IACR Communications in Cryptology
Vol. 0, No. 0, 30 pages.

Shared OT and Its Applications

Lucas Piske[1], Jeroen van de Graaf[2], Anderson C. A. Nascimento[3] and Ni Trieu[1]

[1] Arizona State University, United States
[2] Universidade Federal de Minas Gerais, Brazil
[3] Visa (United States), United States

Abstract. We present unconditionally perfectly secure protocols in the semi-honest setting for several functionalities: (1) private elementwise equality; (2) private bitwise integer comparison; and (3) bit-decomposition. These protocols are built upon a new concept called Shared Oblivious Transfer (Shared OT). Shared OT extends the one-out-of-N String OT by replacing strings with integers modulo M and allowing additive secret-sharing of all inputs and outputs. These extensions can be implemented by simple local computations without incurring additional OT invocations. We believe our Shared OT may be of independent interest.
Our protocols demonstrate the best round, communication, and computational complexities compared to all other protocols secure in a similar setting. Moreover, all of our protocols involve either 2 or 3 rounds.

Keywords: unconditional security · secure comparison · equality test · OT.

1 Introduction

Secure two-party computation enables two parties, typically denoted as Alice and Bob, to compute a function f using their individual private inputs x_A and x_B, while ensuring that only the function output $f(x_A, x_B)$ is revealed, without disclosing any further information. Garbled circuits offer a generic method for implementing secure two-party computation, allowing the evaluation of any Boolean circuit securely with a constant number of communication rounds, without revealing any intermediate information beyond the output. Initially introduced by Yao [Yao86] and extended to the multi-party scenario by Beaver, Micali, and Rogaway [BMR90], garbled circuits have since seen improvements in efficiency through various garbling schemes, with several implementations available in the literature.

General solutions for secure two-party (and multiparty) computation can often be inefficient. Thus, the research community has focused on finding efficient methods to evaluate specific functions. In this work, we propose customized protocols for three popular functions: integer equality test, integer comparison and bit-decomposition. We consider the unconditionally perfectly secure setting, which has received less attention recently [GLS19, YNKM24] compared to the computationally/probabilistic one [Yao82, DSZ15, RR21, DILO22, DDG+23, HKN24]. With the advent of quantum computing, however, the importance of this model is growing. Unlike computational models, it does not rely on assumptions about the adversary's computational limitations, ensuring protocols remain secure against both current and future technological advancements.

Our work has significant practical applications in scenarios requiring long-term security and resistance to adversaries with unlimited computational power. These include privacy-preserving genomic analysis of sensitive DNA data, secure e-voting systems that

E-mail: lpiske@asu.edu (Lucas Piske), jeroenvandegraaf@proton.me (Jeroen van de Graaf), annascim@visa.com (Anderson C. A. Nascimento), ntrieu1@asu.edu (Ni Trieu)

guarantee tamper-proof elections, and financial systems supporting sealed-bid auctions and private transactions resilient to quantum attacks. Additionally, critical domains such as healthcare and national security can leverage these protocols for collaborative analysis of sensitive inputs, such as patient records or strategic intelligence, without compromising confidentiality. Applications like private information retrieval (PIR) and secure IoT data aggregation further highlight the value of unconditional security in protecting privacy and data integrity in highly sensitive environments.

1.1 Contribution

This paper introduces novel approaches for evaluating functions for integer equality tests, integer comparisons, and bit-decomposition. The context involves two parties, Alice and Bob, along with an adversary possessing unconditional computational power but behaving in a semi-honest manner. All the proposed protocols require only a small constant number of rounds (either 2 or 3) to execute, are perfectly correct, and offer perfect security against any computationally unbounded semi-honest adversaries, assuming a trusted third party generates the correlated randomness required by the protocols.

Compared to existing protocols in the same setting, our proposed protocols demonstrate better efficiency, as shown in Table 1. For example, we reduce the complexity of the secure integer equality from $O(\ell^2)$ in existing protocols [LT13, Yu11, NO07] to $O(\ell \log(\ell))$, where ℓ denotes the bit-length of the protocol inputs.

To construct these protocols, we introduce a new variation of the widely known OT protocol, termed Shared OT (SOT). We show that SOT can be easily implemented utilizing a single instance of 1-out-of-N OT over elements modulo M. Our SOT takes one round to execute, is perfectly correct, and is perfectly secure against any computationally unbounded malicious adversaries.

1. **Secure Integer Equality.** The functionality takes two additively shared ℓ-bit elements as input and outputs an additively shared element modulo 2, which indicates if the two elements are the same. To implement this functionality, we propose two protocols: $\Pi_{\texttt{EEQ}}$ and $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$. Both require 2 online rounds for execution. However, $\Pi_{\texttt{EEQ}}$ has online computation and communication complexity of $O(\ell \log(\ell))$, whereas $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$ has complexity of $O(\ell)$, making it more efficient. Nevertheless, $\Pi^{\mathcal{P}}_{\texttt{EEQ}}$ requires a pre-processing round with computation and communication complexity of $O(\ell \log(\ell))$, while $\Pi_{\texttt{EEQ}}$ does not require pre-processing.

2. **Secure Integer Comparison.** The functionality takes the binary representation of two ℓ-bit elements, a and b, additively shared modulo 2 as input, and outputs an additively shared element modulo 2 indicating whether $a < b$. We propose two protocols: $\Pi_{\texttt{BLT}}$ and $\Pi^{\mathcal{P}}_{\texttt{BLT}}$. Both protocols require 3 online rounds to be executed. However, $\Pi_{\texttt{BLT}}$ has online computational and communication complexity of $O(\ell \log(\ell) \log(\log(\ell)))$, whereas $\Pi^{\mathcal{P}}_{\texttt{BLT}}$ has complexity of $O(\ell \log(\ell))$, making it more efficient. Nonetheless, $\Pi^{\mathcal{P}}_{\texttt{BLT}}$ necessitates a pre-processing round with computational and communication complexity of $O(\ell \log(\ell) \log(\log(\ell)))$, whereas $\Pi_{\texttt{BLT}}$ does not require pre-processing.

3. **Secure bit-Decomposition.** The functionality takes a single additively shared ℓ-bit element β as input and outputs the binary representation of β additively shared modulo 2. We propose two protocols for this purpose: $\Pi_{\texttt{BD}}$ and $\Pi'_{\texttt{BD}}$. Protocol $\Pi_{\texttt{BD}}$ requires 2 overall rounds and has computational and communication complexity of $O(\ell^3)$. Protocol $\Pi'_{\texttt{BD}}$ necessitates 3 overall rounds and has computational and communication complexity of $O(\ell^2 \log(\ell))$.

1.2 Related Work

Private integer equality, comparison, and bit-decomposition protocols have garnered significant interest due to their wide applications such as privacy preserving machine learning [BIK+17, NWKT24] and secure advertising [vBP24, MMT+24]. These protocols can be categorized based on two criteria: whether the computational power is available to the adversary considered in the security proofs and whether the protocol ensures perfect correctness. Perfect correctness denotes that a protocol returns the correct result with a probability equal to one. Regarding the computational power of the adversary, it means whether the adversary has unbounded computational power or is limited to a polynomial-time algorithm.

Several works, such as [Yao82, BK04, Veu12, DSZ15, DDG+23, HKN24], present constructions that assume the adversary has polynomial computational power, resulting in computationally secure solutions. Additionally, constructions with non-perfect correctness (also known as probabilistic correctness) have been proposed in [SCJ13, YY12, LT13]. In this paper, for the sake of comparison, we solely consider related works that fulfill specific criteria: they must be *secure against unconditional adversaries, provide perfect correctness, require only $O(1)$ communication rounds between the parties, and work in the two-party setting.* Protocols ensuring unconditional security typically assume the existence of an ideal functionality or primitive that enables the computation of non-trivial functions. For instance, [DFK+06, Rei09, NO07] assume the existence of an unconditional secret shared multiplication protocol. Alternatively, [LT13, RT10, Yu11] relies on the existence of an arithmetic black-box (ABB) [DN03] or OT. The complexity of such protocols is typically assessed in terms of the number of invocations to the cryptographic primitive. For example, the count may include invocations to a secret shared multiplication functionality, and the round complexity is measured by the number of sequential invocations. Additionally, some protocols are divided into two phases, termed offline and online.

Unconditional Secure Equality. The first relevant work to our study is [DFK+06]. Later, [NO07] improved this by achieving $O(\ell)$ multiplications and reducing communication rounds to 8. The state-of-the-art protocols are [LT13] and [Yu11], both requiring $O(\ell)$ shared multiplications in total and $O(1)$ shared multiplications with 2 online rounds. Their difference lies in offline rounds: [Yu11] requires 9, while [LT13] achieves $O(1)$.

Unconditional Secure Comparison. The first solution to the private comparison problem relevant to our scenario is presented in [DFK+06], which requires a linear number of shared multiplications. Nishide and Ohta later reduced the round complexity to 2 offline and 6 online rounds in [NO07], while still requiring a linear number of shared multiplications. Reistad [Rei09] proposed a solution with similar efficiency to [NO07]. Subsequently, [Yu11] introduced a protocol using a sublinear number of shared multiplications, achieving $O(\ell \log(\ell))$ shared multiplications with a total of 7 rounds, including 3 offline and 4 online.

Unconditional Secure Bit-decomposition. The initial solution to the bit-decomposition problem satisfying our previously stated requirements was also introduced in [DFK+06]. Subsequently, a protocol outlined in [NO07] enhanced the expected number of rounds needed for bit-decomposition while maintaining the same asymptotic number of shared multiplication executions as in [DFK+06]. Toft later introduced a new protocol in [Tof09], which improved upon every efficiency aspect of prior results. This protocol requires an almost-linear amount of shared multiplications and $23 + c$ expected communication rounds, where $c > 1$ can be adjusted to achieve a trade-off between communication and round efficiency (increasing c results in fewer data being transferred but requires more rounds). Additionally, Toft and Reistat proposed an even more efficient bit-decomposition proto-

col in [RT10], requiring a linear number of shared multiplications and 12 rounds to be performed. However, unlike previous results, this protocol does not offer perfect security.

2 Preliminaries

2.1 Notation

In all proposed protocols, the input and output are "subtractively" shared elements modulo M, where $M \in \mathbb{Z}^{\geq 2}$. Let $M \in \mathbb{Z}^{\geq 2}$ and $a \in \mathbb{Z}_M$. We use $[\![a]\!]_M$ to denote the subtractive sharing modulo M of a. Alice's and Bob's respective shares of $[\![a]\!]_M$ are denoted as $[\![a]\!]_M^A \in \mathbb{Z}_M$ and $[\![a]\!]_M^B \in \mathbb{Z}_M$, and we have $[\![a]\!]_M^B - [\![a]\!]_M^A = a \pmod M$. Although we use the term additive secret sharing interchangeably with "subtractively" shared elements, as the two schemes are fundamentally equivalent, we opted for subtractive shares because they resulted in cleaner functionality and protocol definitions. However, there is no reason why the protocols presented could not be adapted to use additive secret shares.

Since this work focuses exclusively on two-party protocols, with the parties consistently referred to as Alice and Bob, we only need notation to represent the shares of these two parties.

The first convention we introduce is to only explicitly display an expression's modulo if it is not explicit 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 $\vec{a} \in \mathbb{Z}_2^{\lceil \log_2(M) \rceil}$, where $\vec{a}$ is the binary expansion of a. We index the vectors in this paper starting from 0, and the least significant bit (LSB) of a binary expansion $\vec{a}$ is $\vec{a}_0$. Also, when $\vec{a} \in \mathbb{Z}_2^{\lceil \log_2(M) \rceil}$ is the binary expansion of $a \in \mathbb{Z}_M$ and $\vec{b} \in \mathbb{Z}_2^{\lceil \log_2(M) \rceil}$ is the binary expansion of $b \in \mathbb{Z}_M$, we use $\vec{a} < \vec{b}$ to denote $a < b$.

Additionally, when adding a scalar modulo M to a vector of elements modulo M, let $u \in \mathbb{Z}_M$ 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 $i \in 0, 1, \ldots, N-1$.

We define $\mathsf{One}_n(i, a)$, where $0 \leq i \leq n-1$ and $0 \leq a \leq n-i$, as a function which outputs a vector $\vec{v} \in \mathbb{Z}_2^n$ containing a number of one, starting from position i, and containing zero 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

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

We define $\mathsf{cshift}_N(\vec{v}, x)$, where $\vec{v} \in \mathbb{Z}^N$ and $x \in \mathbb{Z}_N$, as a function which 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 $\mathsf{cshift}_4(\vec{v}, 3) = (2, 3, 4, 1)$. More precisely, 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 Unconditional Security

Unconditional security refers to a level of security that remains secure regardless of the computational power of an adversary. Unlike computational security, which relies on assumptions about computational limitations, unconditional security ensures defense against all possible attacks, including those involving unlimited computational resources. This becomes particularly relevant in the context of quantum computing.

In this work, we achieve unconditional security because our protocols rely solely on Oblivious Transfer (OT) as the cryptographic primitive. As demonstrated in [Riv], OT

can be performed with unconditional security, provided a trusted third party generates and distributes correlated randomness to both parties involved in the protocol.

2.3 Commodity-Based Cryptography

First introduced by Beaver in [Bea97], Commodity-Based Cryptography is a paradigm used to design efficient secure multi-party computation protocols. In this paradigm, there are both servers and clients, with servers assisting clients in executing cryptographic primitives. The level of corruption tolerated within a subset of servers may differ between protocols, and the same is true for the clients.

The Commodity-Based paradigm not only defines the set of players but also restricts what information these players have about each other and how they interact. This is what sets this paradigm apart 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 imposing these restrictions, the paradigm offers several advantageous properties. Clients are not required to provide sensitive data to servers, minimizing the trust that clients need to have in servers. Additionally, the paradigm is scalable since multiple servers can be employed simultaneously. Utilizing many servers also enhances confidence that at least a portion of the material provided by the servers is secure and correct.

In this work, we assume that a single trusted server generates random oblivious transfers locally and distributes them to the parties that will execute the desired protocol. By working in the Commodity-Based Cryptography paradigm and assuming the existence of this trusted third party, the two other parties can execute the proposed protocols with perfect security against any computationally unbounded semi-honest adversaries.

2.4 Oblivious Transfer (OT)

OT is a widely used cryptographic primitive essential for secure computation, first introduced by Rabin [Rab81], and later modified into another widely used variant called 1-out-of-2 Oblivious Transfer in [EGL85]. In 1-out-of-2 OT, a sender having two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. The receiver securely learns x_b without gaining any information about x_{1-b}, ensuring privacy. Simultaneously, the sender remains oblivious to the value of b.

In this paper, we work with a slightly different variant of OT, called 1-out-of-N OT (OT) over elements modulo M. In 1-out-of-N OT over elements modulo M, we have a party named Bob providing a choice index c modulo N as input, while a party named Alice provides an options vector $\vec{m} \in \mathbb{Z}_M^N$. Bob receives $\vec{m}_c$ as output, while Alice receives nothing. In this work, the vector $\vec{m}$ consists of elements modulo M. While an OT variant [KK13, KKRT16], where the sender transmits messages as bit-strings, is popular in the literature, we found it more convenient to work with messages modulo M when constructing our new proposed protocols. We formally present the functionality for 1-out-of-N OT over elements modulo M as below.

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

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

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

By generalizing the OT protocol proposed by Rivest, in [Riv], in a straightforward manner, it is possible to implement a protocol that fulfills the description for $\mathcal{F}_{\texttt{OT}_M^N}$ while providing perfect security in the malicious setting, when assuming the existence of a trusted initializer. Thus, our 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 $O(\log_2(N) + N \cdot \log_2(M))$. Consequently, this yields an equivalent amount of transferred bits and computation to be performed by the Trusted Initializer.

3 Shared OT

3.1 Functionality

We now present a novel variant of Oblivious Transfer (OT), termed Shared OT, which extends the 1-out-of-N OT [KK13, KKRT16] to operate over elements modulo M, enabling the receiver to select one of the sender's inputs within this modular domain. Our extension introduces two significant differences from traditional OT: (1) the selection index input is additively shared between the two parties, Alice and Bob, and (2) the output is also additively secret shared between them.

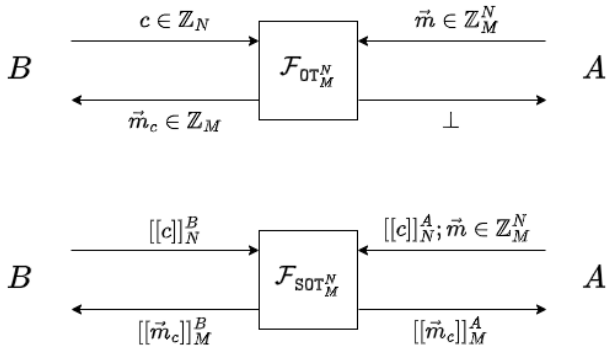

Figure 1: Difference between input and output structure of 1-out-of-N binary OT and 1-out-of-N SOT over elements modulo M.

Figure 1 illustrates the differences in input and output structures between 1-out-of-N Bit OT and 1-out-of-N Shared OT (SOT) over elements modulo M. In Shared OT, Alice inputs an options vector $\vec{m}$, which contains N elements modulo M. Alice and Bob also input their respective shares of an index c modulo N. The output of SOT is the additive shares modulo M of $\vec{m}_c$ to both Alice and Bob, ensuring that neither party learns additional information. Note that the options vector $\vec{m}$ is not shared between the parties; only Alice knows its value. We formally present the functionality of SOT as follows.

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

- Upon receiving a message $(\texttt{choose}, [\![c]\!]_N^B)$ from Bob: Ignore any subsequent $(\texttt{choose}, [\![c]\!]_N^B)$ messages. If $[\![c]\!]_N^B \notin \mathbb{Z}_N$, then send (`invalid input`) to both parties and halt. Store $[\![c]\!]_N^B$ and send the public delayed message (`chosen`) to Alice.
- Upon receiving a message (`sample share`) from Alice: Ignore any subsequent messages (`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 $(\texttt{propose}, [\![c]\!]_N^A, \vec{m})$ from Alice: Ignore any subsequent $(\texttt{propose}, [\![c]\!]_N^A, \vec{m})$ 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 (`invalid input`) to both parties and halt. If it is the case, reconstruct $c \in \mathbb{Z}_N$ from shares $([\![c]\!]_N^A, [\![c]\!]_N^B)$, and send $[\![\vec{m}_c]\!]_M^B = \vec{m}_c + [\![\vec{m}_c]\!]_M^A \pmod M$ to Bob.

3.2 Protocol

We implement the $\Pi_{\mathtt{SOT}}$ protocol for Shared Oblivious Transfer (SOT) using a single instance of $\mathcal{F}_{\mathtt{OT}_M^N}$ and performing only basic local operations (such as cyclic shifts of the vector $\vec{m}$, sampling, and addition modulo an integer) on the protocol's inputs. In more detail, the protocol proceeds as follows:

- Alice and Bob execute an instance of $\mathcal{F}_{\mathtt{OT}_M^N}$. Bob provides $[\![c]\!]_N^B$ as the choice index, while Alice provides the input vector $\vec{m}'$, defined as $\vec{m}' = \mathsf{cshift}_N(\vec{m}, [\![c]\!]_N^A) + u$, where $u \in_R \mathbb{Z}_M$ is randomly sampled by Alice.
- As a result of executing $\mathcal{F}_{\mathtt{OT}_M^N}$, Bob receives $\vec{m'}_{[\![c]\!]_N^B}$ and Alice retains u, since she sampled it.
- These two values, $\vec{m'}_{[\![c]\!]_N^B}$ and u, serve as the respective outputs for Alice and Bob in the $\Pi_{\mathtt{SOT}}$ protocol.

From this brief description, we can explain the main arguments behind the correctness and security of $\Pi_{\mathtt{SOT}}$.

Correctness. Given the inputs provided to $\mathcal{F}_{\mathtt{OT}}$, during the execution of $\Pi_{\mathtt{SOT}}$, Bob receive $\vec{m'}_{[\![c]\!]_N^B}$. Based on the definition of cshift_N and the construction of $\vec{m}'$, this implies that Bob receives

$$\vec{m'}_{[\![c]\!]_N^B} = \vec{m}_{[\![c]\!]_N^B - [\![c]\!]_N^A \pmod N} + u = \vec{m}_c + u \pmod M$$

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 $\Pi_{\mathtt{SOT}}$.

Security. 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 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 of how the mapping between the two types of inputs can be performed is found in the security proof for $\Pi_{\mathtt{SOT}}$ further along in this section, along with the corresponding security theorem.

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

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

Parameters:

- The ideal functionality $\mathcal{F}_{\mathtt{OT}^N_M}$ described in Section 2.4
- The function cshift in Section 2.1

Inputs:

- Bob inputs $[\![c]\!]^B_N$.
- Alice inputs $\vec{m} \in \mathbb{Z}_M^N$ 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 the description of this protocol and assuming the correctness of its security proof in Theorem 1, we can conclude that, despite being a more flexible primitive, Shared OT is as efficient as OT while also being secure in the malicious setting. The protocol $\Pi_{\mathtt{SOT}}$ requires the same number of rounds and transfers the same amount of bits between the two parties as the protocol implementing $\mathcal{F}_{\mathtt{OT}^N_M}$, with negligible computational overhead. Additionally, note that our primitive can be pre-computed in the trusted initializer model as proposed by Rivest [Riv].

Theorem 1. *Protocol $\Pi_{\mathtt{SOT}^N_M}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{SOT}^N_M}$ against malicious adversaries in the $\mathcal{F}_{\mathtt{OT}}$-hybrid model.*

Proof. We formally present the security of our SOT protocol in Theorem 1. We prove it by showing 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 in the presence of a malicious adversary $\mathcal{A}$. Mathematically,

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

where $\mathcal{S}$ is the simulator and $\mathcal{E}$ is the environment. 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, meaning the simulator $\mathcal{S}$ can read her inputs ($[\![c]\!]^A_N$ 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 interactions between $\mathcal{S}$ and $\mathcal{A}'$ replicate those that Alice has with other parties (e.g., $\mathcal{F}_{\mathtt{OT}}$ and the environment $\mathcal{E}$). The behavior of the simulator is described as follows:

Simulation Description

1. The environment $\mathcal{E}$ delivers the inputs $[\![c]\!]^A_N$ and $\vec{m}$ to the simulator $\mathcal{S}$, this action activates $\mathcal{S}$. Upon its activation, $\mathcal{S}$ performs two actions. First, $\mathcal{S}$ delivers $[\![c]\!]^A_N$ and $\vec{m}$ to $\mathcal{A}'$. Second, $\mathcal{S}$ sends a message (`sample share`) to $\mathcal{F}_{\mathtt{SOT}}$, 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 $\vec{v} \notin \mathbb{Z}_M^N$, 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 $\vec{v} \in \mathbb{Z}_M^N$, $\mathcal{S}$ computes $\vec{v}' = \vec{v} - [\![\vec{m}_c]\!]_M^A \pmod M$ and sends $(\texttt{propose}, 0, \vec{v}')$ to $\mathcal{F}_{\texttt{SOT}}$. Note that this causes Bob to receive $\vec{v}_{[\![c]\!]_N^B}$ 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 poth 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 $\vec{v} \notin \mathbb{Z}_M^N$, which again causes poth 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 $\vec{v} \in \mathbb{Z}_M^N$. In the hybrid world, this will cause Alice and Bob to execute an $\mathcal{F}_{\texttt{OT}}$ where the selection index is $[\![c]\!]_N^B$ 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 $[\![c]\!]_N^B$, $[\![c]\!]_N^A := 0$ 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 demonstrate 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 demonstrate 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 interactions 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.

Simulation Description

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 (`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.

Indistinguishability

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.

Part I: On the Simulation

- 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 (`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.

□

4 Applications

In this section, we start by introducing a relaxed variant of equality test, called Element Equality* and denoted as $\texttt{EEQ}^*$. Utilizing this concept as a building block, we construct secure protocols for integer equality, comparison, and bit-decomposition.

4.1 Element Equality*

In this section, we define two slightly different cryptography functionalities for determining equality $\mathcal{F}_{\texttt{EEQ}^*_{N,M}}$ and $\mathcal{F}_{\texttt{EEQ}_N}$, and their respective protocols. Intuitively speaking, it is useful to think of these functionalities as privately computing the following functions:

$$\texttt{EEQ}_N(a,b) = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases} \quad \text{and} \quad \texttt{EEQ}^*_{N,M}(a,b) = \begin{cases} 0 \text{ if } a = b \\ i \neq 0 \text{ if } a \neq b \end{cases}$$

Here, $a, b \in \mathbb{Z}_N$ and i is an integer such that $0 < i < M$, where M is a protocol parameter. Our functionalities, however, will be taking subtractive shares of a and b as input and outputting subtractive secret shares of the result. The functionalities will be formally defined in the coming subsections.

We first introduce a protocol for evaluating $\texttt{EEQ}^{\star}_{N,M}$ and in the subsequent subsection, we demonstrate how a protocol for $\texttt{EEQ}_N$ can be derived straightforwardly from the previous one for $\texttt{EEQ}^{\star}_{N,M}$. Below, we present the functionality corresponding to computing the function $\texttt{EEQ}^{*}_{N,M}$. The shared input and output moduli can be different.

Functionality: $c = \mathcal{F}_{\texttt{EEQ}^{*}_{N,M}}(a, b)$ with $a, b \in \mathbb{Z}_N, c \in \mathbb{Z}_M$

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

- **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 the shares of both parties, compute $[\![d]\!]_N$, where $[\![d]\!]_N = [\![a]\!]_N - [\![b]\!]_N$. After computing $[\![d]\!]_N$, set c as the Hamming distance between $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$. Then, return to Alice and Bob their respective shares of $[\![c]\!]_M$. Note that $c = 0$ if $a = b$ and $1 \leq c \leq \lceil \log_2(N) \rceil$, otherwise.

We implement a protocol for $\mathcal{F}_{\texttt{EEQ}^{*}_{N,M}}$ using Shared OTs and elementary local operation over shared elements. At the high-level idea, let $[\![d]\!]_N = [\![a]\!]_N - [\![b]\!]_N$. Since $d = 0$ iff $[\![d]\!]^A_N = [\![d]\!]^B_N$, we can just privately compute h, the Hamming distance between the binary representations of $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$ to obtain the desired output as specified by the functionality, given that $d = 0$ iff $a = b$ and that $h = 0$ iff $[\![d]\!]^A_N = [\![d]\!]^B_N$. The value of h can be obtained by computing the weight of the bitwise XOR of $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$, which implies that the underlying modulus must be changed from 2 to $M > \log_2 N$ to perform this addition. We rely on SOT for the modulus conversion.

Protocol $\Pi_{\texttt{EEQ}^{*}_{N,M}}$

Set $\ell = \lceil \log_2 N \rceil$.

1. Party $X \in \{A, B\}$ locally computes $[\![d]\!]^X_N = [\![a]\!]^X_N - [\![b]\!]^X_N \pmod N$
2. Alice locally computes the binary expansion $\vec{u} \in \mathbb{Z}_2^\ell$ of $[\![d]\!]^A_N$.
3. Bob locally computes the binary expansion $\vec{v} \in \mathbb{Z}_2^\ell$ of $[\![d]\!]^B_N$.
4. Alice sets $[\![\vec{x}_i]\!]^A_2 = \vec{u}_i$, for $0 \leq i \leq \ell - 1$.
5. Bob sets $[\![\vec{x}_i]\!]^B_2 = \vec{v}_i$, for $0 \leq i \leq \ell - 1$. ($\vec{x}_i = \vec{u}_i \oplus \vec{v}_i$)
6. Execute $[\![\vec{x}_i]\!]_M \leftarrow \mathcal{F}_{\texttt{SOT}^2_M}((0, 1), [\![\vec{x}_i]\!]_2)$, for $0 \leq i \leq \ell - 1$. (This converts $[\![\vec{x}_i]\!]_2$ to $[\![\vec{x}_i]\!]_M$)
7. Party $X \in \{A, B\}$ locally computes $[\![c]\!]^X_M = \sum_{i=0}^{\ell-1} [\![x_i]\!]^X_M \pmod M$. (Here, c is the Hamming distance between $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$)

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

Proof. **Correctness:** From the definition of $\Pi_{\texttt{EEQ}^*_{N,M}}$, we know that $\vec{u}$ and $\vec{v}$ are the binary expansion of $[\![d]\!]^A_N$ and $[\![d]\!]^A_N$, 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 7, is the Hamming distance between $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$. Thus, $\Pi_{\texttt{EEQ}^*_{N,M}}$ 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 $\mathcal{F}_{\texttt{SOT}^N_M}$, since its output distributions are always known. Note that this holds true regardless of the distribution of the input secret shares provided to $\mathcal{F}_{\texttt{SOT}^N_M}$, which in this case are represented by the values $\vec{u}_i$ and $\vec{v}_i$, as the SOT functionality does not impose any assumptions on the distribution of these shares.

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. □

$\Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}$ with Pre-processing Phase. We observe that the only interaction in $\Pi_{\texttt{EEQ}^*_{N,M}}$ occurs in Step 6. Therefore, we can replace this interaction with another one that can be performed in advance during a preprocessing phase by implementing a randomized $\texttt{SOT}^2_M$. However, in doing so, we must take additional care to use the random values computed during the preprocessing phase to convert $[\![\vec{x}_i]\!]_2$ to $[\![\vec{x}_i]\!]_M$. This conversion is carried out in steps 8 through 10 of the following protocol.

Protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}$

1. Party $X \in \{A, B\}$ locally samples $[\![\vec{r}_i]\!]^X_2 \in_R \mathbb{Z}_2$, for $0 \leq i \leq \ell - 1$.
2. Execute $[\![\vec{r}_i]\!]_M \leftarrow \mathcal{F}_{\texttt{SOT}^2_M}((0,1), [\![\vec{r}_i]\!]_2)$, 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 $[\![d]\!]^X_N = [\![a]\!]^X_N - [\![b]\!]^X_N \pmod N$
4. Alice locally computes the binary expansion $\vec{u} \in \mathbb{Z}^\ell_2$ of $[\![d]\!]^A_N$.
5. Bob locally computes the binary expansion $\vec{v} \in \mathbb{Z}^\ell_2$ of $[\![d]\!]^B_N$.
6. Alice sets $[\![\vec{x}_i]\!]^A_2 = \vec{u}_i$, for $0 \leq i \leq \ell - 1$.
7. Bob sets $[\![\vec{x}_i]\!]^B_2 = \vec{v}_i$, for $0 \leq i \leq \ell - 1$.
8. Party $X \in \{A, B\}$ locally computes and reveals $[\![\vec{g}_i]\!]^X_2 = [\![\vec{x}_i]\!]^X_2 \oplus [\![\vec{r}_i]\!]^X_2$, for $0 \leq i \leq \ell - 1$. (Reveals $\vec{g}_i = \vec{x}_i \oplus \vec{r}_i$)
9. Alice locally computes $[\![\vec{x}_i]\!]^A_M = [\![\vec{r}_i]\!]^A_M - 2 \cdot \vec{g}_i \cdot [\![\vec{r}_i]\!]^A_M \pmod M$, for $0 \leq i \leq \ell - 1$. ($[\![\vec{x}_i]\!]^A_M = [\![\vec{u}_i \oplus \vec{v}_i]\!]^A_M$)
10. Bob locally computes $[\![\vec{x}_i]\!]^B_M = \vec{g}_i + [\![\vec{r}_i]\!]^B_M - 2 \cdot \vec{g}_i \cdot [\![\vec{r}_i]\!]^B_M \pmod M$, for $0 \leq i \leq \ell - 1$. ($[\![\vec{x}_i]\!]^B_M = [\![\vec{u}_i \oplus \vec{v}_i]\!]^B_M$)
11. Party $X \in \{A, B\}$ locally computes $[\![c]\!]^X_M = \sum_{i=0}^{\ell-1} [\![\vec{x}_i]\!]^X_M \pmod M$. ($c$ is the hamming distance between $[\![d]\!]^A_N$ and $[\![d]\!]^B_N$)

Theorem 3. *Protocol $\Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}$ is correct and securely implements the functionality $\mathcal{F}_{\texttt{EEQ}^*_{N,M}}$*

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 8, 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 9 through 10, 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 11, we can see that c is the Hamming distance between $[\![d]\!]_N^A$ and $[\![d]\!]_N^B$. Thus, protocol $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*_{N,M}}$ 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 $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*_{N,M}}$, 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 $\vec{g}$, the vector revealed in the 8th step. □

4.2 Element-wise Equality

Below is the ideal functionality $\mathcal{F}_{\mathtt{EEQ}_N}$ of Element-wise Equality. We observe that the output modulus M always equals 2 and is therefore omitted from the notation.

Functionality $\mathcal{F}_{\mathtt{EEQ}_N}$

The functionality $\mathcal{F}_{\mathtt{EEQ}_N}$ 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 $[\![c]\!]_2$.

It is easy to see that the function $\mathtt{EEQ}_N$ can be derived from $\mathtt{EEQ}^*_{N,M}$ by remapping the possible outputs as follows: 0 maps to 1, while any value greater than 0 maps to 0.

This remapping can be implemented by employing a randomized 1-out-of-N OT with the choice vector $\vec{m} = (1, 0, \ldots, 0)$ and the choice value c. In our notation, this corresponds to a call to $\mathtt{SOT}_2^M$ with inputs $\mathsf{One}_M(0, 1)$ and $[\![h]\!]$, where $M = \ell + 1$ and $[\![h]\!]$ represents substractive shares outputted by $\mathcal{F}_{\mathtt{EEQ}^*_{N,M}}(a, b)$.

We use this strategy to construct two protocols: $\Pi_{\mathtt{EEQ}_N}$ and $\Pi^{\mathcal{P}}_{\mathtt{EEQ}_N}$. The only difference between these two constructions is that while $\Pi_{\mathtt{EEQ}_N}$ uses $\Pi_{\mathtt{EEQ}^\star}$ as a subprotocol, $\Pi^{\mathcal{P}}_{\mathtt{EEQ}_N}$ uses $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^\star}$ as a subprotocol. The full description of $\Pi_{\mathtt{EEQ}_N}$ together with its correctness and security proofs can be found below. The description of $\Pi^{\mathcal{P}}_{\mathtt{EEQ}_N}$ with its respective proofs can be found in Appendix A.

Protocol $\Pi_{\mathtt{EEQ}_N}$

Set $\ell = \lceil \log_2 N \rceil$.

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

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

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

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

4.3 Bitwise Integer Comparison

The bitwise integer comparison of two secret shared elements (a and b) is defined as follows:

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

Note that a and b b can be shared either as elements modulo an integer $\mathbb{N}$ or by sharing the bits of their binary representation modulo 2. In this context, we consider the latter approach. This leads to the following definition for the private bitwise comparison functionality $\mathcal{F}_{\mathtt{BLT}_\ell}$.

Functionality $\mathcal{F}_{\mathtt{BLT}_\ell}$

$\mathcal{F}_{\mathtt{BLT}_\ell}$ 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 $[\![\vec{a}]\!]_2$ and $[\![\vec{b}]\!]_2$, 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 $[\![c]\!]_2$ to Alice and Bob.

Using Shared OTs and the previously described protocols $\Pi_{\mathtt{EEQ}^*_{N,M}}$ and $\Pi^{\mathcal{P}}_{\mathtt{EEQ}^*_{N,M}}$, we present two protocols that implement $\mathcal{F}_{\mathtt{BLT}_\ell}$, denoted by $\Pi_{\mathtt{BLT}_\ell}$ and $\Pi^{\mathcal{P}}_{\mathtt{BLT}_\ell}$. First, we provide an intuitive explanation of the idea behind these protocols, which is to use Shared OTs to compute the following boolean expression privately:

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

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

which we arrived at by interpreting the "algorithm being privately evaluated" through the private comparison protocol proposed in [DFK+06], and further adapted to leverage our SOT primitive. To intuitively understand why the previously described Boolean expression computes the desired comparison, we start by analyzing $\vec{s}$ and understanding its behavior when $\vec{a} = \vec{b}$ and when $\vec{a} \neq \vec{b}$.

When $\vec{a} = \vec{b}$, the behavior of $\vec{s}$ is straightforward to predict because in this case, $\vec{a}_i \oplus \vec{b}_i = 0$ for $i \in \{0, 1, \ldots, \ell - 1\}$. This implies that if $\vec{a} = \vec{b}$, then $\vec{s}_i = 0$ for $i \in \{0, 1, \ldots, \ell - 1\}$. Now, let's consider the scenario 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 $\vec{b}_i$ differ). By using k, we can understand the behavior of $\vec{s}$ when $\vec{a} \neq \vec{b}$ by dividing the vector into three sections: the section between 0 and $k - 1$, the section between $k + 1$ and $\ell - 1$, and the section that only contains $\vec{s}_k$. Now, we can analyze each section separately.

Let's begin with the section between $k+1$ and $\ell-1$, which represents the most significant section. Since k is the position of the most significant pair of bits where $\vec{a}_i \oplus \vec{b}_i = 1$, it follows that $\vec{a}_j \oplus \vec{b}_j = 0$ for $j \in \{k+1, \ldots, \ell-1\}$. Given this observation and the definition of $\vec{s}$, we find that $\vec{s}_i = 0$ for $i \in \{k+1, \ldots, \ell-1\}$, indicating that all positions of $\vec{s}$ between $k + 1$ and $\ell - 1$ contain only 0's. Now, let's turn our attention to the section containing only $\vec{s}_k$. Since $\vec{a}_k \oplus \vec{b}_k = 1$, according to the definition of $\vec{s}$, we have $\vec{s}_k = 1$.

Next, let's examine the behavior of the final section of the vector $\vec{s}$, spanning from 0 to $k - 1$. To better understand this section, let's rewrite the definition of $\vec{s}$ as follows, for $i \in \{0, 1, \ldots, k - 1\}$:

$$\vec{s}_i = \left(\bigvee_{i=0}^{k-1} \vec{a}_i \oplus \vec{b}_i \right) \vee \left(\vec{a}_k \oplus \vec{b}_k \right) \vee \left(\bigvee_{i=k+1}^{\ell-1} \vec{a}_i \oplus \vec{b}_i \right)$$

Since $\vec{a}_k \oplus \vec{b}_k = 1$, we can observe that $\vec{s}_i = 1$ for $i \in \{0, 1, \ldots, k - 1\}$. Based on the behavior of these three sections, we know that vector $\vec{s}$ appears as follows when $\vec{a} \neq \vec{b}$:

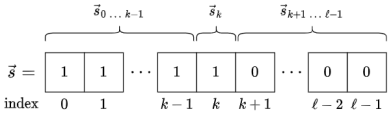

Figure 2: Three Sections of the Vector $\vec{s}$.

Now, still assuming that $\vec{a} \neq \vec{b}$, let's examine how this behavior of $\vec{s}$ ensures the correctness of the Boolean expression defining c. To do this, we first need to define four additional vectors: $\vec{s}', \vec{y}, \vec{y}', \vec{z} \in \mathbb{Z}_2^\ell$. These vectors are formally defined as follows:

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

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

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

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

Furthermore, the vector $\vec{z}$ contains the value of $\vec{b}_k$ in exactly one of its positions and 0's in all others. This can be understood by visualizing the vectors $\vec{s}, \vec{s}', \vec{y}, \vec{y}'$. In the following diagram, we illustrate these four vectors along with the vector $\vec{z}$ to make the reasoning completely clear.

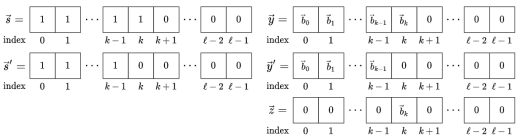

Figure 3: Relationship between Vectors $\vec{s}$,$\vec{s}'$,$\vec{y}$,$\vec{y}'$ and $\vec{z}$.

Based on this crucial fact about z, we can also conclude that $\bigoplus_{i=0}^{\ell-1} \vec{z}_i = \vec{b}_k$. This implies 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 $\vec{b}_k = \bigoplus_{i=0}^{\ell-1} \vec{z} = 1$ if and only if $b > a$. Thus, assuming $\vec{a} \neq \vec{b}$, $\bigoplus_{i=0}^{\ell-1} \vec{z} = 1$ if and only if $b > a$. 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:

$$\bigoplus_{i=0}^{\ell-1} \vec{z}_i = \bigoplus_{i=0}^{\ell-1} \vec{y}_i \oplus \vec{y}'_i = \bigoplus_{i=0}^{\ell-1} \vec{y}_i \oplus \bigoplus_{i=0}^{\ell-1} \vec{y}'_i \tag{1}$$

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

$$= (\bigoplus_{i=0}^{\ell-1} \vec{s}_i \wedge \vec{b}_i) \oplus (\bigoplus_{i=0}^{\ell-2} \vec{s}'_i \wedge \vec{b}_i) \tag{3}$$

$$= (\bigoplus_{i=0}^{\ell-1} \vec{s}_i \wedge \vec{b}_i) \oplus (\bigoplus_{i=0}^{\ell-2} \vec{s}_{i+1} \wedge \vec{b}_i) = c \tag{4}$$

Thus, if $\vec{a} \neq \vec{b}$, then $c = 1$ if and only if $b > a$. Moreover, since we have $\vec{s}_i = 0$ if $\vec{a} = \vec{b}$ for $i \in \{0, 1, \ldots, \ell - 1\}$, it is straightforward to analyze the Boolean expression defining c and realize that $c = 0$ if $\vec{a} = \vec{b}$.

The protocols presented in this section leverage SOTs to compute the Boolean expression defining c. Comments have been incorporated into the protocol descriptions to clarify the relationship between each step and the target Boolean expression. Two protocols are provided: one without preprocessing and one with preprocessing. The sole distinction lies in their instantiation of the functionality $\mathcal{F}_{\texttt{EEQ}^*}$, with one utilizing $\Pi_{\texttt{EEQ}^*_{N,M}}$ and the other employing $\Pi^{\mathcal{P}}_{\texttt{EEQ}^*_{N,M}}$.

The full description of $\Pi_{\texttt{BLT}_\ell}$ and its correctness and security proofs can be found below, while the description of $\Pi^{\mathcal{P}}_{\texttt{BLT}_\ell}$ and its respective proofs can be found in Appendix B.

Protocol $\Pi_{\texttt{BLT}_\ell}$

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 $0 \leq i \leq \ell - 1$. ($0 \leq \vec{h}_i \leq \ell'$; $\vec{\beta}_i = \vec{b}_i \cdot \frac{\lambda}{2}$; $\vec{t}_i = \vec{h}_i + \vec{\beta}_i > \frac{\lambda}{2} \iff \vec{h}_i > 0 \wedge \vec{\beta}_i = \frac{\lambda}{2} \iff \vec{b}_i \wedge \bigvee_{j=i}^{\ell-1} \vec{x}_j$)
6. Locally compute $[\![\vec{q_i}]\!]_\lambda = [\![\vec{h}_{i+1}]\!]_\lambda + [\![\vec{\beta_i}]\!]_\lambda$, for $0 \leq i \leq \ell - 2$. ($\vec{q}_i = \vec{h}_{i+1} + \vec{\beta}_i > \frac{\lambda}{2} \iff \vec{b}_i \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x}_j$)
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 $0 \leq i \leq \ell - 1$. ($\vec{d}_i = [\vec{t}_i > \frac{\lambda}{2}] = \vec{b}_i \wedge \bigvee_{j=i}^{\ell-1} \vec{x}_j$)
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 $0 \leq i \leq \ell - 2$. ($\vec{e}_i = [\vec{q}_i > \frac{\lambda}{2}] = \vec{b}_i \wedge \bigvee_{j=i+1}^{\ell-1} \vec{x}_j$)
9. Locally compute $[\![c]\!]_2 = \sum_{i=0}^{\ell-1} [\![\vec{d_i}]\!]_2 + \sum_{i=0}^{\ell-2} [\![\vec{e_i}]\!]_2$. ($c = \bigoplus_{i=0}^{\ell-1} \vec{d}_i \oplus \bigoplus_{i=0}^{\ell-2} \vec{e}_i$)

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 behavior of variables $\vec{x}, \vec{\beta}, \vec{d}, \vec{e}$ and c. It is straightforward 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 $\Pi_{\mathtt{EEQ}_N}$ can also be used to prove the security of $\Pi_{\mathtt{BLT}_\ell}$. □

4.4 Bit-Decomposition Protocol

In the context of private two-party computations involving a shared element $\beta \in \mathbb{Z}_N$, it might be beneficial to access the binary expansion of the value β. This process, commonly referred to as Bit-Decomposition, is crucial for various cryptographic tasks.

Here, we formally define a Bit-Decomposition functionality $\mathcal{F}_{\mathtt{BD}_\ell}$ where the input is an element β that is additively shared modulo 2^ℓ. The output consists of a sequence of shared bits, where $\ell \geq 2$, and the sequence's length is ℓ. Note that this functionality can also be utilized in a black-box manner to conduct the Bit-Decomposition of integer-secret-shared values. This is achievable by reducing both shares modulo 2^ℓ before presenting them as input to $\mathcal{F}_{\mathrm{BD}_\ell}$, where $\ell = \lceil(\log_2(m2^\kappa)\rceil$, m denotes the upper bound for the value being secret shared, and κ represents the statistical security parameter employed by the integer secret sharing scheme.

Functionality $\mathcal{F}_{\mathtt{BD}_\ell}$

$\mathcal{F}_{\mathtt{BD}_\ell}$ 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 $[\![\vec{\beta}]\!]_{2^\ell}$, 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 $[\![b_{\ell-1}]\!]_2, [\![b_{\ell-2}]\!]_2 \dots [\![b_0]\!]_2$.

We propose two protocols $\Pi_{\mathtt{BD}_\ell}$ and $\Pi'_{\mathtt{BD}_\ell}$ that efficiently implement $\mathcal{F}_{\mathtt{BD}_\ell}$. These protocols offer a tradeoff between the number of bits transferred and the number of communication rounds required for execution.

The underlying concept of both protocols is the same: they take the binary expansions u and v from $\mathbb{Z}_2^\ell$ of $[\![\beta]\!]^B_{2^\ell}$ and $-[\![\beta]\!]^A_{2^\ell}$, respectively. Then, they perform binary addition over $\vec{u}$ and $\vec{v}$, disregarding the last carry bit generated during the addition. This omission ensures that the result of the binary addition is equivalent to computing binary addition modulo 2^ℓ. Since the output of the binary addition modulo 2^ℓ is a sequence of shared bits, and $\beta = [\![\beta]\!]^B_{2^\ell} - [\![\beta]\!]^A_{2^\ell} \pmod{2^\ell}$, the output of the addition serves as the desired output for the Bit-Decomposition protocol.

To compute the binary addition modulo 2^ℓ, we start by calculating the carry bit vector $\vec{c}$, which stores the carry bits generated during the binary addition of $\vec{u}$ and $\vec{v}$. Then, we compute $\vec{b_i} = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i$ for $0 \leq i \leq \ell - 1$, forming the vector that represents the binary expansion of β. By initially defining the expressions for the least significant bits of $\vec{c}$, we can derive the expression for $\vec{c}$ as a whole. The expressions defining the four least significant bits of $\vec{c}$ are as follows:

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

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

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

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

As mentioned earlier, we can analyze these expressions and derive the following set of Boolean equations that define the carry bit vector $\vec{c}$. A formal proof showing the correctness of these equations can be found in appendix D.

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

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

$$\vec{g}_i = \vec{u}_i \wedge \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$

$$\vec{x}_i = \vec{u}_i \oplus \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$

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

The description of $\Pi_{\mathtt{BD}_\ell}$ can be found below together with its correctness and security proofs, while the description of $\Pi'_{\mathtt{BD}_\ell}$ and its respective proofs can be found in Appendix D.

Protocol $\Pi_{\mathtt{BD}_\ell}$

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

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

Theorem 6. *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 $(-[\![\beta]\!]^A_{2^\ell} \pmod{2^\ell})$ and $[\![\beta]\!]^B_{2^\ell}$, respectively, and $\vec{c}' \in \mathbb{Z}_2^{\ell+1}$ be the carry bit vector generated when computing $\alpha = [\![\beta]\!]^B_{2^\ell} + (-[\![\beta]\!]^A_{2^\ell} \pmod{2^\ell})$. 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:

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

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

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

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

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

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

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

$$\vec{h}_{i,j} = \vec{g}_j + \sum_{k=j+1}^{i-1} \vec{x}_k, \text{ for } 0 \leq j < i \leq \ell - 1$$

$$\vec{x}_i = \vec{a}_i \oplus \vec{d}_i, \text{ for } 0 \leq i \leq \ell - 1$$

$$\vec{g}_i = \vec{a}_i \wedge \vec{d}_i, \text{ for } 0 \leq i \leq \ell - 1$$

Looking at these equations we can see that $\vec{x}_i, \vec{g}_i \in \{0, 1\}$ for $0 \leq i \leq \ell - 1$, which implies that $0 \leq \vec{h}_{i,j} \leq i - j$ and $\vec{h}_{i,j} = i - j$ iff $\vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k$, for $0 \leq j < i \leq \ell - 1$. 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 $0 \leq j < i \leq \ell - 1$. Thus, looking at the equation that dictates the value of $\vec{c}$, we can conclude that $\vec{c}_i = \vec{c'}_i$ for $0 \leq i \leq \ell - 1$. Therefore, we have that $\Pi_{\texttt{BD}_\ell}$ is correct.

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

5 Results and Comparison

To ensure a fair comparison with existing works, we limit our analysis to those with the following characteristics:

- We consider only two-party protocols.
- We exclusively examine protocols that offer unconditional security.
- The protocols must exhibit perfect correctness, guaranteeing a probability of 1 for returning the correct output.
- The protocols must have constant round complexity.

Due to these limitations, works such as [DSZ15, Cou18, RRK+20, EGK+20], are not included in our comparisons. Specifically, the protocols proposed in these papers do not feature a constant number of rounds. For instance, the secure comparison protocol in [EGK+20] exhibits logarithmic round complexity with respect to the input size. Note that protocols with a constant number of rounds that are information-theoretically secure are relatively rare in the literature.

We also restrict our comparison to the three most efficient protocols that meet our previously outlined criteria for each protocol type. As a result, works like [DFK+06] are excluded from our comparison, as more efficient alternatives [NO07, LT13, Yu11] to the protocols proposed in that paper are already covered.

Table 1: Protocol Efficiency Comparison

$\mathcal{F}_{\texttt{EEQ}}$ **Protocol**	[LT13]	[Yu11]	[NO07]	$\Pi_{\texttt{EEQ}}$	$\Pi^{\mathcal{P}}_{\texttt{EEQ}}$
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
$\mathcal{F}_{\texttt{BLT}}$ **Protocol**	[Rei09]	[NO07]	[Yu11]	$\Pi_{\texttt{BLT}}$	$\Pi^{\mathcal{P}}_{\texttt{BLT}}$
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
$\mathcal{F}_{\texttt{BD}}$ **Protocol**	[NO07]	[Tof09]	[RT10]	$\Pi_{\texttt{BD}}$	$\Pi_{\texttt{BD}'}$
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.

To compare the efficiency of the protocols that match the previously described criteria with our constructions, we analyze and compare the number of communication rounds required to execute the protocol, their computational complexity, and communication complexity (complexity class of the number of bits transferred during the protocol's execution). All previously published works considered in our comparisons measure computational complexity and the number of bits transferred by the number of times a multiplication protocol is invoked. However, since we use our SOT functionality as a primitive instead of a private multiplication protocol as previous works do, we cannot use the same comparison methodology.

To deal with this difference in primitives, we measure the communication complexity in the number of bits transmitted by the two parties, and we measure the computational complexity of the protocols in the same way the computational complexity of algorithms is measured. To do this we start by assuming that adding and multiplying to elements modulo N have computational complexity $O(\ell)$ and $O(\ell^2)$, respectively, where $\ell = \lceil \log_2(N) \rceil$. Next, we analyze the complexities of the private multiplication protocol and our SOT construction.

For private multiplication, we assume the two parties A and B already hold a Beaver triple generated by a trusted initializer, and they use this beaver triple in a straightforward manner to execute private multiplication and receive an additively secret-shared output. Assuming the two parties are performing a private multiplication modulo M, they need to transmit a constant amount of elements modulo M and execute a constant amount of local multiplications modulo M. This gets us computational complexity of $O(\ell^2)$ and communication complexity of $O(\ell)$, where $\ell = \lceil \log_2(M) \rceil$.

Running $\Pi_{\texttt{SOT}^N_M}$ requires sampling a single element modulo M, adding N elements modulo M, performing a cyclic shift over a vector length N and running a single instance of

$\mathcal{F}_{\mathtt{OT}_M^N}$. Assuming the OT protocol proposed in [Riv] is used to implement $\mathcal{F}_{\mathtt{OT}_M^N}$, sampling is done in constant time and cshift_N has computational complexity of $O(N)$, we can conclude that $\Pi_{\mathtt{SOT}_M^N}$ has computational complexity of $O(N \cdot \log(M) + \log(N))$ and communication complexity of $O(N \cdot \log(M))$.

Using the complexity analysis of the two primitives and the assumptions made about the complexity of addition and multiplication mod N, we can then inspect the existing protocols and ours, and arrive in the complexity classes presented in Table 1. We would like to note that when inspecting previously protocols that are secure against malicious adversaries, we considered straightforward changes that could improve their performance when considering only semi-honest adversaries. However, we did not find any optimization that improved performance asymptotically.

We believe our asymptotic improvements do not come from the fact we are considering only semi-honest adversaries but actually from a combination of our setting having exactly two parties and the ways we use our new SOT functionality to implement the newly proposed protocols, especially when it comes to using SOTs to convert secret shared values between different modulo.

6 Conclusion and Future Work

In this work, we studied a natural extension of the OT functionality, which we termed Shared OT. We utilized this new primitive to develop protocols for private equality ($\mathcal{F}_{\mathtt{EEQ}}$), private comparison ($\mathcal{F}_{\mathtt{BLT}}$), and bit-decomposition ($\mathcal{F}_{\mathtt{BD}}$) functionalities. All these protocols satisfy the following properties: unconditional security in the two-party semi-honest setting, perfect correctness, and constant round complexity. Our constructions demonstrate superior performance compared to previous protocols that share these same properties. Three interesting questions remained unexplored:

- Can the protocols presented in this work be modified to be secure in the malicious adversary model while maintaining their efficiency advantages?
- Can the ideas proposed in this paper be adapted to the computational security setting in a way that leads to improvements compared to other works in that setting?
- Can the proposed protocols be adapted to the multi-party setting such that they provide performance improvements compared to other protocols in that setting?

Malicious Protocols. To achieve security in the malicious setting, a natural starting point is to replace our additive secret shares with committed additive secret shares. Note that unconditionally secure linear homomorphic commitment schemes have been proposed in the past, such as in [NMO+03, Riv]. By using committed additive secret shares, the two parties can prove to each other that linear operations over the committed shares were performed correctly. This verification covers virtually all operations in our protocols, except for executing the $\mathcal{F}_{\mathtt{SOT}}$ primitive.

While our $\mathcal{F}_{\mathtt{SOT}}$ is secure in the malicious setting, it does not support committed inputs or output committed secret shares. Therefore, a new committed variant of the primitive would need to be introduced. A promising solution is to study the already proposed Committed Oblivious Transfer [CvT95]. However, our protocols work with values of different moduli, a setting not addressed by previously proposed unconditional commitment schemes, adding a layer of complexity.

Aside from this main challenge, the use of commitment schemes adds performance overhead. We would need to study how this overhead impacts the efficiency of our protocols compared to other previously malicious-secure solutions. Due to these complexities, we

decided to focus only on the semi-honest setting in this work and leave the task of securing these protocols against malicious adversaries for future research.

Protocols in the Computational Security Setting. Given the extensive body of work in this setting that has been published, we believe that investigating this question might lead to fruitful results given the promising outcomes presented in Section 5. In addition, our protocols rely on such an efficient primitive as OT, support this belief. Many works, such as [IKNP03, PRTY19, Roy22, BCG+22, RRT23] have proposed methods to pre-compute the OT primitive in a batched manner with very good runtimes and rate-1 communication.

Multi-party Setting. The equality, comparison, and bit-decomposition protocols introduced in this paper all rely on the newly proposed SharedOT primitive, which currently supports only two parties: a sender and a receiver. An interesting direction for future research would be to explore how this functionality could be extended to accommodate multiple parties, while still preserving the necessary behavior to implement the protocols presented in this work.

Acknowledgments.

Lucas Piske and Ni Trieu were partially supported by NSF award #2115075, and ARPA-H SP4701-23-C-0074. Portions of this work were conducted while Nascimento was affiliated with University of Washington, Tacoma.

References

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. Cryptology ePrint Archive, Report 2022/1014, 2022. https://eprint.iacr.org/2022/1014.

[Bea97] Donald Beaver. Commodity-based cryptography (extended abstract). In *29th ACM STOC*, pages 446–455. ACM Press, May 1997. doi:10.1145/258533.258637.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017. doi:10.1145/3133956.3133982.

[BK04] Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 515–529, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 503–513, 1990.

[Cou18] Geoffroy Couteau. New protocols for secure equality test and comparison. In *Applied Cryptography and Network Security: 16th International Conference,*

ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings, page 303–320, Berlin, Heidelberg, 2018. Springer-Verlag. doi:10.1007/978-3-319-93387-0_16.

[CvT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 110–123. Springer, Heidelberg, August 1995. doi:10.1007/3-540-44750-4_9.

[DDG+23] Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 360–392. Springer, Heidelberg, August 2023. doi:10.1007/978-3-031-38557-5_12.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 285–304, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15985-5_3.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 247–264, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[EGK+20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56880-1_29.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985. doi:10.1145/3812.3818.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In *Annual International Cryptology Conference*, pages 85–114. Springer, 2019.

[HKN24] David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. EUROCRYPT, 2024. https://eprint.iacr.org/2024/369. URL: https://eprint.iacr.org/2024/369.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_9.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_4.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016. doi:10.1145/2976749.2978381.

[LT13] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear 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.

[MMT+24] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin Case. Delegated private matching for compute. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2024(2):49–72, 2024. Also available at Cryptology ePrint Archive, Paper 2023/012, https://eprint.iacr.org/2023/012. URL: https://petsymposium.org/popets/2024/popets-2024-0040.php, doi:10.56553/popets-2024-0040.

[NMO+03] Anderson C. A. Nascimento, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, and Hideki Imai. Unconditionally secure homomorphic pre-distributed bit commitment and secure two-party computations. In Colin Boyd and Wenbo Mao, editors, *ISC 2003*, volume 2851 of *LNCS*, pages 151–164. Springer, Heidelberg, October 2003.

[NO07] 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.

[NWKT24] Truong Son Nguyen, Lun Wang, Evgenios M. Kornaropoulos, and Ni Trieu. Aitia: Efficient secure computation of bivariate causal discovery. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2024.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_13.

[Rab81] Michael Rabin. How to exchange secrets by oblivious transfer, 1981.

[Rei09] Tord Ingolf Reistad. Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values. In *International Conference on Security and Cryptography*, volume 1, pages 325–330. SCITEPRESS, 2009.

[Riv] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. unpublished.

[Roy22] Lawrence Roy. SoftSpokenOT: Communication–computation tradeoffs in OT extension. Cryptology ePrint Archive, Report 2022/192, 2022. https://eprint.iacr.org/2022/192.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 94–124, Cham, 2021. Springer International Publishing.

[RRK+20] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 325–342, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372297.3417274.

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632. Springer, Heidelberg, August 2023. doi:10.1007/978-3-031-38551-3_19.

[RT10] 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.

[SCJ13] Bharath K. K. Samanthula, Hu Chun, and Wei Jiang. An efficient and probabilistic secure bit-decomposition. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, page 541–546, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2484313.2484386.

[Tof09] 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.

[vBP24] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. In *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part V*, page 340–369, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-58740-5_12.

[Veu12] Thijs Veugen. Improving the dgk comparison protocol. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 49–54, 2012. doi:10.1109/WIFS.2012.6412624.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982. doi:10.1109/SFCS.1982.38.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

[YNKM24] Albert Yu, Hai H. Nguyen, Aniket Kate, and Hemanta K. Maji. Unconditional security using (random) anonymous bulletin board. Cryptology ePrint Archive, Paper 2024/101, 2024.

[Yu11] Ching-Hua Yu. Sign modules in secure arithmetic circuits. Cryptology ePrint Archive, Report 2011/539, 2011. https://ia.cr/2011/539.

[YY12] Ching-Hua Yu and Bo-Yin Yang. Probabilistically correct secure arithmetic computation for modular conversion, zero test, comparison, mod and exponentiation. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 426–444, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

A Element Equality* with Pre-processing Phase

Protocol $\Pi^{\mathcal{P}}_{\mathtt{EEQ}_N}$

Let $\ell = \lceil \log_2(N) \rceil$ be the minimum amount of bits necessary to represent an element of $\mathbb{Z}_N$.

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

Theorem 7. *Protocol $\Pi^{\mathcal{P}}_{\mathtt{EEQ}_N}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{EEQ}_N}$ against semi-honest adversaries in the commodity-based model.*

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

B Bitwise Comparison with Pre-processing Phase

Protocol $\Pi^{\mathcal{P}}_{\mathtt{BLT}_\ell}$

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

1. Execute $[\![\vec{x}_i]\!]_{\ell+1} \leftarrow \mathcal{F}_{\mathtt{SOT}^2_{\ell+1}}((0,1), [\![\vec{a}_i + \vec{b}_i]\!]_2)$, for $0 \le i \le \ell - 1$. ($\vec{x}_i = \vec{a}_i \oplus \vec{b}_i$)
2. Execute $[\![\vec{\beta}_i]\!]_\lambda \leftarrow \mathcal{F}_{\mathtt{SOT}^2_\lambda}((0, \frac{\lambda}{2}), [\![\vec{b}_i]\!]_2)$, 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. $[\![\vec{h}_i]\!]_\lambda \leftarrow \Pi^{\mathcal{P}}_{\mathtt{EEQ}^*_{\ell+1,\lambda}}([\![\vec{s}_i]\!]_{\ell+1}, [\![0]\!]_{\ell+1})$, 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 $[\![c]\!]_2 = \sum_{i=0}^{\ell-1} [\![\vec{d}_i]\!]_2 + \sum_{i=0}^{\ell-2} [\![\vec{e}_i]\!]_2$. $(c = \bigoplus_{i=0}^{\ell-1} \vec{d}_i \oplus \bigoplus_{i=0}^{\ell-2} \vec{e}_i)$

Theorem 8. *Protocol $\Pi^{\mathcal{P}}_{\mathtt{BLT}_\ell}$ is correct and securely implements the functionality $\mathcal{F}_{\mathtt{BLT}_\ell}$ against semi-honest adversaries in the commodity-based model.*

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

C Bit-Decomposition with Pre-processing Phase

Protocol $\Pi'_{\mathtt{BD}_\ell}$

Let $\vec{v} \in \mathbb{Z}_2^\ell$ and $\vec{u} \in \mathbb{Z}_2^\ell$ be the binary expansions of $(-[\![\beta]\!]^A_{2^\ell} \pmod{2^\ell})$ and $[\![\beta]\!]^B_{2^\ell}$, 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 $0 \le i \le \ell - 1$. $(\vec{g}_i = \vec{u}_i \wedge \vec{v}_i)$
2. Execute $[\![\vec{x}_i]\!]_\ell \leftarrow \mathcal{F}_{\mathtt{SOT}_\ell^2}((0,1), [\![\vec{u}_i + \vec{v}_i]\!]_2)$, for $0 \le i \le \ell - 1$. $(\vec{x}_i = \vec{u}_i \oplus \vec{v}_i)$
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 $[\![\vec{t}_{i,j}]\!]_2 \leftarrow \mathcal{F}_{\mathtt{EEQ}_\ell}([\![\vec{h}_{i,j}]\!]_\ell, [\![i-j]\!]_\ell)$, 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 $0 \le i \le \ell - 1$. $(\vec{b}_i = \vec{u}_i \oplus \vec{v}_i \oplus \vec{c}_i)$

Theorem 9. *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 $\mathcal{F}_{\texttt{EEQ}}$ we can see that the values of $\vec{t}_i, j$, in the description for $\Pi'_{\texttt{BD}_\ell}$, are defined by the following equation:

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

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

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

□

D Correctness Proof for Carry-bit Expression

Theorem 10. *Let $\ell \in \mathbb{N}^{>1}, a \in \mathbb{Z}_{2^\ell}, b \in \mathbb{Z}_{2^\ell}$, and $\vec{u}, \vec{v} \in \mathbb{Z}^{\ell+1}$ be the binary expansion of a and b, respectively. The vector $\vec{w} = \vec{u} + \vec{v} + \vec{c}$ is the binary expansion of $a + b$, where c is defined by the following expressions:*

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

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

$$\vec{g}_i = \vec{u}_i \wedge \vec{v}_i, \text{ for } 0 \leq i \leq \ell$$

$$\vec{x}_i = \vec{u}_i \oplus \vec{v}_i, \text{ for } 0 \leq i \leq \ell$$

Proof. We start this proof by rewriting the expression for $\vec{c}$ and $\vec{t}$ in a recursive form:

$$\begin{aligned} \vec{t}_{i,j} &= \vec{g}_j \wedge \bigwedge_{k=j+1}^{i-1} \vec{x}_k \\ &= \vec{g}_j \wedge \vec{x}_{i-1} \wedge \bigwedge_{k=j+1}^{i-2} \vec{x}_k \\ &= \vec{x}_{i-1} \wedge \vec{t}_{i-1,j} \end{aligned}, \text{ for } 0 \leq j < i - 1 \leq \ell$$

$$\begin{aligned} \vec{c}_i &= \bigoplus_{j=0}^{i-1} \vec{t}_{i,j} \\ &= \vec{t}_{i,i-1} \oplus \bigoplus_{j=0}^{i-2} \vec{t}_{i,j} \\ &= \vec{t}_{i,i-1} \oplus \bigoplus_{j=0}^{i-2} \vec{x}_{i-1} \wedge \vec{t}_{i-1,j} \\ &= \vec{t}_{i,i-1} \oplus \vec{x}_{i-1} \wedge \bigoplus_{j=0}^{i-2} \vec{t}_{i-1,j} \\ &= \vec{t}_{i,i-1} \oplus \vec{x}_{i-1} \wedge \vec{c}_{i-1} \end{aligned}, \text{ for } 1 \leq i \leq \ell$$

Now, by further developing the recursive form of $\vec{c}$ we have:

$$\begin{aligned}\vec{c}_i &= \vec{t}_{i,i-1} \oplus \vec{x}_{i-1} \wedge \vec{c}_{i-1} \\ &= (\vec{u}_{i-1} \wedge \vec{v}_{i-1}) \oplus \vec{x}_{i-1} \wedge \vec{c}_{i-1} \qquad \text{, for } 1 \leq i \leq \ell \\ &= (\vec{u}_{i-1} \wedge \vec{v}_{i-1}) \oplus (\vec{u}_{i-1} \wedge \vec{c}_{i-1}) \oplus (\vec{v}_{i-1} \wedge \vec{c}_{i-1})\end{aligned}$$

Based on this, we can see by inspection that

$$(\vec{u}_{i-1} + \vec{v}_{i-1} + \vec{c}_{i-1}) > 1 \iff \vec{c}_i = 1, \text{ for } 1 \leq i \leq \ell$$

Thus, by using the binary addition algorithm, we have that $\vec{w} = \vec{u} + \vec{v} + \vec{c}$ is the binary expansion of $a + b$ when defining $\vec{c}$ as defined by this theorem.

□

Theorem 11. *Let $\ell \in \mathbb{N}^{>1}, a \in \mathbb{Z}_{2^\ell}, b \in \mathbb{Z}_{2^\ell}$, and $\vec{u}, \vec{v} \in \mathbb{Z}_2^\ell$ be the binary expansion of a and b, respectively. The vector $\vec{w} = \vec{u} + \vec{v} + \vec{c}$ is the binary expansion of $a + b \pmod{2^\ell}$, where c is defined by the following expressions:*

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

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

$$\vec{g}_i = \vec{u}_i \wedge \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$

$$\vec{x}_i = \vec{u}_i \oplus \vec{v}_i, \text{ for } 0 \leq i \leq \ell - 1$$

Proof. Let $\vec{f}, \vec{h} \in \mathbb{Z}_2^{\ell+1}$ be the binary expansions of a and b, respectively, and $\vec{y} \in \mathbb{Z}_2^{\ell+1}$ be the binary expasion of $a + b$. We start by noting that from Theorem 10 we know that $y = \vec{f} + \vec{h} + \vec{c}$ is the binary expansion of $a + b$, where c is defined in the following way:

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

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

$$\vec{g}_i = \vec{f}_i \wedge \vec{h}_i, \text{ for } 0 \leq i \leq \ell$$

$$\vec{x}_i = \vec{f}_i \oplus \vec{h}_i, \text{ for } 0 \leq i \leq \ell$$

Next, we also note that the following is true:

$$a + b = \sum_{i=0}^{\ell} \vec{y}_i \cdot 2^i = \vec{y}_\ell \cdot 2^\ell + \sum_{i=0}^{\ell-1} \vec{y}_i \cdot 2^i = \sum_{i=0}^{\ell-1} \vec{y}_i \cdot 2^i \pmod{2^\ell}$$

$$\sum_{i=0}^{\ell-1} \vec{y}_i \cdot 2^i \pmod{2^\ell} = \sum_{i=0}^{\ell-1} \vec{y}_i \cdot 2^i$$

.

Which means that $\vec{w} \in \mathbb{Z}_2^\ell$ is the binary expansion of $a + b \pmod{2^\ell}$ if and only if $\vec{w}_i = \vec{y}_i = \vec{f}_i + \vec{h}_i + \vec{c}_i$ for $0 \leq i \leq \ell - 1$. Because $\vec{u}$ and $\vec{f}$ are both binary expansions of a, and $\vec{v}$ and $\vec{h}$ are both binary expansions of b, we know that $\vec{u}_i = \vec{f}_i$ and $\vec{v}_i = \vec{h}_i$ for $0 \leq i \leq \ell - 1$. This, in turn, concludes the proof by implying that $\vec{w}_i = \vec{y}_i = \vec{u}_i + \vec{v}_i + \vec{c}_i$ for $0 \leq i \leq \ell - 1$, where $\vec{c}_i$ is defined as described by the theorem for $0 \leq i \leq \ell - 1$.

□