputs of a $\mathcal{F}_{\texttt{SOT}_M^N}$, since its outputs distributions are known. Considering this, it is clear that we can simulate the message exchanges that happen during the protocol, for any of the two parties. Now regarding the protocol's output, by the end of the protocol's simulation, $\mathcal{S}$ will have the corrupted party's shares of $[\![a]\!]_N$ and $[\![b]\!]_N$, which means $\mathcal{S}$ can fix these values in $\mathcal{F}_{\texttt{EEQ}^*_{N,M}}$. This will make the protocol's output compatible with the inputs chosen by $\mathcal{E}$. Based on this, we can conclude that no enviroment $\mathcal{E}$ can distinguish the real and ideal worlds. □

Note that the only interaction in $\Pi_{\texttt{EEQ}^*_{N,M}}$ takes place in Step 5. We can replace this interaction by another one that can be perfomed in advance during a preprocessing phase by implementing a randomized $\texttt{SOT}_M^2$. But in this case, some additional care must be taken in order to use the random values, computed during the preprocessing phase, to convert $[\![\vec{x}_i]\!]_2$ to $[\![\vec{x}_i]\!]_M$. This conversion is done on steps 7 through 9 of the following protocol.

Protocol: $([\![c]\!]_M^A, [\![c]\!]_M^B) \leftarrow \Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}([\![a]\!]_N^A, [\![b]\!]_N^A)([\![a]\!]_N^B, [\![b]\!]_N^B)$

Set $\ell = \lceil \lg N \rceil$.

1. Party $X \in \{A, B\}$ locally samples $[\![\vec{r}_i]\!]_2^X \in_R \mathbb{Z}_2$, for $0 \leq i \leq \ell - 1$.
2. Execute $[\![\vec{r}_i]\!]_M \leftarrow \mathcal{F}_{\texttt{SOT}_M^2}((0,1), [\![\vec{r}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. (Convert $[\![\vec{r}_i]\!]_2$ to $[\![\vec{r}_i]\!]_M$)
3. Party $X \in \{A, B\}$ locally computes $[\![d]\!]_N^X = [\![a]\!]_N^X - [\![b]\!]_N^X \pmod N$
4. Alice locally computes the binary expansion $\vec{u} \in \mathbb{Z}_2^\ell$ of $[\![d]\!]_N^A$.
5. Bob locally computes the binary expansion $\vec{v} \in \mathbb{Z}_2^\ell$ of $[\![d]\!]_N^B$.
6. Party $X \in \{A, B\}$ locally computes $[\![\vec{x}_i]\!]_2^X = [\![\vec{u}_i]\!]_2^X \oplus [\![\vec{v}_i]\!]_2^X$, for $0 \leq i \leq \ell - 1$.
7. Party $X \in \{A, B\}$ locally computes and reveals $[\![\vec{g}_i]\!]_2^X = [\![\vec{x}_i]\!]_2^X \oplus [\![\vec{r}_i]\!]_2^X$, for $0 \leq i \leq \ell - 1$. (Reveals $\vec{g}_i = \vec{x}_i \oplus \vec{r}_i$)

8. Alice locally computes $[\![\vec{x}_i]\!]_M^A = [\![\vec{r}_i]\!]_M^A - 2 \cdot \vec{g}_i \cdot [\![\vec{r}_i]\!]_M^A \pmod{M}$, for $0 \leq i \leq \ell - 1$. ($[\![\vec{x}_i]\!]_M^A = [\![\vec{u}_i \oplus \vec{v}_i]\!]_M^A$)

9. Bob locally computes $[\![\vec{x}_i]\!]_M^B = \vec{g}_i + [\![\vec{r}_i]\!]_M^B - 2 \cdot \vec{g}_i \cdot [\![\vec{r}_i]\!]_M^B \pmod{M}$, for $0 \leq i \leq \ell - 1$. ($[\![\vec{x}_i]\!]_M^B = [\![\vec{u}_i \oplus \vec{v}_i]\!]_M^B$)

10. Party $X \in \{A, B\}$ locally computes $[\![c]\!]_M^X = \sum_{i=0}^{\ell-1} [\![\vec{x}_i]\!]_M^X \pmod{M}$. ($c$ is the hamming distance between $[\![d]\!]_N^A$ and $[\![d]\!]_N^B$)

As we did for $\Pi_{\texttt{EEQ}^*_{N,M}}$, we now go on to describe the execution of $\Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}$ for a particular set of arguments. This is done for the same reason an example of an execution of $\Pi_{\texttt{EEQ}^*_{N,M}}$ was described in more details.

Example: $([\![1]\!]_5^A, [\![1]\!]_5^B) \leftarrow \Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{10,5}}(5, 2)(2, 5)$

Let $N = 10$, $M = 5$ and, $([\![a]\!]_{10}^A, [\![b]\!]_{10}^A) = (5, 2)$ and $([\![a]\!]_{10}^B, [\![b]\!]_{10}^B) = (2, 5)$ be Alice and Bob's inputs, respectively. Note that this implies $a = [\![a]\!]_{10}^B - [\![a]\!]_{10}^A = 7 \pmod{10}, b = [\![b]\!]_{10}^B - [\![b]\!]_{10}^A = 3 \pmod{10}$.

1. Alice and Bob respectively sample:
   - $[\![\vec{r}_0]\!]_2^A = 0, [\![\vec{r}_0]\!]_2^B = 0$. ($[\![\vec{r}_0]\!]_2 = [\![0]\!]_2$)
   - $[\![\vec{r}_1]\!]_2^A = 1, [\![\vec{r}_1]\!]_2^B = 0$. ($[\![\vec{r}_1]\!]_2 = [\![1]\!]_2$)
   - $[\![\vec{r}_2]\!]_2^A = 1, [\![\vec{r}_2]\!]_2^B = 1$. ($[\![\vec{r}_2]\!]_2 = [\![0]\!]_2$)
   - $[\![\vec{r}_3]\!]_2^A = 0, [\![\vec{r}_3]\!]_2^B = 1$. ($[\![\vec{r}_3]\!]_2 = [\![1]\!]_2$)

2. Alice and Bob perform the following 4 SOTs:
   - $[\![\vec{r}_0]\!]_5 \leftarrow \mathcal{F}_{\texttt{SOT}^2_5}((\mathbf{0}, 1), [\![\vec{r}_0]\!]_2)$. ($[\![\vec{r}_0]\!]_5 = [\![0]\!]_5$)
   - $[\![\vec{r}_1]\!]_5 \leftarrow \mathcal{F}_{\texttt{SOT}^2_5}((0, \mathbf{1}), [\![\vec{r}_1]\!]_2)$. ($[\![\vec{r}_1]\!]_5 = [\![1]\!]_5$)
   - $[\![\vec{r}_2]\!]_5 \leftarrow \mathcal{F}_{\texttt{SOT}^2_5}((\mathbf{0}, 1), [\![\vec{r}_2]\!]_2)$. ($[\![\vec{r}_2]\!]_5 = [\![0]\!]_5$)
   - $[\![\vec{r}_3]\!]_5 \leftarrow \mathcal{F}_{\texttt{SOT}^2_5}((0, \mathbf{1}), [\![\vec{r}_3]\!]_2)$. ($[\![\vec{r}_3]\!]_5 = [\![1]\!]_5$)

3. Alice and Bob locally compute $[\![d]\!]_{10}^A = [\![4]\!]_{10}^A = 3$ and $[\![d]\!]_{10}^B = [\![4]\!]_{10}^B = 7$, respectively.

4. Alice locally computes the binary expansion $\vec{u} = (0, 0, 1, 1)$ of $[\![d]\!]_{10}^A = 3$.

5. Bob locally computes the binary expansion $\vec{v} = (0, 1, 1, 1)$ of $[\![d]\!]_{10}^B = 7$.

6. Alice and Bob locally compute $[\![\vec{x}]\!]_2^A = (0,0,1,1)$ and $[\![\vec{x}]\!]_2^B = (0,1,1,1)$, respectively. This implies $\vec{x} = (0,1,0,0)$.

7. Alice and Bob locally compute and reveal $[\![\vec{g}]\!]_2^A = (0,1,0,1)$ and $[\![\vec{g}]\!]_2^B = (0,1,0,0)$, respectively. This implies $\vec{g} = (0,0,0,1)$.

8. Alice and Bob respectively locally compute:

   - $[\![\vec{x}_0]\!]_5^A = [\![\vec{r}_0]\!]_5^A - 2 \cdot 0 \cdot [\![\vec{r}_0]\!]_5^A = [\![\vec{r}_0]\!]_5^A$
   - $[\![\vec{x}_1]\!]_5^A = [\![\vec{r}_1]\!]_5^A - 2 \cdot 0 \cdot [\![\vec{r}_1]\!]_5^A = [\![\vec{r}_1]\!]_5^A$
   - $[\![\vec{x}_2]\!]_5^A = [\![\vec{r}_2]\!]_5^A - 2 \cdot 0 \cdot [\![\vec{r}_2]\!]_5^A = [\![\vec{r}_2]\!]_5^A$
   - $[\![\vec{x}_3]\!]_5^A = [\![\vec{r}_3]\!]_5^A - 2 \cdot 1 \cdot [\![\vec{r}_3]\!]_5^A = -[\![\vec{r}_3]\!]_5^A$
   - $[\![\vec{x}_0]\!]_5^B = 0 + [\![\vec{r}_0]\!]_5^B - 2 \cdot 0 \cdot [\![\vec{r}_0]\!]_5^B = [\![\vec{r}_0]\!]_5^B$
   - $[\![\vec{x}_1]\!]_5^B = 0 + [\![\vec{r}_1]\!]_5^B - 2 \cdot 0 \cdot [\![\vec{r}_1]\!]_5^B = [\![\vec{r}_1]\!]_5^B$
   - $[\![\vec{x}_2]\!]_5^B = 0 + [\![\vec{r}_2]\!]_5^B - 2 \cdot 0 \cdot [\![\vec{r}_2]\!]_5^B = [\![\vec{r}_2]\!]_5^B$
   - $[\![\vec{x}_3]\!]_5^B = 1 + [\![\vec{r}_3]\!]_5^B - 2 \cdot 1 \cdot [\![\vec{r}_3]\!]_5^B = 1 - [\![\vec{r}_3]\!]_5^B = 0 - [\![\vec{r}_3]\!]_5^A$

9. Alice and Bob locally compute shares of $[\![c]\!]_5 = [\![1]\!]_5$.

Theorem 2. *Protocol $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*_{N,M}}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{EEQ}^*_{N,M}}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** From step 3 through 5 of the protocol's definition, we can see that $\vec{u}$ and $\vec{v}$ are the binary expasion of $[\![d]\!]_N^A$ and $[\![d]\!]_N^A$, respectively. From step 6 through 7, we can also see that $[\![\vec{x}_i]\!]_2 = [\![\vec{u}_i \oplus \vec{v}_i]\!]_2$ and $[\![\vec{g}_i]\!]_2 = [\![\vec{x}_i \oplus \vec{r}_i]\!]_2$, for $0 \leq i \leq \ell - 1$. Based on this and step 8, we have that $[\![\vec{x}_i]\!]_M = [\![\vec{g}_i + \vec{r}_i - 2 \cdot \vec{g}_i \cdot \vec{r}_i]\!]_M = [\![\vec{g}_i \oplus \vec{r}_i]\!]_M = [\![\vec{x}_i]\!]_M$ for $0 \leq i \leq \ell - 1$. Given this and step 10, we can see that c is the Hamming distance between $[\![d]\!]_N^A$ and $[\![d]\!]_N^B$. Thus, protocol $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*}$ is correct.

Security: The reasoning behind the security proof for this protocol is very similar to the previous proof. The only difference is the levarage that the simulator has over $\mathcal{A}$ and $\mathcal{E}$. In the case of $\Pi_{\mathtt{EEQ}^*_{N,M}}$, the leverage the simulator has over $\mathcal{A}$ and $\mathcal{E}$ is its capacity to perfectly simulate the $\mathcal{F}_{\mathtt{SOT}^N_M}$'s outputs, because the distribution of the outputs is always the same. In the case of $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*}$, the simulator is also capable of perfectly simulating the outputs of the $\mathcal{F}_{\mathtt{SOT}^N_M}$s, also for the same reasoning, but in this case, the simulator can leverage the fact that it will always know the distribution for the values of $\vec{d}$, the vector revealed in the 8th step. $\square$

3.3 Elementwise Equality

We now address secure evaluation of the function $\texttt{EEQ}$. First we present its functionality. Observe that the output modulus M always equals 2, and is therefore omitted from the notation.

Functionality: $([\![c]\!]_2^A, [\![c]\!]_2^B) \leftarrow \mathcal{F}_{\texttt{EEQ}_N}([\![a]\!]_N^A, [\![b]\!]_N^A)([\![a]\!]_N^B, [\![b]\!]_N^B)$

The functionality $\mathcal{F}_{\texttt{EEQ}_N}$ runs with the parties Alice and Bob, and is parameterized by an integer $N \geq 2$.

- **Input**: Upon receiving a message from a party containing its shares of $[\![a]\!]_N$ and $[\![b]\!]_N$, check if both shares belong to $\mathbb{Z}_N$. If one of them does not belong, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other parties about the receival.
- **Output**: Upon receiving both parties shares, reconstruct a and b. After reconstruction, set $c = 1$ if $a = b$, otherwise set $c = 0$. Then, return to Alice and Bob their respective shares of $[\![c]\!]_2$.

Note that $\texttt{EEQ}$ can be obtained from $\texttt{EEQ}^*$ followed remapping the possible outputs as follows: 0 maps to 1; any value greater than 0 maps to 0. This remapping is implemented by a randomized 1-out-of-N OT with choice vector $\vec{m} = (1, 0, \ldots, 0)$ and choice value c. In our notation this corresponds to a call to $\texttt{SOT}_2^M$ with inputs $\mathsf{One}_M(0)$ and h, where $M = \ell + 1$ and h represents the output of $\texttt{EEQ}^*(a, b)$.

Protocol: $([\![c]\!]_2^A, [\![c]\!]_2^B) \leftarrow \Pi_{\texttt{EEQ}_N}([\![a]\!]_N^A, [\![b]\!]_N^A)([\![a]\!]_N^B, [\![b]\!]_N^B)$

Set $\ell = \lceil \lg N \rceil$.

1. $[\![h]\!]_{\ell+1} \leftarrow \Pi_{\texttt{EEQ}^*_{N,\ell+1}}([\![a]\!]_N, [\![b]\!]_N)$. $(h = 0 \iff a = b)$
2. Execute $[\![c]\!]_2 = \mathcal{F}_{\texttt{SOT}_2^{\ell+1}}(\mathsf{One}_{\ell+1}(0), [\![h]\!]_{\ell+1})$. ($c = 1$ if $h = 0$, otherwise $c = 0$)

Protocol: $([\![c]\!]_2^A, [\![c]\!]_2^B) \leftarrow \Pi^{\mathcal{P}}_{\texttt{EEQ}_N}([\![a]\!]_N^A, [\![b]\!]_N^A)([\![a]\!]_N^B, [\![b]\!]_N^B)$

Set $\ell = \lceil \lg N \rceil$.

1. $[\![h]\!]_{\ell+1} \leftarrow \Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,\ell+1}}([\![a]\!]_N, [\![b]\!]_N)$. ($h = 0 \iff a = b$)
2. Execute $[\![c]\!]_2 = \mathcal{F}_{\texttt{SOT}^{\ell+1}_2}(\mathsf{One}_{\ell+1}(0), [\![h]\!]_{\ell+1})$. ($c = 1$ if $h = 0$, otherwise $c = 0$)

Next, we instantiate $\Pi_{\texttt{EEQ}_N}$ for a particular set of arguments and go through its two steps showing the values of the variables contained in the protocol's description. Since the only difference between $\Pi_{\texttt{EEQ}_N}$ and $\Pi^{\mathcal{P}}_{\texttt{EEQ}_N}$ is the equality star protocol used in their first steps, the same execution example is virtually interchangeable for these two protocols.

Example: $([\![0]\!]^A_2, [\![0]\!]^B_2) \leftarrow \Pi_{\texttt{EEQ}_8}(5, 2)(2, 5)$

Let $N = 8$ and, $([\![a]\!]^A_N, [\![b]\!]^A_N) = (5, 2)$ and $([\![a]\!]^B_N, [\![b]\!]^B_N) = (2, 5)$ be Alice and Bob's inputs, respectively. Note that this implies $a = [\![a]\!]^B_8 - [\![a]\!]^A_8 = 7 \pmod 8$, $b = [\![b]\!]^B_8 - [\![b]\!]^A_8 = 3 \pmod 8$ and $\ell = 3$.

1. Alice and Bob execute $[\![1]\!]_4 \leftarrow \Pi_{\texttt{EEQ}^*_{8,4}}([\![7]\!]_8, [\![3]\!]_8)$.
2. Alice and Bob execute $[\![0]\!]_2 = \mathcal{F}_{\texttt{SOT}^4_2}((1, \mathbf{0}, 0, 0), [\![1]\!]_4)$.

Theorem 3. *Protocol $\Pi_{\texttt{EEQ}_N}$ is correct and securely implements the functionality $\mathcal{F}_{\texttt{EEQ}_N}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** The correctness of this protocol follows directly from the correctness of $\Pi_{\texttt{EEQ}^*_{N,M}}$ and the fact that we will have $c = 1$ iff $h = 0$.

Security: By making some small alterations to the security proof of $\Pi_{\texttt{EEQ}^*_{N,M}}$, we can also prove the security of the protocol $\Pi_{\texttt{EEQ}_N}$. In the case of $\Pi_{\texttt{EEQ}^*_{N,M}}$, the leverage the simulator has over $\mathcal{A}$ and $\mathcal{E}$ is its capacity to perfectly simulate the $\mathcal{F}_{\texttt{SOT}^N_M}$'s outputs, because the distribution of the outputs is always the same. In the case of $\Pi_{\texttt{EEQ}_N}$, the simulator has higher leverage over the $\mathcal{A}$ and $\mathcal{E}$, because it cannot only perfectly simulate the outputs of $\mathcal{F}_{\texttt{SOT}^N_M}$s but also perfectly simulate the output of the protocol used to instantiate $\mathcal{F}_{\texttt{EEQ}^*_{N_M}}$, since the distribution of the output values is always known. $\square$

Theorem 4. *Protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}_N}$ is correct and securely implements the functionality $\mathcal{F}_{\texttt{EEQ}_N}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness and Security:** The same ideas used prove $\Pi_{\texttt{EEQ}^*_{N,M}}$'s correctness and security apply to the correctness and security of $\Pi^{\mathcal{P}}_{\texttt{EEQ}_N}$. $\square$

3.4 Bitwise Integer Comparison

Another important task, usually perfomed as part of some other task, is the comparison of two secret shared elements, a and b.

$$\texttt{BLT}(a,b) = \begin{cases} 1 & \text{if } a < b \\ 0 & \text{if } a \geq b \end{cases}$$

Note that a and b can be shared as elements modulo some integer N, or the bits of their binary representation can be shared modulo 2. Here we consider the second alternative. This leads to the following formal definition for the private bitwise comparison functionality $\mathcal{F}_{\texttt{BLT}_\ell}$.

Functionality: $(\llbracket c\rrbracket_2^A, \llbracket c\rrbracket_2^B) \leftarrow \mathcal{F}_{\texttt{BLT}_\ell}(\llbracket\vec{a}\rrbracket_2^A, \llbracket\vec{b}\rrbracket_2^A)(\llbracket\vec{a}\rrbracket_2^B, \llbracket\vec{b}\rrbracket_2^B)$

$\mathcal{F}_{\texttt{BLT}_\ell}$ runs with the parties Alice and Bob, and is parametrized by the length ℓ of the bit arrays being compared.

- **Input**: Upon receiving a message from a party with its shares of $\llbracket\vec{a}\rrbracket_2$ and $\llbracket\vec{b}\rrbracket_2$, check if the shares of $\vec{a}$ and $\vec{b}$ are both in $\mathbb{Z}_2^\ell$. If one of them is not, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other parties about the receipt.
- **Output**: Upon receiving the shares of both parties, reconstruct $\vec{a}$ and $\vec{b}$. After reconstruction, perform the bitwise comparison of $\vec{a}$ and $\vec{b}$, and set $c = 1$ if $\vec{a} < \vec{b}$, otherwise set $c = 0$. Then, return shares of $\llbracket c\rrbracket_2$ to Alice and Bob.

Using Shared OTs and the previously described protocols $\Pi^{\mathcal{P}}_{\texttt{EEQ}^*}$ and $\Pi_{\texttt{EEQ}^*}$, we present two protocols which implement $\mathcal{F}_{\texttt{BLT}_\ell}$, which we denote by $\Pi_{\texttt{BLT}_\ell}$ and $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$. But first we give an intuitive explanation for the idea behind the protocols proposed in this section, which is to use Shared OTs to privatly compute the following boolean expression:

$$c = (\bigoplus_{i=0}^{\ell-1} \vec{b}_i \wedge \vec{s}_i) \oplus (\bigoplus_{i=0}^{\ell-2} \vec{b}_i \wedge \vec{s}_{i+1})$$

This idea is inspired by the comparision protocol proposed in [9], with adjustments to take advantage of our SOT primitive.

In order to intuitively understand why the previously described Boolean expression computes the comparison we desire, we must first accurately define $\vec{s}$ and understand its behaviour when $\vec{a} = \vec{b}$ and when $\vec{a} \neq \vec{b}$. We define $\vec{s}$ in the following way:

$$\vec{s}_i = \bigvee_{j=i}^{\ell-1} \vec{a}_j \oplus \vec{b}_j, \text{ for } i \in \{0, 1, \ldots, \ell-1\}$$

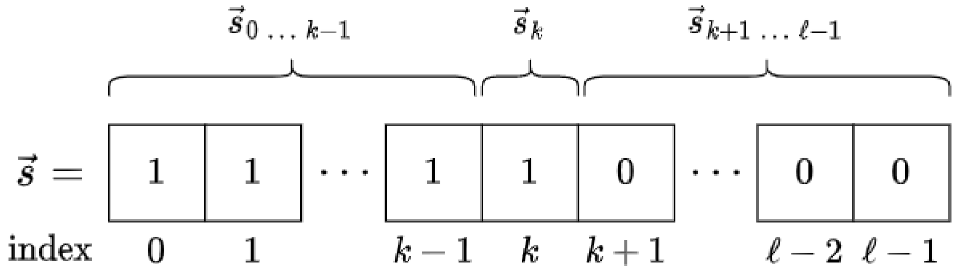
The behaviour of $\vec{s}$ is easy to predict when $\vec{a} = \vec{b}$, since in this case we have $\vec{a}_i \oplus \vec{b}_i = 0$ for $i \in \{0, 1, \ldots, \ell-1\}$. This means that if $\vec{a} = \vec{b}$, then $\vec{s}_i = 0$, for $i \in \{0, 1, \ldots, \ell-1\}$. Lets now consider the case where $\vec{a} \neq \vec{b}$. Since $\vec{a} \neq \vec{b}$, there exists exactly one most significant bit position k where $\vec{a}_i \oplus \vec{b}_i = 1$ (where bits $\vec{a}_i$ and $\vec{b}_i$ are different). In turn we can divide $\vec{s}$ in three sections based on k, which allows us to understand the behaviour of $\vec{s}$ when $\vec{a} \neq \vec{b}$. These three sections of $\vec{s}$ are: between 0 and $k-1$, between $k+1$ and $\ell-1$ and the element $\vec{s}_k$. We can now analyze every section separately.

Let us start with the section between $k+1$ and $\ell-1$, the most significant of the three. Since k is the position of the most significant pair of bits where $\vec{a}_i \oplus \vec{b}_i = 1$, we know that $\vec{a}_j \oplus \vec{b}_j = 0$, for $j \in \{k+1, \ldots, \ell-1\}$. Based on the definition of $\vec{s}$, we can see that $\vec{s}_i = 0$ for $i \in \{k+1, \ldots, \ell-1\}$, which means that all positions of $\vec{s}$ between $k+1$ and $\ell-1$ contain only 0s. Now, let us focus on the section that only contains $\vec{s}_k$. Since $\vec{a}_k \oplus \vec{b}_k = 1$ and based on the definition of $\vec{s}$, it is obvious that $\vec{s}_k = 1$. Now, let us study the behaviour of the last section of the vector $\vec{s}$, the section between 0 and $k-1$. Let us first rewrite the definition of $\vec{s}$ in the following way, for $i \in \{0, 1, \ldots, k-1\}$:

$$\vec{s}_i = (\bigvee_{i=0}^{k-1} \vec{a}_i \oplus \vec{b}_i) \vee (\vec{a}_k \oplus \vec{b}_k) \vee (\bigvee_{i=k+1}^{\ell-1} \vec{a}_i \oplus \vec{b}_i), \text{ for } i \in \{0, 1, \ldots, k-1\}$$

Since $\vec{a}_k \oplus \vec{b}_k = 1$, we can see that $\vec{s}_i = 1$, for $i \in \{0, 1, \ldots, k-1\}$. Based on the behaviour of these three analyzed sections, we can conclude that the vector $\vec{s}$ will look the following way, when $\vec{a} \neq \vec{b}$:

Figure 3.2: The three sections of vector $\vec{s}$.



Source: Created by author.

Now, still assuming that $\vec{a} \neq \vec{b}$, we proceed to explain how this behaviour of $\vec{s}$ leads to the correctness of the Boolean expression that defines c. In order to do this, we need to first define four more vectors, $\vec{s}', \vec{y}, \vec{y}', \vec{z} \in \mathbb{Z}_2^\ell$. These vectors are formally defined in the following way:

$$\vec{s}'_{\ell-1} = 0; \vec{s}'_i = \vec{s}_{i+1}, \text{ for } \in \{0, 1, \ldots, \ell-2\}$$

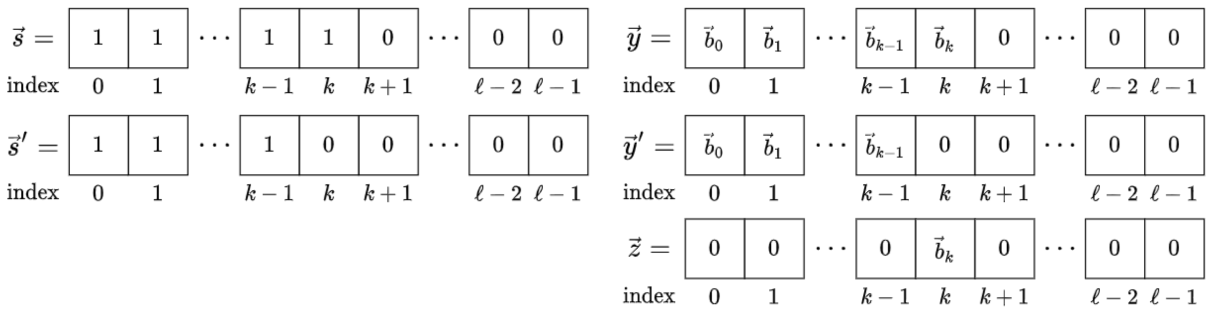
$$\vec{y}_i = \vec{s}_i \wedge \vec{b}_i, \text{ for } \in \{0, 1, \ldots, \ell-1\}$$

$$\vec{y}'_i = \vec{s}'_i \wedge \vec{b}_i, \text{ for } \in \{0, 1, \ldots, \ell-1\}$$

$$\vec{z}_i = \vec{y}_i \oplus \vec{y}'_i, \text{ for } \in \{0, 1, \ldots, \ell-1\}$$

It is important to note one particular fact about $\vec{z}$. The vector $\vec{z}$ contains the value of $\vec{b}_k$ in exactly one of its positions and 0s in all others. The reason for this can be understood by visualizing what the vectors $\vec{s}, \vec{s}', \vec{y}, \vec{y}'$ look like. In the following diagram we show these four vectors, plus the vector $\vec{z}$, in order to make the reasoning completely clear.

Figure 3.3: Relationship between vectors $\vec{s}$,$\vec{s}'$,$\vec{y}$,$\vec{y}'$ and $\vec{z}$.



$\vec{s} =$ 1 1 ⋯ 1 1 0 ⋯ 0 0
index 0 1 $k-1$ k $k+1$ $\ell-2$ $\ell-1$
$\vec{s}' =$ 1 1 ⋯ 1 0 0 ⋯ 0 0
index 0 1 $k-1$ k $k+1$ $\ell-2$ $\ell-1$
$\vec{y} =$ $\vec{b}_0$ $\vec{b}_1$ ⋯ $\vec{b}_{k-1}$ $\vec{b}_k$ 0 ⋯ 0 0
index 0 1 $k-1$ k $k+1$ $\ell-2$ $\ell-1$
$\vec{y}' =$ $\vec{b}_0$ $\vec{b}_1$ ⋯ $\vec{b}_{k-1}$ 0 0 ⋯ 0 0
index 0 1 $k-1$ k $k+1$ $\ell-2$ $\ell-1$
$\vec{z} =$ 0 0 ⋯ 0 $\vec{b}_k$ 0 ⋯ 0 0
index 0 1 $k-1$ k $k+1$ $\ell-2$ $\ell-1$

Source: Created by author.

Based on this important fact about z, we can also conclude that $\bigoplus_{i=0}^{\ell-1} \vec{z}_i = \vec{b}_k$. This means that if $\vec{a} \neq \vec{b}$, then $\bigoplus_{i=0}^{\ell-1} \vec{z} = \vec{b}_k$. Since k is the position of the most significant pair of bits where $\vec{a}_i \oplus \vec{b}_i = 1$ (where $\vec{a}_i \neq \vec{b}_i$), we know that if $\vec{a} \neq \vec{b}$, then $\vec{b}_k = \bigoplus_{i=0}^{\ell-1} \vec{z} = 1 \iff b > a$. Thus, assuming $\vec{a} \neq \vec{b}$, $\bigoplus_{i=0}^{\ell-1} \vec{z} = 1 \iff b > a$. It turns out that if we simply expand the equation $\bigoplus_{i=0}^{\ell-1} \vec{z}_i$ and rearrange this expanded equation, we have

$$\begin{aligned}
\bigoplus_{i=0}^{\ell-1} \vec{z}_i &= \bigoplus_{i=0}^{\ell-1} \vec{y}_i \oplus \vec{y}'_i && (3.1)\\
&= \bigoplus_{i=0}^{\ell-1} \vec{y}_i \oplus \bigoplus_{i=0}^{\ell-1} \vec{y}'_i && (3.2)\\
&= (\bigoplus_{i=0}^{\ell-1} \vec{s}_i \wedge \vec{b}_i) \oplus (\bigoplus_{i=0}^{\ell-1} \vec{s}'_i \wedge \vec{b}_i) && (3.3)\\
&= (\bigoplus_{i=0}^{\ell-1} \vec{s}_i \wedge \vec{b}_i) \oplus (\bigoplus_{i=0}^{\ell-2} \vec{s}'_i \wedge \vec{b}_i) && (3.4)\\
&= (\bigoplus_{i=0}^{\ell-1} \vec{s}_i \wedge \vec{b}_i) \oplus (\bigoplus_{i=0}^{\ell-2} \vec{s}_{i+1} \wedge \vec{b}_i) && (3.5)\\
&= c && (3.6)
\end{aligned}$$

Because of this, we can conclude that if $\vec{a} \neq \vec{b}$, then $c = 1 \iff b > a$. Now we just need to show that $c = 0$ if $\vec{a} = \vec{b}$, in order to see that the Boolean expression that defines c is correct. But since we already saw that $\vec{s}_i = 0$ if $\vec{a} = \vec{b}$, for $i \in \{0, 1, \ldots, \ell - 1\}$, it is easy to analyze the Boolean expresion that defines c and realize that $c = 0$ if $\vec{a} = \vec{b}$. Therefore, we can see that $c = 1 \iff b > a$, which means that the expression that defines c behaves in the way it is suppose to.

The two protocols proposed in this section simply use SOTs in order to compute the Boolean expression that defines c. Comments have been added to the description of the protocols in order to make the relation between every line of the protocol and the Boolean expression that we want to compute clearer. As in the last section, we present two protocols, one without preprocessing, and one with preprocessing. The only difference is how they instantiate the functionality $\mathcal{F}_{\texttt{EEQ*}}$, one uses $\Pi_{\texttt{EEQ*}}$ and the other uses $\Pi^{\mathcal{P}}_{\texttt{EEQ*}}$.

Protocol: $([\![c]\!]_2^A, [\![c]\!]_2^B) \leftarrow \Pi_{\texttt{BLT}_\ell}([\![\vec{a}]\!]_2^A, [\![\vec{b}]\!]_2^A)([\![\vec{a}]\!]_2^B, [\![\vec{b}]\!]_2^B)$

Let $\lambda = 2(\ell' + 1)$, where ℓ' is the amount of bits necessary to represent an element of $\mathbb{Z}_{\ell+1}$.

1. Execute $[\![\vec{x}_i]\!]_{\ell+1} \leftarrow \mathcal{F}_{\texttt{SOT}^2_{\ell+1}}((0, 1), [\![\vec{a}_i + \vec{b}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. ($\vec{x}_i = \vec{a}_i \oplus \vec{b}_i$)
2. Execute $[\![\vec{\beta}_i]\!]_\lambda \leftarrow \mathcal{F}_{\texttt{SOT}^2_\lambda}((0, \frac{\lambda}{2}), [\![\vec{b}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. ($\vec{\beta}_i \in \{0, \frac{\lambda}{2}\}$; $\vec{\beta}_i = \vec{b}_i \cdot \frac{\lambda}{2}$)
3. Locally compute $[\![\vec{s}_i]\!]_{\ell+1} = \sum_{j=i}^{\ell-1} [\![\vec{x}_j]\!]_{\ell+1}$, for $0 \leq i \leq \ell - 1$. ($0 \leq \vec{s}_i \leq \ell$; $\vec{s}_i > 0 \iff \bigvee_{j=i}^{\ell-1} \vec{x}_j$)
4. Execute $[\![\vec{h}_i]\!]_\lambda \leftarrow \Pi_{\texttt{EEQ}^*_{\ell+1,\lambda}}([\![\vec{s}_i]\!]_{\ell+1}, [\![0]\!]_{\ell+1})$, for $0 \leq i \leq \ell - 1$. ($0 \leq \vec{h}_i \leq \ell'$; $\vec{h}_i > 0 \iff \vec{s}_i > 0 \iff \bigvee_{j=i}^{\ell-1} \vec{x}_j$)

5. Locally compute $[\![\vec{t_i}]\!]_\lambda = [\![\vec{h_i}]\!]_\lambda + [\![\vec{\beta_i}]\!]_\lambda$, for $0 \le i \le \ell - 1$. ($0 \le \vec{h_i} \le \ell'$; $\vec{\beta_i} = \vec{b_i} \cdot \frac{\lambda}{2}$; $\vec{t_i} = \vec{h_i} + \vec{\beta_i} > \frac{\lambda}{2} \iff \vec{h_i} > 0 \wedge \vec{\beta_i} = \frac{\lambda}{2} \iff \vec{b_i} \wedge \bigvee_{j=i}^{\ell-1} \vec{x_j}$)

6. Locally compute $[\![\vec{q_i}]\!]_\lambda = [\![\vec{h}_{i+1}]\!]_\lambda + [\![\vec{\beta_i}]\!]_\lambda$, for $0 \le i \le \ell - 2$. ($\vec{q_i} = \vec{h}_{i+1} + \vec{\beta_i} > \frac{\lambda}{2} \iff \vec{b_i} \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x_j}$)

7. Execute $[\![\vec{d_i}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^\lambda}(\mathsf{One}_\lambda(\frac{\lambda}{2}+1, \frac{\lambda}{2}-1), [\![\vec{t_i}]\!]_\lambda)$, for $0 \le i \le \ell - 1$. ($\vec{d_i} = [\vec{t_i} > \frac{\lambda}{2}] = \vec{b_i} \wedge \bigvee_{j=i}^{\ell-1} \vec{x_j}$)

8. Execute $[\![\vec{e_i}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^\lambda}(\mathsf{One}_\lambda(\frac{\lambda}{2}+1, \frac{\lambda}{2}-1), [\![\vec{q_i}]\!]_\lambda)$, for $0 \le i \le \ell - 2$. ($\vec{e_i} = [\vec{q_i} > \frac{\lambda}{2}] = \vec{b_i} \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x_j}$)

9. Locally compute $[\![c]\!]_2 = \sum_{i=0}^{\ell-1} [\![\vec{d_i}]\!]_2 + \sum_{i=0}^{\ell-2} [\![\vec{e_i}]\!]_2$. ($c = \bigoplus_{i=0}^{\ell-1} \vec{d_i} \oplus \bigoplus_{i=0}^{\ell-2} \vec{e_i}$)

Protocol: $([\![c]\!]_2^A, [\![c]\!]_2^B) \leftarrow \Pi_{\mathtt{BLT}_\ell}^{\mathcal{P}}([\![\vec{a}]\!]_2^A, [\![\vec{b}]\!]_2^A)([\![\vec{a}]\!]_2^B, [\![\vec{b}]\!]_2^B)$

Let $\lambda = 2(\ell' + 1)$, where ℓ' is the amount of bits necessary to represent an element of $\mathbb{Z}_{\ell+1}$.

1. Execute $[\![\vec{x_i}]\!]_{\ell+1} \leftarrow \mathcal{F}_{\mathtt{SOT}_{\ell+1}^2}((0,1), [\![\vec{a_i} + \vec{b_i}]\!]_2)$, for $0 \le i \le \ell - 1$. ($\vec{x_i} = \vec{a_i} \oplus \vec{b_i}$)

2. Execute $[\![\vec{\beta_i}]\!]_\lambda \leftarrow \mathcal{F}_{\mathtt{SOT}_\lambda^2}((0, \frac{\lambda}{2}), [\![\vec{b_i}]\!]_2)$, for $0 \le i \le \ell - 1$. ($\vec{\beta_i} = \frac{\lambda}{2}$ if $\vec{b_i} = 1$ and $\beta = 0$, otherwise)

3. Locally compute $[\![\vec{s_i}]\!]_{\ell+1} = \sum_{j=i}^{\ell-1} [\![\vec{x_j}]\!]_{\ell+1}$, for $0 \le i \le \ell - 1$.

4. $[\![\vec{h_i}]\!]_\lambda \leftarrow \Pi_{\mathtt{EEQ}_{\ell+1,\lambda}^*}^{\mathcal{P}}([\![\vec{s_i}]\!]_{\ell+1}, [\![0]\!]_{\ell+1})$, for $0 \le i \le \ell - 1$. ($\vec{h_i} = 0 \iff \vec{s_i} = 0 \iff \neg \bigvee_{j=i}^{\ell-1} \vec{x_j}$)

5. Locally compute $[\![\vec{t_i}]\!]_\lambda = [\![\vec{h_i}]\!]_\lambda + [\![\vec{\beta_i}]\!]_\lambda$, for $0 \le i \le \ell - 1$. ($\vec{t_i} > \frac{\lambda}{2} \iff \vec{b_i} \wedge \bigvee_{j=i}^{\ell-1} \vec{x_j}$)

6. Locally compute $[\![\vec{q_i}]\!]_\lambda = [\![\vec{h}_{i+1}]\!]_\lambda + [\![\vec{\beta_i}]\!]_\lambda$, for $0 \le i \le \ell - 2$. ($\vec{q_i} > \frac{\lambda}{2} \iff \vec{b_i} \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x_j}$)

7. Execute $[\![\vec{d_i}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^\lambda}(\mathsf{One}_\lambda(\frac{\lambda}{2}+1, \frac{\lambda}{2}-1), [\![\vec{t_i}]\!]_\lambda)$, for $0 \le i \le \ell - 1$. ($\vec{d_i} = [\vec{t_i} > \frac{\lambda}{2}] = \vec{b_i} \wedge \bigvee_{j=i}^{\ell-1} \vec{x_j}$)

8. Execute $[\![\vec{e_i}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^\lambda}(\mathsf{One}_\lambda(\frac{\lambda}{2}+1, \frac{\lambda}{2}-1), [\![\vec{q_i}]\!]_\lambda)$, for $0 \le i \le \ell - 2$. ($\vec{e_i} = [\vec{q_i} > \frac{\lambda}{2}] = \vec{b_i} \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x_j}$)

9. Locally compute $[\![c]\!]_2 = \sum_{i=0}^{\ell-1} [\![\vec{d_i}]\!]_2 + \sum_{i=0}^{\ell-2} [\![\vec{e_i}]\!]_2$. ($c = \bigoplus_{i=0}^{\ell-1} \vec{d_i} \oplus \bigoplus_{i=0}^{\ell-2} \vec{e_i}$)

As we have done for the equality protols, we now describe an execution of $\Pi_{\texttt{BLT}_\ell}$ in detail to help the reader better understand how the comparison protocol behaves. We only provide an example for $\Pi_{\texttt{BLT}_\ell}$ because the execution of $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$ using the same arguments would be really similar. After providing the description of execution for $\Pi_{\texttt{BLT}_\ell}$, we move on to prove the correctness and security of both $\Pi_{\texttt{BLT}_\ell}$ and $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$.

Example: $(\llbracket 0\rrbracket_2^A, \llbracket 0\rrbracket_2^B) \leftarrow \Pi_{\texttt{BLT}_8}(\llbracket\vec{a}\rrbracket_2^A, \llbracket\vec{b}\rrbracket_2^A)(\llbracket\vec{a}\rrbracket_2^B, \llbracket\vec{b}\rrbracket_2^B)$, where $\vec{a} = (1,0,1)$ and $\vec{b} = (0,1,1)$

This implies $\ell = 3$, $\ell' = 2$ and $\lambda = 6$. Also, note that $\vec{a}$ and $\vec{b}$ are the binary expansions of 5 and 3, respectively.

1. Alice and Bob perform the following 3 SOTs:
   - $\llbracket\vec{x}_0\rrbracket_4 \leftarrow \mathcal{F}_{\texttt{SOT}^2_4}((\mathbf{0}, 1), \llbracket 0\rrbracket_2)$. $(\vec{x}_0 = 0)$
   - $\llbracket\vec{x}_1\rrbracket_4 \leftarrow \mathcal{F}_{\texttt{SOT}^2_4}((0, \mathbf{1}), \llbracket 1\rrbracket_2)$. $(\vec{x}_1 = 1)$
   - $\llbracket\vec{x}_2\rrbracket_4 \leftarrow \mathcal{F}_{\texttt{SOT}^2_4}((0, \mathbf{1}), \llbracket 1\rrbracket_2)$. $(\vec{x}_2 = 1)$
2. Alice and Bob perform the following 3 SOTs:
   - $\llbracket\vec{\beta}_0\rrbracket_6 \leftarrow \mathcal{F}_{\texttt{SOT}^2_6}((0, \mathbf{3}), \llbracket 1\rrbracket_2)$. $(\vec{\beta}_0 = 3)$
   - $\llbracket\vec{\beta}_1\rrbracket_6 \leftarrow \mathcal{F}_{\texttt{SOT}^2_6}((0, \mathbf{3}), \llbracket 1\rrbracket_2)$. $(\vec{\beta}_1 = 3)$
   - $\llbracket\vec{\beta}_2\rrbracket_6 \leftarrow \mathcal{F}_{\texttt{SOT}^2_6}((\mathbf{0}, 3), \llbracket 0\rrbracket_2)$. $(\vec{\beta}_2 = 0)$
3. Alice and Bob compute their respective shares of the following 3 shared values:
   - $\llbracket\vec{s}_0\rrbracket_4 = \sum_{j=0}^{2}\llbracket\vec{x}_j\rrbracket_4 = \llbracket 0\rrbracket_4 + \llbracket 1\rrbracket_4 + \llbracket 1\rrbracket_4 = \llbracket 2\rrbracket_4$
   - $\llbracket\vec{s}_1\rrbracket_4 = \sum_{j=1}^{2}\llbracket\vec{x}_j\rrbracket_4 = \llbracket 1\rrbracket_4 + \llbracket 1\rrbracket_4 = \llbracket 2\rrbracket_4$
   - $\llbracket\vec{s}_2\rrbracket_4 = \sum_{j=2}^{2}\llbracket\vec{x}_j\rrbracket_4 = \llbracket 1\rrbracket_4$
4. Alice and Bob execute $\Pi_{\texttt{EEQ}^*_{4,6}}$ three times:
   - $\llbracket\vec{h}_0\rrbracket_6 \leftarrow \Pi_{\texttt{EEQ}^*_{4,6}}(\llbracket 2\rrbracket_4, \llbracket 0\rrbracket_4)$. $(1 \le \vec{h}_0 \le 2)$
   - $\llbracket\vec{h}_1\rrbracket_6 \leftarrow \Pi_{\texttt{EEQ}^*_{4,6}}(\llbracket 2\rrbracket_4, \llbracket 0\rrbracket_4)$. $(1 \le \vec{h}_1 \le 2)$
   - $\llbracket\vec{h}_2\rrbracket_6 \leftarrow \Pi_{\texttt{EEQ}^*_{4,6}}(\llbracket 1\rrbracket_4, \llbracket 0\rrbracket_4)$. $(1 \le \vec{h}_2 \le 2)$
5. Alice and Bob compute their respective shares of the following 3 shared values:
   - $\llbracket\vec{t}_0\rrbracket_6 = \llbracket\vec{h}_0\rrbracket_6 + \llbracket\vec{\beta}_0\rrbracket_6 = \llbracket\vec{h}_0\rrbracket_6 + \llbracket 3\rrbracket_6$. $(4 \le \vec{t}_0 \le 5)$
   - $\llbracket\vec{t}_1\rrbracket_6 = \llbracket\vec{h}_1\rrbracket_6 + \llbracket\vec{\beta}_1\rrbracket_6 = \llbracket\vec{h}_1\rrbracket_6 + \llbracket 3\rrbracket_6$. $(4 \le \vec{t}_1 \le 5)$

- $[\![\vec{t}_2]\!]_6 = [\![\vec{h}_2]\!]_6 + [\![\vec{\beta}_2]\!]_6 = [\![\vec{h}_2]\!]_6$. $(1 \leq \vec{t}_2 \leq 2)$

6. Alice and Bob compute their respective shares of the following 2 shared values:
   - $[\![\vec{q}_0]\!]_6 = [\![\vec{h}_1]\!]_6 + [\![\vec{\beta}_0]\!]_6 = [\![\vec{h}_1]\!]_6 + [\![3]\!]_6$. $(4 \leq \vec{q}_0 \leq 5)$
   - $[\![\vec{q}_1]\!]_6 = [\![\vec{h}_2]\!]_6 + [\![\vec{\beta}_1]\!]_6 = [\![\vec{h}_2]\!]_6 + [\![3]\!]_6$. $(4 \leq \vec{q}_1 \leq 5)$
7. Alice and Bob perform the following 3 SOTs:
   - $[\![\vec{d}_0]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^6}((0,0,0,0,\mathbf{1},\mathbf{1}), [\![\vec{t}_0]\!]_6)$. $(\vec{d}_0 = 1)$
   - $[\![\vec{d}_1]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^6}((0,0,0,0,\mathbf{1},\mathbf{1}), [\![\vec{t}_1]\!]_6)$. $(\vec{d}_1 = 1)$
   - $[\![\vec{d}_2]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^6}((0,\mathbf{0},\mathbf{0},0,1,1), [\![\vec{t}_2]\!]_6)$. $(\vec{d}_2 = 0)$
8. Alice and Bob perform the following 2 SOTs:
   - $[\![\vec{e}_0]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^6}((0,0,0,0,\mathbf{1},\mathbf{1}), [\![\vec{q}_0]\!]_6)$. $(\vec{e}_0 = 1)$
   - $[\![\vec{e}_1]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^6}((0,0,0,0,\mathbf{1},\mathbf{1}), [\![\vec{q}_1]\!]_6)$. $(\vec{e}_1 = 1)$
9. Alice and Bob compute their respective shares of the following shared value:
   - $[\![c]\!]_2 = \sum_{i=0}^{2} [\![\vec{d}_i]\!]_2 + \sum_{i=0}^{1} [\![\vec{e}_i]\!]_2 = ([\![1]\!]_2 + [\![1]\!]_2 + [\![0]\!]_2) + ([\![1]\!]_2 + [\![1]\!]_2) = [\![0]\!]_2$

Theorem 5. *Protocol $\Pi_{\mathtt{BLT}_\ell}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{BLT}_\ell}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** First, it is important to note the behaviour of variables $\vec{x}, \vec{\beta}, \vec{d}, \vec{e}$ and c. It is straightfoward to see that they respect the following equations:

$$x_i = a_i \oplus b_i, \text{ for } 0 \leq i \leq \ell - 1$$

$$\beta_i = b_i \cdot \frac{\lambda}{2}, \text{ for } 0 \leq i \leq \ell - 1$$

$$\vec{d}_i = \begin{cases} 1 & \text{if } t_i \geq \frac{\lambda}{2} + 1 \\ 0 & \text{otherwise} \end{cases}, \text{ for } 0 \leq i \leq \ell - 1$$

$$\vec{e}_i = \begin{cases} 1 & \text{if } q_i \geq \frac{\lambda}{2} + 1 \\ 0 & \text{otherwise} \end{cases}, \text{ for } 0 \leq i \leq \ell - 2$$

$$c = \bigoplus_{i=0}^{\ell-1} \vec{d}_i \oplus \bigoplus_{i=0}^{\ell-2} \vec{e}_i$$

Now, suppose that $a = b$. This means $s_i = 0$, for $0 \leq i \leq \ell - 1$. This implies that $t_i < \frac{\lambda}{2} + 1$ and $q_j < \frac{\lambda}{2} + 1$, for $0 \leq i \leq \ell - 1$ and $0 \leq j \leq \ell - 2$. This leads to the fact that $d_i = 0$ and $e_j = 0$, for $0 \leq i \leq \ell - 1$ and $0 \leq j \leq \ell - 2$. Therefore, $c = 0$ if $a = b$.

Next, suppose that $a \neq b$. This implies the existence of a pair of most significant bits $\vec{a}_k$ and $\vec{b}_k$, where $\vec{a}_k \neq \vec{b}_k$. For $i > k$, we have $\vec{t}_i, \vec{q}_i < \frac{\lambda}{2}+1$ and $\vec{d}_i = \vec{e}_i = 0$, since $\vec{s}_i = 0$. For $i < k$, we have $\vec{s}_i, \vec{s}_{i+1} \geq 1$, since $a_k \neq b_k$, which implies that $t_i > \frac{\lambda}{2} \iff \beta = \frac{\lambda}{2}$ and $q_i > \frac{\lambda}{2} \iff \beta = \frac{\lambda}{2}$. This leads to the fact that $\vec{d}_i = \vec{e}_i$, for $i < k$. Based on this, $c = \vec{d}_k \oplus \vec{e}_k$ if $k \leq \ell - 2$ and $c = \vec{d}_k$, otherwise. But, since $\vec{s}_k = 1$ and $\vec{a}_k \neq \vec{b}_k$, if $k \leq \ell - 2$, we will have $\vec{s}_{k+1} = 0$, which leads to $\vec{e}_k = 0$. Thus, $c = \vec{d}_k$ for $0 \leq k \leq \ell - 1$, if $a \neq b$.

Suppose that $a < b$. Since $a \neq b$, we have $c = \vec{d}_k$. Because $b > a$, we have $\vec{b}_k = 1, \vec{a}_k = 0$ and $\vec{s}_k = 1$, implying that $\vec{d}_k = 1$. This means that $c = 1$, if $a < b$.

Suppose that $a > b$. Since $a \neq b$, we have $c = \vec{d}_k$. Because $b < a$, we have $\vec{b}_k = 0, \vec{a}_k = 1$ and $\vec{s}_k = 1$, implying that $\vec{d}_k = 0$. This means that $c = 0$, if $a > b$.

This demonstrates that the described protocol will output 1, if $a < b$ and 0, otherwise.

Security: The same rationale used to prove the security of $\Pi_{\texttt{EEQ}_N}$ can also be used to prove the security of $\Pi_{\texttt{BLT}_\ell}$. ☐

Theorem 6. *Protocool $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$ is correct and securely implements the functionality $\mathcal{F}_{\texttt{BLT}_\ell}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness and Security:** The same arguments used to prove the correctness and security of $\Pi_{\texttt{BLT}_\ell}$ can be used to prove the correctness and security of $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$. ☐

3.5 Bit-Decomposition Protocol

When performing private two-party computations over a shared element $\beta \in \mathbb{Z}_N$, it is often useful to also have access to the binary expansion of β. Therefore, many protocols perform a task over this shared element β that takes the shares of β as input and outputs the binary expasion of β as a sequence of shared bits. This task is widely known and is commonly called Bit-Decomposition. In this section we formally define this task as a functionality and propose two protocols that implement this functionality.

In the case of the Bit-Decomposition functionality defined by us in this section, we assume that the input is an element β additively shared modulo 2^ℓ and the output is a sequence of ℓ shared bits, where $\ell \geq 2$. Based on the previously described intuitive understanding of the Bit-Decomposition task, we formally define the following functionality $\mathcal{F}_{\texttt{BD}_\ell}$ that performs this task:

Functionality: $(\llbracket\vec{b}\rrbracket_2^A, \llbracket\vec{b}\rrbracket_2^B) \leftarrow \mathcal{F}_{\mathtt{BD}_\ell}(\llbracket\beta\rrbracket_{2^\ell}^A)(\llbracket\beta\rrbracket_{2^\ell}^B)$

$\mathcal{F}_{\mathtt{BD}_\ell}$ runs with the parties Alice and Bob, and is parametrized by $\ell \geq 2$.

- **Input**: Upon receiving a message from a party with its share of $\llbracket\vec{\beta}\rrbracket_{2^\ell}$, check if its share is contained in $\mathbb{Z}_{2^\ell}$. If it's not, then abort. Otherwise, record the share, ignore any subsequent message from that party and inform the other parties about the receival.

- **Output**: Upon receiving both parties shares, reconstruct β. After reconstruction, compute the binary expansion $b_{\ell-1}b_{\ell-2}\dots b_0$ of β and return to Alice and Bob there respective shares of $\llbracket b_{\ell-1}\rrbracket_2, \llbracket b_{\ell-2}\rrbracket_2 \dots \llbracket b_0\rrbracket_2$.

With this definition in mind, we now present two protocols which efficiently implement it. The general idea behind both protocols is the same, which is to take the binary expansions $u, v \in \mathbb{Z}_2^\ell$ of $\llbracket\beta\rrbracket_{2^\ell}^B$ and $-\llbracket\beta\rrbracket_{2^\ell}^A$, respectively, and perform binary addition over $\vec{u}$ and $\vec{v}$, but ignoring the last carry bit generated when computing the binary addition, which causes the binary addition result to be the same as that of computing binary addition modulo 2^ℓ. Since the output of the binary addition modulo 2^ℓ, performed by the protocol, is a sequence of shared bits and $\beta = \llbracket\beta\rrbracket_{2^\ell}^B - \llbracket\beta\rrbracket_{2^\ell}^A \pmod{2^\ell}$, the output of the addition is the desired output for the bit-decomposition protocol.

In order to compute the binary addition modulo 2^ℓ, we first compute the carry bit vector $\vec{c}$, the vector containing the carry bits generated when computing the binary addition of $\vec{u}$ and $\vec{v}$, and then compute $\vec{b}_i = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i$, for $0 \leq i \leq \ell - 1$, which will be the vector containing the binary expansion of β. By writing down the expressions that define the values of the least significant bits of $\vec{c}$ we can easily get to the expression that defines $\vec{c}$ as a whole. The expressions that define the four least significant bits of $\vec{c}$ are:

$$\vec{c}_0 = 0$$

$$\vec{c}_1 = \vec{u}_0 \wedge \vec{v}_0$$

$$\vec{c}_2 = (\vec{u}_1 \wedge \vec{v}_1) \oplus ((\vec{u}_1 \oplus \vec{v}_1) \wedge (\vec{u}_0 \wedge \vec{v}_0))$$

$$\vec{c}_3 = (\vec{u}_2 \wedge \vec{v}_2) \oplus ((\vec{u}_2 \oplus \vec{v}_2) \wedge (\vec{u}_1 \wedge \vec{v}_1)) \oplus ((\vec{u}_2 \oplus \vec{v}_2) \wedge (\vec{u}_1 \oplus \vec{v}_1) \wedge (\vec{u}_0 \wedge \vec{v}_0))$$

As previously stated, we can analyse these expressions and workout the following set of Boolean equations that define the carry bit vector $\vec{c}$.

$$\vec{c}_0 = 0 \text{ and } \vec{c}_i = \bigoplus_{j=0}^{i-1} \vec{t}_{i,j}, \text{ for } 1 \leq i \leq \ell - 1$$

$$\vec{t}_{i,j} = \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k, \text{ for } 0 \leq j < i \leq \ell - 1$$
$$\vec{g}_i = \vec{u}_i \wedge \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$
$$\vec{x}_i = \vec{u}_i \oplus \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$

Both these protocols just privately compute the vector $\vec{c}$, by using SOTs to evaluate the previously described Boolean equations, and then finish by computing the vector $\vec{b}$. Note that the only difference between the the two protocols is found in their 4th step. But the values of $\vec{t}_{i,j}$, computed in the protocols 4th step, will be the same in both protocols, since the protocols differ only in how the values are computed. More specifically, $\Pi_{\mathtt{BD}}$ uses SOTs to compute the values of $\vec{t}_{i,j}$, while $\Pi'_{\mathtt{BD}}$ uses the functionality $\mathcal{F}_{\mathtt{EEQ}_\ell}$.

Protocol: $([\![\vec{b}]\!]_2^A, [\![\vec{b}]\!]_2^B) \leftarrow \Pi_{\mathtt{BD}_\ell}([\![\beta]\!]_{2^\ell}^A)([\![\beta]\!]_{2^\ell}^B)$

Let $\vec{v} \in \mathbb{Z}_2^\ell$ and $\vec{u} \in \mathbb{Z}_2^\ell$ be the binary expansions of $(-[\![\beta]\!]_{2^\ell}^A \pmod{2^\ell})$ and $[\![\beta]\!]_{2^\ell}^B$, respectively.

1. Execute $[\![\vec{g}_i]\!]_\ell \leftarrow \mathcal{F}_{\mathtt{SOT}_\ell^3}((0,0,1), [\![\vec{u}_i + \vec{v}_i]\!]_3)$, for $0 \leq i \leq \ell - 1$ $(\vec{g}_i = \vec{u}_i \wedge \vec{v}_i)$.
2. Execute $[\![\vec{x}_i]\!]_\ell \leftarrow \mathcal{F}_{\mathtt{SOT}_\ell^2}((0,1), [\![\vec{u}_i + \vec{v}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. $(\vec{x}_i = \vec{u}_i \oplus \vec{v}_i)$
3. Locally compute $[\![\vec{h}_{i,j}]\!]_\ell \leftarrow [\![\vec{g}_j]\!]_\ell + \sum_{k=j+1}^{i-1} [\![\vec{x}_k]\!]_\ell$, for $0 \leq j < i \leq \ell - 1$. $(\vec{h}_{i,j} = i - j \iff \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k)$
4. Perform $[\![\vec{t}_{i,j}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^\ell}(\mathsf{One}_\ell(i-j), [\![\vec{h}_{i,j}]\!]_\ell)$, for $0 \leq j < i \leq \ell - 1$. $(\vec{t}_{i,j} = 1 \iff \vec{h}_{i,j} = i - j \iff \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k;\ \vec{t}_{i,j} = \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k)$
5. Let $\vec{c}_0 = 0$. Locally compute $[\![c_i]\!]_2 = \bigoplus_{j=0}^{i-1} [\![\vec{t}_{i,j}]\!]_2$, for $1 \leq i \leq \ell - 1$. $(\vec{c}_i = \bigoplus_{j=0}^{i-1} \vec{t}_{i,j})$
6. Locally compute $[\![b_i]\!]_2 = [\![u_i]\!]_2 + [\![v_i]\!]_2 + [\![c_i]\!]_2$, for $0 \leq i \leq \ell - 1$. $(\vec{b}_i = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i)$

Protocol: $([\![\vec{b}]\!]_2^A, [\![\vec{b}]\!]_2^B) \leftarrow \Pi'_{\mathtt{BD}_\ell}([\![\beta]\!]_{2^\ell}^A)([\![\beta]\!]_{2^\ell}^B)$

Let $\vec{v} \in \mathbb{Z}_2^\ell$ and $\vec{u} \in \mathbb{Z}_2^\ell$ be the binary expansions of $(-[\![\beta]\!]_{2^\ell}^A \pmod{2^\ell})$ and $[\![\beta]\!]_{2^\ell}^B$, respectively.

1. Execute $[\![\vec{g}_i]\!]_\ell \leftarrow \mathcal{F}_{\mathtt{SOT}_\ell^3}((0,0,1), [\![\vec{u}_i + \vec{v}_i]\!]_3)$, for $0 \leq i \leq \ell - 1$. $(\vec{g}_i = \vec{u}_i \wedge \vec{v}_i)$
2. Execute $[\![\vec{x}_i]\!]_\ell \leftarrow \mathcal{F}_{\mathtt{SOT}_\ell^2}((0,1), [\![\vec{u}_i + \vec{v}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. $(\vec{x}_i = \vec{u}_i \oplus \vec{v}_i)$

3. Locally compute $[\![\vec{h}_{i,j}]\!]_\ell \leftarrow [\![\vec{g}_j]\!]_\ell + \sum_{k=j+1}^{i-1}[\![\vec{x}_k]\!]_\ell$, for $0 \le j < i \le \ell - 1$. ($\vec{h}_{i,j} = i - j \iff \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$)

4. Perform $[\![\vec{t}_{i,j}]\!]_2 \leftarrow \mathcal{F}_{\texttt{EEQ}_\ell}([\![\vec{h}_{i,j}]\!]_\ell, [\![i-j]\!]_\ell)$, for $0 \le j < i \le \ell - 1$. ($\vec{t}_{i,j} = 1 \iff \vec{h}_{i,j} = i - j \iff \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$; $\vec{t}_{i,j} = \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$)

5. Let $\vec{c}_0 = 0$. Locally compute $[\![c_i]\!]_2 = \bigoplus_{j=0}^{i-1}[\![\vec{t}_{i,j}]\!]_2$, for $1 \le i \le \ell - 1$. ($\vec{c}_i = \bigoplus_{j=0}^{i-1} \vec{t}_{i,j}$)

6. Locally compute $[\![b_i]\!]_2 = [\![u_i]\!]_2 + [\![v_i]\!]_2 + [\![c_i]\!]_2$, for $0 \le i \le \ell - 1$. ($\vec{b}_i = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i$)

Below we added two examples of executions for the two bit-decomposition protocols to help the reader better understand how the protocols work.

Example: $([\![\vec{b}]\!]_2^A, [\![\vec{b}]\!]_2^B) \leftarrow \Pi_{\texttt{BD}_3}(5)(5)$, where $\vec{b} = (0,0,0)$

Let $\ell = 3$, $[\![\beta]\!]_8^B = 5$ and $[\![\beta]\!]_8^A = 5$. This implies that $\vec{v} = (0,1,1)$ and $\vec{u} = (1,0,1)$.

1. Alice and Bob execute the following 3 SOTs:
   - $[\![\vec{g}_0]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^3}((0,0,\mathbf{1}), [\![2]\!]_3)$. ($\vec{g}_0 = 1$)
   - $[\![\vec{g}_1]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^3}((0,\mathbf{0},1), [\![1]\!]_3)$. ($\vec{g}_1 = 0$)
   - $[\![\vec{g}_2]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^3}((0,\mathbf{0},1), [\![1]\!]_3)$. ($\vec{g}_2 = 0$)
2. Alice and Bob execute the following 3 SOTs:
   - $[\![\vec{x}_0]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^2}((\mathbf{0},1), [\![0]\!]_2)$. ($\vec{x}_0 = 0$)
   - $[\![\vec{x}_1]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^2}((0,\mathbf{1}), [\![1]\!]_2)$. ($\vec{x}_1 = 1$)
   - $[\![\vec{x}_2]\!]_3 \leftarrow \mathcal{F}_{\texttt{SOT}_3^2}((0,\mathbf{1}), [\![1]\!]_2)$. ($\vec{x}_2 = 1$)
3. Alice and Bob compute their respective shares of the following 3 shared values:
   - $[\![\vec{h}_{1,0}]\!]_3 = [\![\vec{g}_0]\!]_3 + \sum_{k=1}^{0}[\![\vec{x}_k]\!]_3 = [\![\vec{g}_0]\!]_3 = [\![1]\!]_3$
   - $[\![\vec{h}_{2,0}]\!]_3 = [\![\vec{g}_0]\!]_3 + \sum_{k=1}^{1}[\![\vec{x}_k]\!]_3 = [\![\vec{g}_0]\!]_3 + [\![\vec{x}_1]\!]_3 = [\![2]\!]_3$
   - $[\![\vec{h}_{2,1}]\!]_3 = [\![\vec{g}_1]\!]_3 + \sum_{k=2}^{1}[\![\vec{x}_k]\!]_3 = [\![\vec{g}_1]\!]_3 = [\![0]\!]_3$
4. Alice and Bob compute their respective shares of the following 3 shared values:
   - $[\![\vec{t}_{1,0}]\!]_2 \leftarrow \mathcal{F}_{\texttt{SOT}_2^3}((0,\mathbf{1},0), [\![\vec{h}_{1,0}]\!]_3)$. ($\vec{t}_{1,0} = 1$)

- $[\![\vec{t}_{2,0}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^3}((0,0,\mathbf{1}), [\![\vec{h}_{2,0}]\!]_3)$. $(\vec{t}_{2,0} = 1)$
- $[\![\vec{t}_{2,1}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{SOT}_2^3}((\mathbf{0},1,0), [\![\vec{h}_{2,1}]\!]_3)$. $(\vec{t}_{2,1} = 0)$

5. Alice and Bob compute their respective shares of the following 3 shared values:

   - $[\![\vec{c}_0]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{c}_1]\!]_2 = \sum_{j=0}^{0} [\![\vec{t}_{1,j}]\!]_2 = [\![\vec{t}_{1,0}]\!]_2 = [\![1]\!]_2$
   - $[\![\vec{c}_2]\!]_2 = \sum_{j=0}^{1} [\![\vec{t}_{2,j}]\!]_2 = [\![\vec{t}_{2,0}]\!]_2 + [\![\vec{t}_{2,1}]\!]_2 = [\![1]\!]_2$

6. Alice and Bob compute their respective shares of the following 3 shared values:

   - $[\![\vec{b}_0]\!]_2 = [\![\vec{u}_0]\!]_2 + [\![\vec{v}_0]\!]_2 + [\![\vec{c}_0]\!]_2 = [\![1]\!]_2 + [\![1]\!]_2 + [\![0]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{b}_1]\!]_2 = [\![\vec{u}_1]\!]_2 + [\![\vec{v}_1]\!]_2 + [\![\vec{c}_1]\!]_2 = [\![0]\!]_2 + [\![1]\!]_2 + [\![1]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{b}_2]\!]_2 = [\![\vec{u}_2]\!]_2 + [\![\vec{v}_2]\!]_2 + [\![\vec{c}_2]\!]_2 = [\![1]\!]_2 + [\![0]\!]_2 + [\![1]\!]_2 = [\![0]\!]_2$

Example: $([\![\vec{b}]\!]_2^A, [\![\vec{b}]\!]_2^B) \leftarrow \Pi'_{\mathtt{BD}_3}(5)(5)$, where $\vec{b} = (0,0,0)$

Let $\ell = 3$, $[\![\beta]\!]_8^B = 5$ and $[\![\beta]\!]_8^A = 5$. This implies that $\vec{v} = (0,1,1)$ and $\vec{u} = (1,0,1)$.

1. Alice and Bob execute the following 3 SOTs:

   - $[\![\vec{g}_0]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^3}((0,0,\mathbf{1}), [\![2]\!]_3)$. $(\vec{g}_0 = 1)$
   - $[\![\vec{g}_1]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^3}((0,\mathbf{0},1), [\![1]\!]_3)$. $(\vec{g}_1 = 0)$
   - $[\![\vec{g}_2]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^3}((0,\mathbf{0},1), [\![1]\!]_3)$. $(\vec{g}_2 = 0)$

2. Alice and Bob execute the following 3 SOTs:

   - $[\![\vec{x}_0]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^2}((\mathbf{0},1), [\![0]\!]_2)$. $(\vec{x}_0 = 0)$
   - $[\![\vec{x}_1]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^2}((0,\mathbf{1}), [\![1]\!]_2)$. $(\vec{x}_1 = 1)$
   - $[\![\vec{x}_2]\!]_3 \leftarrow \mathcal{F}_{\mathtt{SOT}_3^2}((0,\mathbf{1}), [\![1]\!]_2)$. $(\vec{x}_2 = 1)$

3. Alice and Bob compute their respective shares of the following 3 shared values:

   - $[\![\vec{h}_{1,0}]\!]_3 = [\![\vec{g}_0]\!]_3 + \sum_{k=1}^{0} [\![\vec{x}_k]\!]_3 = [\![\vec{g}_0]\!]_3 = [\![1]\!]_3$
   - $[\![\vec{h}_{2,0}]\!]_3 = [\![\vec{g}_0]\!]_3 + \sum_{k=1}^{1} [\![\vec{x}_k]\!]_3 = [\![\vec{g}_0]\!]_3 + [\![\vec{x}_1]\!]_3 = [\![2]\!]_3$
   - $[\![\vec{h}_{2,1}]\!]_3 = [\![\vec{g}_1]\!]_3 + \sum_{k=2}^{1} [\![\vec{x}_k]\!]_3 = [\![\vec{g}_1]\!]_3 = [\![0]\!]_3$

4. Alice and Bob perform $\mathcal{F}_{\mathtt{EEQ}_3}$ 3 times in the following way:
   - $[\![\vec{t}_{1,0}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{EEQ}_3}([\![\vec{h}_{1,0}]\!]_3, [\![1]\!]_3)$. $(\vec{t}_{1,0} = 1)$
   - $[\![\vec{t}_{2,0}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{EEQ}_3}([\![\vec{h}_{2,0}]\!]_3, [\![2]\!]_3)$. $(\vec{t}_{2,0} = 1)$
   - $[\![\vec{t}_{2,1}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{EEQ}_3}([\![\vec{h}_{2,1}]\!]_3, [\![1]\!]_3)$. $(\vec{t}_{2,1} = 0)$
5. Alice and Bob compute their respective shares of the following 3 shared values:
   - $[\![\vec{c}_0]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{c}_1]\!]_2 = \sum_{j=0}^{0}[\![\vec{t}_{1,j}]\!]_2 = [\![\vec{t}_{1,0}]\!]_2 = [\![1]\!]_2$
   - $[\![\vec{c}_2]\!]_2 = \sum_{j=0}^{1}[\![\vec{t}_{2,j}]\!]_2 = [\![\vec{t}_{2,0}]\!]_2 + [\![\vec{t}_{2,1}]\!]_2 = [\![1]\!]_2$
6. Alice and Bob compute their respective shares of the following 3 shared values:
   - $[\![\vec{b}_0]\!]_2 = [\![\vec{u}_0]\!]_2 + [\![\vec{v}_0]\!]_2 + [\![\vec{c}_0]\!]_2 = [\![1]\!]_2 + [\![1]\!]_2 + [\![0]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{b}_1]\!]_2 = [\![\vec{u}_1]\!]_2 + [\![\vec{v}_1]\!]_2 + [\![\vec{c}_1]\!]_2 = [\![0]\!]_2 + [\![1]\!]_2 + [\![1]\!]_2 = [\![0]\!]_2$
   - $[\![\vec{b}_2]\!]_2 = [\![\vec{u}_2]\!]_2 + [\![\vec{v}_2]\!]_2 + [\![\vec{c}_2]\!]_2 = [\![1]\!]_2 + [\![0]\!]_2 + [\![1]\!]_2 = [\![0]\!]_2$

We now proceed to prove the security and correctness of both protocols, starting by $\Pi_{\mathtt{BD}}$.

Theorem 7. *Protocol $\Pi_{\mathtt{BD}_\ell}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{BD}_\ell}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** Let $\vec{v} \in \mathbb{Z}_2^\ell$ and $\vec{u} \in \mathbb{Z}_2^\ell$ be the binary expansions of $(-[\![\beta]\!]_{2^\ell}^A \pmod{2^\ell})$ and $[\![\beta]\!]_{2^\ell}^B$, respectively, and $\vec{c'} \in \mathbb{Z}_2^{\ell+1}$ be the carry bit vector generated when computing $\alpha = [\![\beta]\!]_{2^\ell}^B + (-[\![\beta]\!]_{2^\ell}^A \pmod{2^\ell})$. Based on this, we have $\alpha = \vec{c}_\ell \cdot 2^\ell + \sum_{i=0}^{\ell-1}(\vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i) \cdot 2^i$, where clearly $\vec{c}_\ell \in \{0,1\}$ and $0 \leq \sum_{i=0}^{\ell-1}(\vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i) \cdot 2^i < 2^\ell$, which implies that $\alpha \equiv \beta \equiv \sum_{i=0}^{\ell-1}(\vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i) \cdot 2^i \pmod{2^\ell}$. This means that $\vec{b}_i = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i$, for $0 \leq i \leq \ell - 1$, is the binary expansion of β. Thus, if the vector $\vec{c}$ computed by the protocol is equal to $\vec{c'}$, from position 0 to position $\ell - 1$, then based on step 6 of $\Pi_{\mathtt{BD}}$, we can see that the protocol's output would in fact be the desired one. Because of this, we proceed to prove that $\vec{c}_i = \vec{c'}_i$ for $0 \leq i \leq \ell - 1$.

The set of Boolean equations that define the value of $\vec{c'}$ are the following:

$$\vec{c'}_0 = 0 \text{ and } \vec{c'}_i = \bigoplus_{j=0}^{i-1} \vec{t'}_{i,j}, \text{ for } 1 \leq i \leq \ell - 1$$

$$\vec{t'}_{i,j} = \vec{g'}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x'}_k, \text{ for } 0 \leq j < i \leq \ell - 1$$

$$\vec{g'}_i = \vec{a}_i \wedge \vec{d}_i, \text{ for } 0 \le i \le \ell - 1$$
$$\vec{x'}_i = \vec{a}_i \oplus \vec{d}_i, \text{ for } 0 \le i \le \ell - 1$$

After quickly analyzing the protocol, we can see that $\Pi_{\mathtt{BD}_\ell}$ computes the bit vector $\vec{c}$ according to the following equations:

$$\vec{c}_0 = 0 \text{ and } \vec{c}_i = \bigoplus_{j=0}^{i-1} \vec{t}_{i,j}, \text{ for } 1 \le i \le \ell - 1$$
$$\vec{t}_{i,j} = \begin{cases} 1, & \vec{h}_{i,j} = i - j \\ 0, & \text{otherwise} \end{cases}, \text{ for } 0 \le j < i \le \ell - 1$$
$$\vec{h}_{i,j} = \vec{g}_j + \sum_{k=j+1}^{i-1} \vec{x}_k, \text{ for } 0 \le j < i \le \ell - 1$$
$$\vec{x}_i = \vec{a}_i \oplus \vec{d}_i, \text{ for } 0 \le i \le \ell - 1$$
$$\vec{g}_i = \vec{a}_i \wedge \vec{d}_i, \text{ for } 0 \le i \le \ell - 1$$

Looking at these equations we can see that $\vec{x}_i, \vec{g}_i \in \{0, 1\}$ for $0 \le i \le \ell - 1$, which implies that $0 \le \vec{h}_{i,j} \le i - j$ and $\vec{h}_{i,j} = i - j$ iff $\vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$, for $0 \le j < i \le \ell - 1$. Based on this, we can see that $\vec{t}_{i,j} = \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$ for $0 \le j < i \le \ell - 1$. Thus, looking at the equation that dictates the value of $\vec{c}$, we can conclude that $\vec{c}_i = \vec{c}'_i$ for $0 \le i \le \ell - 1$. Therefore, we have that $\Pi_{\mathtt{BD}_\ell}$ is correct.

Security: The rationale used in $\Pi_{\mathtt{EEQ}^*_{N,M}}$'s security proof can be used to prove $\Pi_{\mathtt{BD}_\ell}$'s security. □

Theorem 8. *The protocol $\Pi'_{\mathtt{BD}_\ell}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{BD}_\ell}$ against semi-honest adversaries in the commodity-based model.*

Proof. **Correctness:** By looking at the descriptions for protocols $\Pi'_{\mathtt{BD}_\ell}$ and $\Pi_{\mathtt{BD}_\ell}$, we can see that the only difference between the two is 4. So if we prove that the values of $\vec{t}_{i,j}$ in $\Pi_{\mathtt{BD}_\ell}$ and $\Pi'_{\mathtt{BD}_\ell}$ respect the same equation, for $0 \le j < i \le \ell - 1$, from the correctness proof of $\Pi_{\mathtt{BD}_\ell}$, we have that $\Pi'_{\mathtt{BD}_\ell}$ is also correct. Again by looking at $\Pi'_{\mathtt{BD}_\ell}$'s description and by the formal definition of $\mathcal{F}_{\mathtt{EEQ}}$ we can see that the values of $\vec{t}_i, j$, in the description for $\Pi'_{\mathtt{BD}_\ell}$, are defined by the following equation:

$$\vec{t}_{i,j} = \begin{cases} 1, & \vec{h}_{i,j} = i - j \\ 0, & \text{otherwise} \end{cases}, \text{ for } 0 \le j < i \le \ell - 1$$

□

This equation also defines the values of $\vec{t}_{i,j}$ in the description for $\Pi_{\mathtt{BD}_\ell}$, for $0 \le j < i \le \ell - 1$. Thus, we can conclude that $\Pi'_{\mathtt{BD}_\ell}$ is correct.

Security: The same ideas used to prove the security of $\Pi_{\mathtt{EEQ}}$ can be applied to prove the security of $\Pi'_{\mathtt{BD}}$.

3.6 Efficiency Comparison

In order to provide meaningful comparisons, we restrict ourselves to protocols with the following characteristics: only two parties are required to execute the protocols; the protocols provide unconditional security to the parties executing them; the protocols are perfectly correct in the presence of computationally unbounded adversaries. With these restrictions in place, we compare computational complexity, bit transfer complexity, and the number of communication rounds.

All the previously published works considered in our comparisons measure computational complexity and the number of bits transferred by the number of times a secure multiplication protocol is invoked. For the sake of our comparison, we assume that the SOT (in the case of our protocol) and multiplication triples (in the case of other protocols) are pre-computed. We believe this assumption is reasonable since the pre-computation of both primitives has similar efficiency (for example, when using OT extensions or a trusted initializer). Thus, we do not include the cost of such pre-computation of the underlying primitives in our analysis. Considering pre-computed multiplications, one invocation of the multiplication protocol transfers $O(l)$ bits during the online phase, has computational complexity $O(l^2)$, and requires one communication round, where l is the number of bits necessary to represent the inputs of the multiplication protocol - usually a ring or field element.

Table 3.1: Protocol Efficiency Comparison

Protocol	[13]	[21]	[15]	$\Pi_{\texttt{EEQ}}$	$\Pi^{\mathcal{P}}_{\texttt{EEQ}}$
Preprocessing Phase					
Communication	$O(\ell^2)$	$O(\ell^2)$	$O(\ell^2)$	$\perp$	$O(\ell\log(\ell))$
Computation	$O(\ell^3)$	$O(\ell^3)$	$O(\ell^3)$	$\perp$	$O(\ell\log(\ell))$
Rounds	$O(1)$	9	2	$\perp$	1
Online Phase					
Communication	$O(\ell)$	$O(\ell)$	$O(\ell^2)$	$O(\ell\log(\ell))$	$O(\ell)$
Computation	$O(\ell^2)$	$O(\ell^2)$	$O(\ell^3)$	$O(\ell\log(\ell))$	$O(\ell)$
Rounds	2	2	6	2	2
Protocol	[16]	[15]	[21]	$\Pi_{\texttt{BLT}}$	$\Pi^{\mathcal{P}}_{\texttt{BLT}}$
Preprocessing Phase					
Communication	$O(\ell^2)$	$O(\ell^2)$	$O(\ell^2/log(\ell))$	$\perp$	$O(\ell\log(\ell)\log(\log(\ell)))$
Computation	$O(\ell^3)$	$O(\ell^3)$	$O(\ell^3/log(\ell))$	$\perp$	$O(\ell\log(\ell)\log(\log(\ell)))$
Rounds	6	2	3	$\perp$	1
Online Phase					
Communication	$O(\ell^2)$	$O(\ell^2)$	$O(\ell^2/log(\ell))$	$O(\ell\log(\ell)\log(\log(\ell)))$	$O(\ell\log(\ell))$
Computation	$O(\ell^3)$	$O(\ell^3)$	$O(\ell^3/log(\ell))$	$O(\ell\log(\ell)\log(\log(\ell)))$	$O(\ell\log(\ell))$
Rounds	3	6	4	3	3
Protocol	[15]	[19]	[17]	$\Pi_{\texttt{BD}}$	$\Pi_{\texttt{BD}'}$
Overall					
Communication	$O(\ell^2\cdot\log(\ell))$	$O(c\cdot\ell\cdot\log^{*(c)}(\ell))$	$O(\ell^2)$	$O(\ell^3)$	$O(\ell^2\log(\ell)\log(\log(\ell)))$
Computation	$O(\ell^3\cdot\log(\ell))$	$O(c\cdot\ell^2\cdot\log^{*(c)}(\ell))$	$O(\ell^3)$	$O(\ell^3)$	$O(\ell^2\log(\ell)\log(\log(\ell)))$
Rounds	(E) 25	(E) $23+c$	(E) 12	2	3

(E) Specifies that a protocol only runs in expected constant rounds.

Source: Created by author.

Our results are presented in Table 3.1. We once more emphasize that our comparisons and conclusions are taken with respect to other 2-party protocols present in the literature, with unconditional perfect security.

Our protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$ is the only 2-party protocol for secure equality test with unconditional, perfect security that communication and computational complexities linear in ℓ. $\Pi_{\texttt{EEQ}}$ is the only protocol that has 2 rounds in the online phase, without a pre-processing phase - at the cost of increasing the computation and communication complexities to $O(\ell\log(\ell))$.

Our protocols for private comparison, $\Pi_{\texttt{BLT}}$ and $\Pi^{\mathcal{P}}_{\texttt{BLT}}$, are the only protocols that have three rounds with communication and computational complexities equal to $O(\ell\log(\ell)\log(\log(\ell)))$.

Finally, our protocols for bit decomposition $\Pi_{\texttt{BD}}$ and $\Pi'_{\texttt{BD}}$ are the only ones in the literature that have three or less rounds.

Chapter 4

Conclusions

Protocols for secure equality tests, comparison, and bit decomposition are important research areas within cryptographic protocols, with their relevance becoming clearer because of the crucial role they play in privacy-preserving machine learning protocols [10, 7, 11].
In this work, we have shown how to build novel protocols for all these functions, improving on the efficiency of previous published solutions, specially on the number of communications of rounds betweem the two parties. In order to do this, we make use of a new cryptographic primtive called Shared Oblivious Transfer, which we believe can be of independent interest, that can be seen as an extension of the widely known Oblivious Transfer primitive.
These new results lead to other interesting questions that are not answered in this work and that could be the focus of future ones. The protocols for equality, comparison and bit-decomposition are prove secure against semi-honest adversaries, that is, adversaries that follow the described protocol. A natural question is if we can modify these protocol to make them secure against adversaries that behave arbitrarily while sacrificing minimal performance.
We saw in this work how Shared OTs can be used to improve the efficiency of protocols for multiple interesting and useful functionalities. Another natural question that this leads to is if this new cryptographic primitive can be applied to improve the efficiency of other interesting functionalities, such as set intersection of secrets and elementwise comparison. Future efforts could be dedicated to studying other functionalities that could benefit from Shared OT.

Bibliography

[1] Donald Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 446–455, New York, NY, USA, 1997. Association for Computing Machinery.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[3] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 486–497, New York, NY, USA, 2007. Association for Computing Machinery.

[4] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[6] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 11–19, New York, NY, USA, 1988. Association for Computing Machinery.

[7] Martine de Cock, Rafael Dowsley, Anderson CA Nascimento, and Stacey C Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2015.

[8] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 280–300, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[9] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, com-

parison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 285–304, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[10] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson CA Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2):217–230, 2017.

[11] Martine De Cock, Rafael Dowsley, Anderson CA Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. High performance logistic regression for privacy-preserving genome analysis. *BMC Medical Genomics*, 14(1):1–18, 2021.

[12] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Transactions on Information Forensics and Security*, 7(3):1053–1066, June 2012.

[13] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sub-linear online complexity. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 645–656, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[14] Anderson C. A. Nascimento, Joern Mueller-Quade, Akira Otsuka, Goichiro Hanaoka, and Hideki Imai. Unconditionally non-interactive verifiable secret sharing secure against faulty majorities in the commodity based model. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security*, pages 355–368, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[15] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, pages 343–360, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] Tord Reistad. Multiparty comparison - an improved multiparty protocol for comparison of secret-shared values. pages 325–330, 01 2009.

[17] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, pages 245–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[18] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Technical report, 1999.

[19] Tomas Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, pages 357–371, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[21] Ching-Hua Yu. Sign modules in secure arithmetic circuits. Cryptology ePrint Archive, Report 2011/539, 2011. `https://ia.cr/2011/539`.