Efficient and Constant-Rounds Secure Comparison through Dynamic Groups and Asymmetric Computations

Ken Goss and Wei Jiang

Departmenet of Computer Science
Missouri University of Science and Technology
{ken.goss, wjiang}@mst.edu

Abstract. Within recent years, secure comparison protocols have been proposed using binary decomposition and properties of algebraic fields. These have been repeatedly optimized and increased in efficiency, but have seemingly reached a plateau. We propose a new approach to this problem that takes advantage of dynamic group sizes for intermediate calculations and asymmetric computations among participating parties. As a consequence, according to our analysis, communication and computation costs have been brought to a very low and efficient level. Particularly, the communication costs have been considerably reduced both in order as well as the dominating term's order of magnitude. In addition, our proposed protocol requires no secure multi-party multiplication invocations in contrast to those required by the existing protocols, leading to inefficient constructions of secure comparisons.

1 Introduction

A need for the ability to perform secure computation in a practical manner is increasing proportionally to the need of our society to securely analyze the data we are amassing, and that at increasingly fantastic rates.

At the heart of many analysis problems is the simple comparison operator which has been identified, repeatedly, as a computational bottleneck for further performance improvements [1]. Considering a variety of problems, the comparison operator will be heavily used in many cases. These problems range from the traditional problem of the Millionaires presented by Yao is considered [2], secure auctioning [3], as well as queries on a secure database [4]. Thus, finding an efficient approach solving this problem - in any given situation - will have a positive performance effect on the rest of the overall system's performance. The boost in performance from improving comparisons will naturally be proportional to the use of this operation in the overall system. This result is well known from the work of Amdahl and his eponymous Law [5].

In the last few decades, several strategies have been proposed to allow for secure computations which would mitigate or eliminate the concerns and threats to data confidentiality. They represent multiple general approaches to solving the

problem with various benefits and costs. Within the realm of secret sharing, two main classes of these approaches to providing a secure comparison exist. First, there are approaches based on binary decomposition. Another class of protocols exploits properties of finite field arithmetic. This is done through indirect comparisons with transformed values, which are logically combined to form the results of the desired comparison.

In general, the approaches currently known have been steadily improving. However, it seems that a plateau has been reached. These approaches represent constant rounds solutions, but the costs of their communications are still high with respect to the values on which they operate. This is at least partially due to the trade-off made in each of their work to achieve constant rounds. In order to achieve constant rounds, greater local computations and communications costs were sacrificed. This is related to the results for unbounded fan-in multiplication given in the work of Bar-Ilan and Beaver [6]. Given that these approaches require this result to be able to reduce their round complexity to a constant, and by self-admission [7], it seems unlikely to drop considerably further within the currently explored veins of inquiry. It is our desire to present a different approach.

1.1 Our Contributions

An important link in the extant work in the field is that all the current secret sharing based protocols are symmetric; that is, all parties perform identical computations on their locally held shares of the data. We propose an asymmetric secure comparison protocol, based on secret sharing, that allows for greater efficiency than previous protocols with respect to required execution rounds, local computational requirements and communications. In our proposed protocol, the following features lead to the efficiency gain:

- Without using Shamir's secret sharing scheme, no two parties need to execute the exact same instructions.
- This asymmetry allows for the control of individual pieces of knowledge to further minimize computation and communication complexities.
- It is important to note, however, that this introduction of asymmetry makes the resulting protocol more difficult to prove secure in general, as well as more difficult to secure against malicious adversaries specifically.
- Additionally, where possible, we take advantage of differing representations of values and field magnitudes for randomization and intermediate secure computations. This further reduces computation and communication costs.
- Most notably, by making use of the group $\mathbb{Z}_2$, we are able to compute `xor` locally, without secure multiplications.

We base our protocol on an additive secret sharing scheme and adopt the semi-honest or honest-but-curious adversary model. Our protocol is very efficient with respect to other contributions in the field, as can be seen by consulting Tables 1 and 2, representing online and total complexities respectively (where all the compared protocols are secure under the semi-honest model). Overall,

our proposed protocol requires 5 rounds, and the communication costs are on the order of $7\ell \log_2 \ell + 26\ell + 11 \log_2 \ell + 32$ bits for private inputs consisting of at most ℓ bits such that $\ell = \log_2 N$ for some N defining the group used for the underlying secret sharing scheme and sufficiently large to represent the values being compared. No secure multi-party multiplication invocations are required. Furthermore, our local computations are guaranteed to be less than those of the other protocols currently extant in the literature. This is due to our local operations being almost exclusively shifts, additions and multiplications, in a simple sense. We do not rely heavily on a large number of group elements being multiplicatively inverted, nor do we require solutions to a large set of systems of equations to be solved. Many of these operations are required by other works.

The communication complexities are described in terms of the required number of bits to be transmitted. This is a sharp contrast to the common convention of the currently optimized works in the area requiring a communication complexity represented by a count of secure multiplication invocations. The comparisons that follow are all calculated under the assumption that all the schemes are employing the secret sharing scheme chosen by their designers. A single secure multiplication under the Shamir secret sharing scheme requires, for m parties, assuming appropriate choices for the degree and threshold, $m(m-1)\ell$ bits to be communicated between the parties. For 3 parties and a protocol requiring $24\ell + 26 \log_2 \ell + 4$ multiplication invocations, this results in more than $144\ell^2 + 156\ell \log_2 \ell + 24\ell$ bits to be transmitted, a distinctly larger quantity than our proposed protocol. This is calculated directly from analyzing the number of bits required for a secure multiplication according to the method common for Shamir's secret sharing scheme [8]. The communication requirements and the number of required rounds would both grow even more, if these protocols were implemented under additive secret sharing because secure multiplication under additive secret sharing is more expensive (in rounds and bits transmitted) than that under Shamir's secret sharing. For all the following analysis, we assume 3 parties, the same as we propose for our own protocol. We then simply multiply the number of bits required for a single Shamir secure multiplication by the number of expected multiplication invocations to yield the figures we present. For our protocol consisting of an arbitrary number of parties, the complexity is dependent on both ℓ and m. We also generalize the analysis of the existing protocols when we discuss this extension separately in Section 5.2.

It is important to note, however, that the figures given for required communications according to the requirements for secure multiplication invocations are, by definition, not a tight bound. Other operations, most frequently invoked, the reveal operation, also require communication. The extant work neglects this operation by explaining that its cost is insignificant in the larger scheme of the protocols, and the resulting increase would grow only the coefficients but not the order of the communication complexity, which is negligible from an asymptotic point of view. Many reveal operations to rebuild shared values will be required with many of the intermediate steps of other protocols, particularly with prefix products due to the use of the procedure given by Bar Ilan and Beaver [6]. In

reality, the communication complexity of a reveal is the same as that of a secure multiplication under Shamir secret sharing. Due to these reasons, it is not considered negligible from our perspective, and this is why we draw the readers attention to this looseness in the cited analysis. In the analysis of our protocol, we make no use of such simplifications or insignificance assumptions. In contrast, the included complexity analysis for our protocol is exact.

Additionally we differ with respect to the round complexity analysis given by [9], in that we have a more strict definition of of a computational round, as discussed in Section 4.3. This stricter definition leads to a somewhat higher count of rounds than that given in the analysis published by the authors. To summarize the extant work in the field with respect to round and communication complexities, we offer Tables 1 and 2 based on the summary present in [7]. The following complexities for our table have been obtained directly from the table in the referenced work by the method mentioned previously. Similar to the work of Reistad and Toft [7], the "A" denotes arbitrary input in the group used for the secret sharing scheme, while "R" denotes a domain of input which is restricted to a proper subset of the group. Here, it is very clear that the communication costs of our protocol are significantly less than any other of which we are aware, by a considerable factor on the dominating term and a reduction of order overall.

Non-constant round secret sharing based secure comparison has not gained much attention from the research community since it can be straightforwardly implemented. As the domain of the values being compared increases, communication cost generally dominates local computation cost. Thus, recent research has been primarily focused on reducing the communication complexity. Homomorphic encryption based secure comparison protocols do exist [10, 11]. Their main advantage is that, at minimum, only two parties are needed to perform the necessary computations. However, they are computationally more expensive than secret sharing based approaches. In this paper, we limit our scope to constant-round secret sharing based secure comparison protocols.

Table 1. Online round and communication complexity for secure comparison protocols

Presented in	Type	Online Rounds	Bits transmitted online
[12]	A	37	$126\ell^2 \log_2 \ell + 336\ell^2$
[13]	A	8	$90\ell^2 + 30\ell$
[7]	A	4	$108\ell^2 + 48\ell$
[7]	R	2	$24\ell^2 + 6\ell$
[9]	R	7	$42\ell^2 + 42\ell\kappa + 18\ell$
This Paper Section 4.1	A	5	$6\ell \log_2 \ell + 22\ell + 9 \log_2 \ell + 28$
This Paper Section 5.1	A	4	$2\ell^2 + 4\ell \log_2 \ell + 19\ell + 4 \log_2 \ell + 16$

The overall progression of our proposed protocol on a high level is somewhat similar to that of Damgård et al [12] from Section 2.1. The parties share their private values in a bit decomposed manner. They cooperate in the construction of a mask related to the differing bit of greatest significance. Finally, the mask

Table 2. Overall round and communication complexity for secure comparison protocols

Presented in	Type	Overall Rounds	Bits transmitted overall
[12]	A	44	$1104\ell^2 \log_2 \ell + 1254\ell^2$
[13]	A	15	$1674\ell^2 + 30\ell$
[7]	A	10	$918\ell^2 + 2592\ell \log_2 \ell + 144\ell$
[7]	R	8	$162\ell^2 + 216\ell \log_2 \ell + 30\ell$
[9]	R	9	$60\ell^2 + 60\ell\kappa + 18\ell$
This paper Section 4.1	A	5	$7\ell \log_2 \ell + 26\ell + 11 \log_2 \ell + 32$
This paper Section 5.1	A	4	$2\ell^2 + 5\ell \log_2 \ell + 23\ell + 6 \log_2 \ell + 22$

is used to identify which of the values is greater through differences found with respect to earlier calculated intermediate values. These differences are used to build shares of the final result.

1.2 Organization

The rest of the paper is organized as follows: In Section 2, we present and discuss a closely related secure comparison protocol that is fundamental to several protocols analyzed in this paper. Next, in Section 3, we enumerate and describe the necessary properties of a secret sharing scheme adopted in our protocol, and provide discussion related to their complexity and composition in our work. We also introduce the adversary models and the security settings for our protocol. Then, we present our protocol in detail in Section 4, describing its functionality and procedure along with analysis concerning its complexity, security, and correctness. Our proposed protocols to follow are intended for various scenarios, and complexity preferences. Our first proposed protocol, discussed in section 4.1, deals with values that are privately held by two of the parties. Our second proposed protocol represents a potentially desirable trade-off in complexity and is discussed in Section 5.1. Finally, we discuss how our protocol can be generalized to larger groups of parties than merely three, and suggest an approach in which our protocol can be transformed to handle values which already exist in a bitwise shared format. We conclude by summarizing our contribution in Section 6, and indicate areas for future research directions.

2 Related Work

Yao's well known construction, garbled circuits and his associated motivator, the Millionaire's Problem, is an early approach to the possibility of a secure multiparty protocol to affect the comparison of two private values [2, 14]. There have been a number of concerns regarding the efficiency of this scheme, and there are many dramatically optimized solutions [15–18]. Yet, for large private data sets, they are still not feasible for a practical situation dealing heavily with arithmetic operations. Secret sharing approaches are generally preferable in this setting as noted in [19] and there are desires for stronger security guarantees.

Methods also exist for the implementation of this functionality based on homomorphic encryption [11, 20–22]. These operate on a very different set of principles and security guarantees than our present work, and in general, are prone to be less efficient than methods based on secret sharing schemes [23]. This is due primarily to the comparatively large keys required to assure security [24]. Without some fundamental change regarding this requirement and the assumptions of public key cryptography, is unlikely to be significantly reduced. While homomorphic encryption based secure comparison protocols are necessary in the two-party setting, in this paper, we focus on secret sharing based approaches which requires at least three parties.

The two main approaches based on secret sharing, as introduced earlier, are methods based on changing the representation of the shared values via bit decomposition [12], and those employing a series of tests related to the properties of finite field arithmetic [13]. Later improvements and optimizations to each of these methods have been published subsequent to the original proposition of the first of each approach. We have chosen to present and discuss the work of Damgård et al [12] due to the foundational place in the literature, clear representation of the underlying principles it represents, and relation to our general approach. Some of the optimizations on this method were proposed in [7, 9, 13].

The latter strategy, that of exploiting properties of finite field arithmetic seeks to affect a comparison through intermediate comparisons and some logic to bring the meaning of these intermediate comparisons together to form the desired solution. This method was introduced in the work of Nishide and Ohta [13]. Other optimizations have since come which cut back on the complexity and number of intermediate calculations necessary based on some restrictions to the domain of values which are shared and compared [7, 25]. We do not give these results as thorough a treatment due to the drastic difference in the approach, unrelated to our methods, though their complexities are important for consideration relative to the results of others as well as comparison with the complexity of our proposed solution which are all reported in Tables 1 and 2.

2.1 Damgård et al

The first known constant rounds result in this area is due to the work of Damgård et al [12]. In this setting, secretly shared values must first be bit decomposed among the parties involved in the computation. Alternatively, the values may exist as bitwise shares initially. This means the protocol takes as input bit decomposed shares of the private values to be compared. If this procedure is necessary, though expensive, it is potentially beneficial when other bit-wise operations may be seen as advantageous. If other bit-wise operations are desirable, the cost incurred in this scheme for bit decomposition may be amortized somewhat across all those sub-protocols which require it.

Addressing the comparison directly, it begins with a bitwise `xor` between the two values is computed. The result from the `xor` is used as the input to a prefix `or`. This results in a series of shared bits such that all the most significant bits are zero until the first difference in the original values was encountered. Following

that bit location down to the least significant bit, all bits are set. Next, from least to most significant bits, a pairwise subtraction is computed finding the difference between each bit and the next most significant bit in the series. The result from this step is a series of bitwise shares such that all shared bits are shares of zero with at most one exception: the location of the most significant difference between the two original inputs. The last several steps perform secure bitwise multiplications to extract the value of the original input at the location of the most significant difference. The key steps are given in Algorithm 1.

Algorithm 1: BIT-LT from Damgård et al [12]

Input: bitwise shares of integers a, b denoted $[a]_B, [b]_B$ with individual bits denoted $[a_0]_p \dots [a_{\ell-1}]_p$ where $\ell = \lceil \log_2 p \rceil$
Output: shares of the single bit result $[c]_p$

begin
  for $i = 0, \dots, \ell - 1$ **do**
    $[e_i]_p \leftarrow \text{XOR}([a_i]_p, [b_i]_p)$
  end
  $([f_{\ell-1}]_p, \dots, [f_0]_p) = \text{PRE}_\vee([e_{\ell-1}]_p, \dots, [e_0]_p)$
  $[g_{\ell-1}]_p = [f_{\ell-1}]_p$
  for $i = 0, \dots, \ell - 2$ **do**
    $[g_i]_p \leftarrow [f_i]_p - [f_{i+1}]_p$
  end
  for $i = 0, \dots, \ell - 1$ **do**
    $[h_i]_p \leftarrow \text{MULT}([g_i]_p, [b_i]_p)$
  end
  $[h]_p \leftarrow \sum_{i=0}^{\ell-1} [h_i]_p$
  Output $[h]_p$
end

Though there is a fairly high computational complexity and communication cost, this important result demonstrates constant rounds secure comparison is indeed possible and well within feasibility.

3 Preliminaries

In our work, we make use of the additive secret sharing scheme. The approach under additive secret sharing makes use of a different set of mathematical principles to achieve secure multiparty computation than the more widely referenced Shamir scheme, though modular arithmetic still lies at the core of its security. The underlying security is dependent on the fact that adding any value to a uniformly and randomly selected value, modulus a value delimiting the group, N, is still uniformly random. Therefore, it is impossible to say what the non-random component of the sum was, when considering only the resulting sum. In this context, the sum is unconditionally secure since any adversary, unbounded by

limits on computational power, can do no better than a random guess on what the original values may have been. This security is similar in principle to the unconditional security guarantees of the one-time pad, which has the weakness of inability to reuse the pad due to leaks in information. In the setting of secret sharing, no such concerns exist. For each and every share, new uniform random values are selected, used, and distributed, which removes the risk for information leakage from padding value reuse. If a shared value is revealed, its former shares may no longer be securely used without risk of information disclosure.

3.1 Operations and Notations in Linear Secret Sharing Schemes

We require that any secret sharing scheme to be used have the ability to perform the following operations:

- Share: given a particular value x, generate shares of x, denoted by $[x]_N^{P_j}$, in a group defined by a modulus N for each party P_j. This must be done in a way that they can be uniquely recombined in a method applicable to the scheme to reconstruct the original value.
- Reveal: a sufficient number of shares $[x]_N^{P_j}$ can be recombined to reveal the original value x.
- Add with a public constant: given shares $[x]_N^{P_j}$, and a public constant c, execute the necessary operations to calculate $c+[x]_N^{P_j} = [c+x]_N^{P_j}$. In a linear secret sharing scheme, this can be done locally.
- Add shares: given two shared values $[x]_N^{P_j}$ and $[y]_N^{P_j}$, calculate the sum of their values in a shared form. This can be executed without communications by using the addition operation implemented in the secret sharing scheme, i.e., $[x]_N^{P_j} + [y]_N^{P_j} = [x+y]_N^{P_j}$.
- Multiplication by a public constant: given shares $[x]_N^{P_j}$, and a public constant c, execute the necessary operations to calculate $c[x]_N^{P_j} = [cx]_N^{P_j}$. In a linear secret sharing scheme, this can be done locally without communication among the parties.

3.2 Random Shift

In our protocol, we will make use of a random permutation. This permutation π is encoded as an integer in $\mathbb{Z}_\ell$ requiring $\log_2 \ell$ bits. We use a specific kind of cyclic permutation called a circular shift. It is important to note that this is not a fully random permutation, in the sense that the placement of all values in the vector after permutation are totally independent from their location previously. In our permutation, we only are concerned about shifting one element in an array by a uniformly random amount. In a circular shift, every element that was previously adjacent to another element will maintain that relation in the permuted vector. Given a value encoding the permutation, the values are circularly shifted right that number of index locations. We denote this as applied to a vector v by $\text{shift}_\pi(v)$, and similarly for an inverse $\text{shift}_\pi^{-1}(v)$. We only require that one

location, for our protocol, the location of the bit flagging the index of the most significant bit difference, be able to be moved to any other array location with a uniform probability. In our use, every other element in the array is meaningless and uniformly random by design, and therefore does not leak any information, whether it maintains some relative position with respect to the bit location of interest or not. Our use is similar to this type of shift permutation in [7]. Consider a trivial example for a vector v consisting of 4 entries $v = \langle 0, 1, 2, 3\rangle$; thus, $\pi \in \{0, 1, 2, 3\}$ or equivalently in binary form $\pi \in \{00, 01, 10, 11\}$. If we are interested in the ability to move the bit position 0 to any position within the array, consider the uniform random choice between the options presented in the application of the permutation for each the options demonstrated in Table 3.

Table 3. Small Cyclic Permutation Illustration

Initial Vector	Permutation	Permuted Vector
$\langle 0, 1, 2, 3\rangle$	00	$\langle 0, 1, 2, 3\rangle$
$\langle 0, 1, 2, 3\rangle$	01	$\langle 3, 0, 1, 2\rangle$
$\langle 0, 1, 2, 3\rangle$	10	$\langle 2, 3, 0, 1\rangle$
$\langle 0, 1, 2, 3\rangle$	11	$\langle 1, 2, 3, 0\rangle$

Clearly, 0 can be in any vector location with equal probability following the application of the permutation. This amounts to a uniform random shift in the elements which would effectively hide its previous location. Again, we do not care at all about any of the values of other elements, or their locations - in either an objective or relative sense. Thus, this permutation approach efficiently, securely, and correctly achieves our desire to hide the sole piece of information about which we have concern.

3.3 Security Definitions and Adversary Model

Aside from the correctness and efficiency of any of the extant protocols, an important consideration is the security guarantee that can be derived from their execution. Toward the end of proving security, we have generally followed the proof style and conventions, as well as the information theoretic expansions on security proof developed by Crépeau et al [26] and Goldreich [27]. The goal in this setting is to demonstrate for an arbitrary functionality f, there exists an equivalence of information disclosure between the ideal execution of a protocol and a real execution of a protocol implementing the same ultimate functionality.

The "ideal" model is defined as a protocol implementing the desired functionality f corresponding to the desired function the parties wish to compute, in a highly idealized setting. As an example, an ideal protocol trivially implemented for two semi-honest parties by algorithm $A = A_1, A_2$ consists of each party sending their private inputs x and y to a trusted third party implementing functionality f. These private inputs x and y are associated with their respective input domains X and Y. In this ideal model, the trusted third party executing

f simply returns the output $\underline{u} \in U$ and $\underline{v} \in V$ to A_1 and A_2 respectively, where U and V are the output domains. There is also an auxiliary input $z \in \{0,1\}^*$ available to both parties. In the case that a component of A is compromised and behaves maliciously, the only possible changes that can be done are alterations in their input or output. The u and v are underscored in this presentation so that they can be notationally distinguished from the real model outputs. This model is denoted

$$(\underline{u}, \underline{v}) = \text{IDEAL}_{f,A(z)}(x, y)$$

The "real" world model is defined as a protocol implementing functionality f corresponding to the desired result via the execution of a protocol p consisting of admissible algorithms $B = B_1, B_2$. The private inputs, $x \in X$, and $y \in Y$, are accepted as parameters. An auxiliary input $z \in \{0,1\}^*$ is also available to both parties. The return of this operation is the set of outputs from the execution of the protocol which follows from the interaction of B_1 and B_2. The real model is symbolically represented as

$$(u, v) = \text{REAL}^p_{f,B(z)}(x, y)$$

What is required for perfect security in this case is the demonstration of the equivalence of information disclosure via messages and outputs in interaction which is denoted as

$$\text{IDEAL}_{f,A(z)}(x, y) \equiv \text{REAL}^p_{f,B(z)}(x, y)$$

Here, the protocol implemented in the real model is perfectly secure if there exists a pair of algorithms $A = A_1, A_2$ admissible in the ideal model for every set of algorithms $B = B_1, B_2$ which are admissible in the real model (where the same parties are honest), for all $x \in X, y \in Y, z \in \{0,1\}^*$. For this definition, "admissibility" or "equivalence" is related to information disclosed in the process of protocol execution. An algorithm is admissible if no additional information can be gleaned from the communications implementing its functionality than would be the case in the ideal setting. This is to say that for all outputs in the domains U and V, and all possible inputs from the domains X, and Y, as well as the additional input Z:

$$P(\underline{u}, \underline{v}|x, y, z) = P(u, v|x, y, z)$$

which simply indicates that the joint probability distributions for the outputs are indistinguishable, regardless of the computational power of the adversary. This means there exists no method by which they can be distinguished or separated at all. Thus, the real model is information theoretically secure. This is instrumental as we make arguments for the uniformly random distributions from which our messages, shares, randomizations, and outputs are constructed.

4 The Proposed Protocol

In this paper, we propose multiparty secure comparison protocols. Unlike the existing secret sharing based secure comparison protocols, the inputs and inter-

mediate computations are not symmetric among the three parties. Expansions to larger numbers of parties are fairly trivial, and can be done based directly on the three-party examples given in the following section. Each protocol considers a different scenario.

- The first protocol allows for a group of three parties to securely compare two private inputs held by two of the three parties. The purpose of the third party is to facilitate secret sharing based secure computations.
- The second protocol presents an option concerning the protocol complexity, with a lower number of rounds.
- For both of the preceding cases, we present an extension of our protocols, and it assumes that the three parties do not know the values they compare, and that they are already shared in a bit decomposed format. This is similar to [12], listed in Tables 1 and 2.

Each proposed protocol targets a unique class of applications. However, they together cover most scenarios related to secure comparison based applications.

4.1 Key Steps and Correctness Analysis for Protocol 1

The main steps of our first proposed protocol are presented in Algorithm 2. Without loss of generality, P_1 has a private input a and P_2 has a private input b. At the end, the protocol returns the secret shares of the comparison result to only P_1 and P_2. The comparison result 1 indicates $a \geq b$, and 0 otherwise. At the beginning of the algorithm, an indexing variable i is defined for arrays of shared values. Unless otherwise noted, i is used to span the entire array. There are two instances in Step 5 when this is not the case. Overall, a few key design principles are important to recall: 1) `xor` can be done locally in $\mathbb{Z}_2$, without secure multiplication or communication, 2) we employ a summation of bits to avoid other costly multiplications and required communication, and 3) switching groups in which the shares are constructed allows us to further reduce communication costs.

Aside from groups $\mathbb{Z}_N$ and $\mathbb{Z}_2$, we also make use of a group $\mathbb{Z}_{N_2}$. We define N_2 to be a prime such that $\lceil \log_2 \ell \rceil + 1 < \log_2 N_2 < \lceil \log_2 \ell \rceil + 2$. This is due to the use of the values which will be shared in this group. By inspection in the protocols to follow, specifically Step 5, the maximum possible value which may be shared in this group is $2(\ell + 1) - 1$ related to γ'_0. Thus, the modulus should have sufficient bitwidth to represent these values, and this gives us the bounds we have defined for its modulus. It is known that a prime must exist in this range from the proof of Bertrand's Postulate which states that, for all $n > 3$ there exists at least one prime p such that $n < p < 2n$ [28–30].

The protocol unfolds as follows along with correctness justifications. In the following discussions, note that we refer to values without the level of specificity that is given in the algorithm. For example, shares of a belonging to P_1 and P_2 in $\mathbb{Z}_2$ may simply be referred to as $[a]$. For full details please reference the given step in the algorithm.

Algorithm 2: $\mathrm{SC}(\langle P_1, a\rangle, \langle P_2, b\rangle, \langle P_3, \perp\rangle) \rightarrow (\langle P_1, [f]_N^{P_1}\rangle, \langle P_2, [f]_N^{P_2}\rangle)$

Input: Public info: N and N_2, $N > N_2$, N is an integer and the modulus of the secret sharing scheme, ℓ is the required bitwidth for the domain of a and b, $N > 2^\ell$. N_2 is a prime such that $\lceil \log_2 \ell \rceil + 1 < \log_2 N_2 < \lceil \log_2 \ell \rceil + 2$, $0 \leq i \leq \ell$, and $j \in \{1, 2\}$

Output: f is secretly shared between P_1 and P_2. $f = 1$ if $a \geq b$; otherwise, $f = 0$

1 P_1

(a) $a = 2a + 1$
(b) $s_a \leftarrow \sum_i a_i$
(c) Generate $r_i, r_i' \in_R \mathbb{Z}_2$, $r'' \in_R \mathbb{Z}_2$, $\tau_i \in_R \mathbb{Z}_{N_2}^*$, and random shift π
(d) Generate $[a_i]_2^{P_j}$, and $[s_a]_{N_2}^{P_j}$
(e) Send $[a_i]_2^{P_2}$, $[s_a]_{N_2}^{P_2}$, r_i, r_i', r'', τ_i and π to P_2

2 P_2

(a) $b = 2b$
(b) Generate $[b_i]_2^{P_j}$
(c) Send $[b_i]_2^{P_1}$ to P_1

3 P_j

(a) $[e_i]_2^{P_j} \leftarrow [a_i]_2^{P_j} + [b_i]_2^{P_j}$
(b) $[e_i]_2^{P_1} \leftarrow r_i - [e_i]_2^{P_1}$, if $r_i = 1$
(c) Send $[e_i]_2^{P_j}$ to P_3

4 P_3

(a) $e_i \leftarrow [e_i]_2^{P_1} + [e_i]_2^{P_2}$
(b) Generate $[e_i]_{N_2}^{P_j}$
(c) Send $[e_i]_{N_2}^{P_j}$ to P_j

5 P_j

(a) $[e_i]_{N_2}^{P_1} \leftarrow r_i - [e_i]_{N_2}^{P_1}$, if $r_i = 1$
(b) $[e_i]_{N_2}^{P_2} \leftarrow -[e_i]_{N_2}^{P_2}$, if $r_i = 1$
(c) $[\gamma_\ell']_{N_2}^{P_j} \leftarrow [e_\ell]_{N_2}^{P_j}$
(d) $[\gamma_i']_{N_2}^{P_j} \leftarrow [\gamma_{i+1}']_{N_2}^{P_j} + [e_i]_{N_2}^{P_j}$, for $i = \ell - 1, \ldots, 0$
(e) $[\gamma_\ell]_{N_2}^{P_j} \leftarrow [\gamma_\ell']_{N_2}^{P_j}$
(f) $[\gamma_i]_{N_2}^{P_j} \leftarrow [\gamma_{i+1}']_{N_2}^{P_j} + [\gamma_i']_{N_2}^{P_j}$, for $i = \ell - 1, \ldots, 0$
(g) $[\gamma_i]_{N_2}^{P_1} \leftarrow [\gamma_i]_{N_2}^{P_1} - 1$
(h) $[u_i]_{N_2}^{P_j} \leftarrow \tau_i [\gamma_i]_{N_2}^{P_j}$
(i) $[v_i]_{N_2}^{P_j} \leftarrow \mathrm{Shift}_\pi\left([u_i]_{N_2}^{P_j}\right)$
(j) Send $[v_i]_{N_2}^{P_j}$ to P_3

6 P_3

(a) $v_i = [v_i]_{N_2}^{P_1} + [v_i]_{N_2}^{P_2} \bmod N_2$
(b) Find the unique index k, where $v_k = 0$
(c) Generate $[h_i]_2^{P_j}$, where $h_k = 1$ and $h_i = 0$ for $i \neq k$
(d) Send $[h_i]_2^{P_j}$ to P_j

7 P_j

(a) $[h]_{2,B}^{P_j} \leftarrow \text{shift}_\pi^{-1}\left([h]_{2,B}^{P_j}\right)$
(b) $[h_i']_2^{P_j} \leftarrow [h_i]_2^{P_j} - [a_i]_2^{P_j}$
(c) $[h_i']_2^{P_1} \leftarrow r_i' - [h_i']_2^{P_1}$, if $r_i' = 1$
(d) Send $[h_i']_2^{P_j}$ to P_3

8 P_3

(a) $h_i' \leftarrow [h_i']_2^{P_1} + [h_i']_2^{P_2}$
(b) Generate $[h_i']_{N_2}^{P_j}$
(c) Send $[h_i']_{N_2}^{P_j}$ to P_j

9 P_j

(a) $[h_i']_{N_2}^{P_1} \leftarrow r_i' - [h_i']_{N_2}^{P_1}$, if $r_i' = 1$
(b) $[h_i']_{N_2}^{P_2} \leftarrow -[h_i']_{N_2}^{P_2}$, if $r_i' = 1$
(c) $[s_a']_{N_2}^{P_j} \leftarrow \sum_i [h_i']_{N_2}^{P_j}$
(d) $[f]_{N_2}^{P_j} \leftarrow [s_a]_{N_2}^{P_j} - [s_a']_{N_2}^{P_j}$
(e) $[f]_{N_2}^{P_1} \leftarrow [f]_{N_2}^{P_1} + 1$
(f) $[f]_{N_2}^{P_j} \leftarrow 2^{-1}[f]_{N_2}^{P_j}$
(g) $[f]_{N_2}^{P_1} \leftarrow r'' - [f]_{N_2}^{P_1}$, if $r'' = 1$
(h) $[f]_{N_2}^{P_2} \leftarrow -[f]_{N_2}^{P_2}$, if $r'' = 1$
(i) Send $[f]_{N_2}^{P_j}$ to P_3

10 P_3

(a) $f \leftarrow [f]_{N_2}^{P_1} + [f]_{N_2}^{P_2}$
(b) Generate $[f]_N^{P_j}$
(c) Send $[f]_N^{P_j}$ to P_j

11 P_j

(a) $[f]_N^{P_1} \leftarrow r'' - [f]_N^{P_1}$, if $r'' = 1$
(b) $[f]_N^{P_2} \leftarrow -[f]_N^{P_2}$, if $r'' = 1$

- *Step 1*: In the first step, P_1 generates the necessary values for later permutation and randomization. The private input of P_1 is also doubled and incremented before shares are generated. Finally, the shares for P_2 along with other values for later use are transmitted to P_2.
- *Step 2*: This step consists of P_2 generating shares of twice the private input it holds and sending one share to P_1. The two private values are modified in this manner (either doubled or doubled and incremented) in order to inject one sure difference into the calculation in the values to be compared. This is done specifically in the least significant bit position. In this way we avoid what would have otherwise been a risk for protocol failure leading to an information leak in the case of equality.
- *Step 3*: In Step 3, all parties compute a bitwise `xor`, and P_1 additionally randomizes its shares by the uniform random bits which make up r_i. Both P_1 and P_2 send the resulting shares to P_3.
- *Step 4*: P_3 uses the two shares received to rebuild the re-randomized secret and generate new shares, now in a group defined by N_2. This amounts to a mapping of shares between groups $\mathbb{Z}_2 \to \mathbb{Z}_{N_2}$. Note that, due to the additional randomization from Step 3, no information is leaked in this process.
- *Step 5*: Recall that the shares from P_1 were randomized, so when they are rebuilt by P_3, no information will be leaked. This randomization must be reversed so that the original values, now secretly shared in a new group, can be reclaimed. This is achieved through sharing the randomization vector r between P_1 and P_2. When $r_i = 1$, P_1 computes $r_i - [e_i]$ and P_2 simply computes the additive inverse of its share. This is accomplished in Steps 5(a) and 5(b) of Algorithm 2. These two parties now hold bitwise shares of the `xor` of their private inputs, each as elements in $\mathbb{Z}_{N_2}$. In Step 5(d), the protocol proceeds by computing the prefix sum of the bits from most to least significant shares of bits. This results in a series of shares such that the most significant bits are shares of zero until the first bit difference is encountered, which is one. Following this position, all bits are non-zero and non-decreasing in value with each successive bit. In Step 5(f), to ensure that exactly one value in the array is equal to one, we iterate through the bit positions again performing a prefix sum of the values from most to least significant bit positions. This is the justification for the bounds which we have given for N_2. If a and b have no bits in common, the vector $[e_i]$ will consist of shares of all ones. Thus, the prefix sum of the shares of these values will yield, at most, $\ell+1$. Repeating this operation will yield a value of at most $2\ell+1$; as a result, this serves as the defining factor for our bounds defined for N_2. Finally, P_1 and P_2 subtract one from every share of this vector, thereby ensuring that every value in the vector is non-zero with one exception, the location of the most significant bit difference. The values contained in the vector are multiplied by the vector τ consisting of random values which are known to P_1 and P_2. This is done to hide the intermediate calculated values from P_3. Since we require N_2 to be prime, these values in τ_i will not yield a product equal to 0 under modulus N_2 when multiplied with any other group element, except 0, which is the flag of the location we wish to uniquely

preserve. Now, every array value related to every bit position is a non-zero and uniformly random value with one exception, the position related to the most significant bit difference. Finally, to hide this information from P_3, the vector of values $[u_i]$ is shifted according to the random value π agreed upon between P_1 and P_2.

- *Step 6*: Coming to Step 6, P_3 rebuilds the vector of these values and finds the unique index k containing the value zero. This player then constructs a bitwise series of shares in $\mathbb{Z}_2$ such that all shares are shares of zero with the exception of the shares indexed by k. These shares are sent to their parties.
- *Step 7*: Upon receipt of these shares, the parties perform an inverse shift on the received vector to place the sole location containing a share of one back into the position related to the most significant bit difference. They then compute, for each location, the difference between the newly received vector of shares and the shares of P_1's original input. P_1 additionally randomizes this result with the vector r', similar to what was done in Step 3(b). Similar again to what has preceded, P_3 reconstructs and performs another mapping in the generation of new shares between groups $\mathbb{Z}_2 \rightarrow \mathbb{Z}_{N_2}$. These have the intermediate randomizations removed by the parties in Step 9(a) and 9(b) as discussed with respect to Step 5(a) and 5(b).
- *Step 8*: Here P_3 once again serves as an oblivious mapper between sharing groups and distributes the shares randomized results of the calculation of h'.
- *Step 9*: All values in the vector h' are summed by P_1 and P_2, and assigned to shares of $[s'_a]$. The difference $[s_a] - [s'_a]$ is calculated. Because there is exactly one bit difference between a and h' in their binary representations, the sums of the set bits within these two values will have a difference of exactly magnitude 1. Calculating the difference $[s_a]-[s'_a]$ results in either one or its additive inverse in $\mathbb{Z}_{N_2}$ being assigned to shares of f. Finally, since the result of the computation so far is $f \in \{1, -1\}$ shared in $\mathbb{Z}_{N_2}$, it is necessary to map this set to $f \in \{0, 1\}$, and bring the shares into the domain for the overall secret sharing scheme. The first part of these last steps is achieved through P_1 adding the value one to its share, thereby incrementing the set of possible values to get shares of either zero or two, and then both parties multiply their shares by the multiplicative inverse of two. In the case that f is zero, it will remain unchanged, and in the case that f is two, it will be reduced to one. Thus, the correct functionality has been achieved and $f = 1$ iff $a \geq b$. All that remains is to map the shares into the appropriate group. As we have done previously, this shared value $f \in \mathbb{Z}_{N_2}$ is re-randomized with another bit r''. This value is sent to P_3.
- *Step 10:* For the last time, P_3 reveals the randomized f and builds shares of that revealed quantity in $\mathbb{Z}_N$ and returns these to the other parties.
- *Step 11:* As the last step of our proposed protocol, P_1 and P_2 remove the temporary additional randomization from the shared result $[f]_N$ and hold their shares of the desired result.

4.2 Security Analysis

For our security formulations and proofs to follow, we consider adversaries in the semi-honest or honest but curious model. We consider each player's role independently, due to the asymmetry used in our protocol. For each case, we will analyze the protocol focusing on communicated information and opportunities for leaks, and we will conclude with a summary discussion of the equivalence necessary as summarized in Section 3.3 required for proving perfect security. Specifically, we discuss the steps toward a proof of the following equivalence assuming semi-honest adversaries

$$\text{IDEAL}_{f,A(z)}(x,y) \equiv \text{REAL}^{p}_{f,B(z)}(x,y)$$

for our set of three algorithms $A = A_1, A_2, A_3$ admissible in the ideal model for every set of algorithms $B = B_1, B_2, B_3$ which are admissible in the real model.

Semi-honest P_1 The view of P_1 during the execution of the proposed protocol includes public domain information for inputs and outputs, and the value a which is the private input for P_1. Additionally, P_1 generates random bit vectors r and r', a random bit r'', the vector of random values in $\mathbb{Z}_{N_2}$ τ, and the random shift π. In the course of execution, more values are exchanged; these consist of shares of b, e, h, and h'.

- Step 1(c) and 1(d) are those which represent the generation of all the values P_1 will share with P_2. Specifically, these are: $r_i, r'_i, r'', \tau_i, \pi, [a_i]_2^{P_2}$, and $[s_a]_{N_2}^{P_2}$.
- In Step 2(c), $[b_i]_2^{P_1}$, the P_1's share of P_2's bitwise shared input, is received.
- Step 4(c) consists of the receipt of the vector $[e_i]_{N_2}^{P_1}$.
- The vector $[h_i]_2^{P_1}$ is received in Step 6(d).
- In Step 8(c), the vector $[h'_i]_{N_2}^{P_1}$ is received.
- Step 10(c) is the final communication in the scheme and consists of the receipt of $[f]_N^{P_1}$.

Since for each of the values listed above in the Steps 2-10, P_1 only receives one share, it is impossible for P_1 to rebuild the actual values. This is immediate from the security guarantees of the underlying secret sharing scheme. Thus, possession of a single share cannot leak information, and P_1 can do no better than random guess on the underlying value.

The values generated in Step 1(c) and 1(d) are all uniformly random, and each consists of use of the randomness source z provided in the real implementation of the protocol, which is permissible under the scheme used in the proof system. This use of the randomness source in no way violates the equivalence sought with the ideal model. These values are either shared via the generation of shares (e.g., only one of the two shares is sent to P_2) or sent directly to P_2 without any kind of obfuscation (e.g., the vectors of random bits r and r', the vector of random values in $\mathbb{Z}_{N_2}$ τ, and the random shift π). These values, since they consist of uniform random values or the result of calculations which are also

uniform random, do not pose any threat to the privacy of the protocol through their use in the manner we have described in the context of the protocol.

The final value retained as the result of the protocol's completion is uniformly random and represents a share of the desired outcome from the evaluation of the functionality we implement. f is an element in $\mathbb{Z}_2$, and its shares are in a larger group. Again, it is impossible to rebuild the underlying secret, or learn anything at all about it, from the possession of a single share. This is once more immediate from the underlying secret sharing scheme. Therefore, P_1 learns nothing beyond what is known at the beginning of the execution of the protocol, namely, the value of the private input already held a, the public domain information, and anything which may be deduced from these data.

Semi-honest P_2 The roles of P_1 and P_2 are fairly similar, though not identical. P_2's view of the protocol differs from P_1 in only two ways, receipt from P_1 of some of the random values to be used to hide information from P_3, and the manner in which some of these values are used. The total execution view consists of the public domain information for inputs and outputs, and the value b which is the private input for P_2. P_2 receives random bit vectors r and r', a random bit r'', the vector of random values τ_i in $\mathbb{Z}_{N_2}$, and the random shift amount π. In the course of execution, more values are exchanged. These consist of shares of a, s_a, e, h, and h'.

- Step 1(e), all values P_1 shared with P_2 are received. Specifically, these are: $r_i, r'_i, r'', \tau_i, \pi, [a_i]_2^{P_2}$, and $[s_a]_{N_2}^{P_2}$.
- Step 4(c) consists of the receipt of the vector $[e_i]_{N_2}^{P_2}$.
- The vector $[h_i]_2^{P_2}$ is received in Step 6(d).
- In Step 8(c), the vector $[h'_i]_{N_2}^{P_2}$ is received.
- Step 10(c) is the final communication in the protocol and consists of the receipt of $[f]_N^{P_2}$.

The understanding of the security of the protocol, with respect to an honest-but-curious P_2, is essentially the same as the case for P_1. Since P_2 is never in possession of both shares of any of the secret shared values, the values represented by the shares are impossible to reconstruct due to the properties of the secret sharing scheme. The possession of the true values of, for example, r_i in no way compromises any security principles since the value is used only to hide information from P_3 and allows for the removal of this additional randomization after the shares have been mapped into their new domain. Specifically, where P_1 calculates, in Step 5(a), $[e_i]_{N_2}^{P_1} \leftarrow r_i - [e_i]_{N_2}^{P_1}$, if $r_i = 1$, P_2 calculates $[e_i]_{N_2}^{P_2} \leftarrow -[e_i]_{N_2}^{P_2}$, if $r_i = 1$.

Since P_2, parallel to many of the ideas presented with respect to P_1, cannot rebuild any of the secret shared values, learns nothing of additional information which should not be gained from the randomization values and the underlying values or result beyond what can be surmised from it's privately held input and public information. Thus, the protocol is secure against a semi-honest P_2.

Semi-honest P_3 The role of P_3 in the overall protocol is very different from the other two. The view of P_3 during the execution of the proposed comparison protocol includes the public domain information for inputs and outputs, receipt of the shares of the shared value v, and bitwise shared values e, and h.

- In Step 3(c), the shares $[e_i]_2^{P_1}$ and $[e_i]_2^{P_2}$. In Step 7(d), the shares $[h'_i]_2^{P_1}$ and $[h'_i]_2^{P_2}$ are received. These are both bitwise shares of values related to a and b, which are recombined to reconstruct the underlying values.
- Shares $[v_i]_{N_2}^{P_1}$ and $[v_i]_{N_2}^{P_2}$ are received in Step 5(j) to reconstruct the randomized values in $\mathbb{Z}_{N_2}$.
- Shares $[f]_{N_2}^{P_1}$ and $[f]_{N_2}^{P_2}$ received in Step 9(i) are two random integers in $\mathbb{Z}_{N_2}$.

P_3 receives no shares of the output of the protocol, and holds no shares of the inputs, so the above mentioned messages comprise the total view of the execution of the protocol from the perspective of P_3. The shares of these values have been built from the original values based on the algorithm standard for the secret sharing scheme in use. Since P_3 receives all necessary information to rebuild values e and h', and in fact does rebuild them, there would be a significant leaking of information for these cases if they were not additionally re-randomized by uniform values in $\mathbb{Z}_2$ before the shares of P_1 were sent to P_3. This additional layer of randomization allows for P_3 to rebuild the shares and generate new shares in a different group, namely $\mathbb{Z}_{N_2}$, without gaining any knowledge about the true values themselves. In the case of Step 5, and the reception of the shares of v, the only information of significance is the index of the bit position related to the most significant difference between the inputs. This position is flagged by being the only zero element in the array. This position is hidden by the permutation π, so even though P_3 once again rebuilds and maps the values to a different group, no meaningful information can be gleaned.

P_3 has no input and only processes results from others' shared inputs. Since the results have been uniformly re-randomized, they retain no information of the actual result beyond what can be ascertained in an ideal case. The best this player can do, independent on the availability of arbitrary computational power, is randomly guess in the domain what the correct values can be, without any means of verifying if the guessed values are indeed the true values. Thus, the protocol with respect to a semi-honest P_3 is information theoretically secure.

Security Summary Since for each of the preceding three cases, analyzing the set of algorithms $B = B_1, B_2, B_3$, these were seen to be admissible in the sense that they neither yield nor leak information beyond that which is desired, and constitute a correct implementation of the desired functionality as discussed in Section 4.1, they are equivalent to a set of algorithms $A = A_1, A_2, A_3$ which are executed in an idealized setting. This immediately leads to their security in an information theoretic sense, the desired property we wished to demonstrate.

4.3 Complexity

The complexity of our protocol is analyzed from two perspectives: that of round complexity, and communications. We define a computational *round* as consisting of any arbitrary amount of local computation accompanied by at most one send and receive cycle, as established and explained in [31]. The round complexity of our protocol is simply a constant since the number of cooperative steps executed by the overall protocol is fixed and independent of the size of the input values. Specifically, the number of required rounds is 5. All the following step numbers refer to the algorithm of our protocol given in Algorithm 2.

The first round consists of Steps 1-3 since this it is in this step that the parties are generating shares of values to be used, exchanging them, performing some computation and transmitting the response to P_3. This is analogous to one send and receive cycle with an arbitrary amount of local computation accompanying. During this phase, the most complex values to be communicated are generated and sent. Specifically, in Step 1(e), generating and sending $[a_i]_2^{P_2}$, $[s_a]_{N_2}^{P_2}$, r_i, r'_i, r'', τ_i and π requires the transmission of $\ell \log_2 \ell + 5\ell + 3\log_2 \ell + 9$ bits. Step 2(c) requires only $\ell + 1$ bits in communication, and concludes round 1 since the next step is dependent on the values transmitted in this step. Now we get into the more strict send and receive relationship for the rest of the required rounds. Steps 3 and 4 together form a round since P_1 and P_2 calculate and send values to P_3, and receive back from P_3 a response based on those values. The same is true for pairs of Steps 5-6 and 7-8. Step 3(c) requires $2(\ell + 1)$ bits in transmission, and 4(c) requires $2(\ell + 1)(2 + \log_2 \ell)$ bits. The next round requires $2(\ell+1)(2+\log_2 \ell)$ bits for Step 5(i) and $2(\ell+1)$ bits for Step 6(d). The penultimate round requires $2(\ell+1)$ bits to be sent in Step 7(d), and in step 8(c) $2(\ell + 1)(2 + \log_2 \ell)$ bits are communicated. The final round consists of Steps 9 and 10 in which $2(2+\log_2 \ell)$ and 2ℓ bits are communicated respectively. The last step follows immediately from the values received in Step 10, and requires no further communication. Therefore, Step 11 is considered as a final part of round 5. For the total communication complexity, we consider the number of bits need to be transmitted, according to the steps in which they occur:

- Step 1: $3(\ell + 1)$ bits for r_i, r'_i, and $[a_i]_2^{P_2}$, 1 for r'', $(\ell + 1)(2 + log_2\ell)$ for τ_i, $1 + \log_2 \ell$ for π, and $2 + \log_2 \ell$ for $[s_a]_{N_2}^{P_2}$. The total: $\ell \log_2 \ell + 5\ell + 3\log_2 \ell + 9$
- Step 2: $\ell + 1$ for $[b_i]_2^{P_1}$
- Step 3: $2(\ell + 1)$ for $[e_i]_2^{P_1}$ and $[e_i]_2^{P_2}$
- Step 4: $2(\ell + 1)(2 + \log_2 \ell)$ for $[e_i]_{N_2}^{P_1}$ and $[e_i]_{N_2}^{P_2}$
- Step 5: $2(\ell + 1)(2 + \log_2 \ell)$ for $[v_i]_{N_2}^{P_1}$ and $[v_i]_{N_2}^{P_2}$
- Step 6: $2(\ell + 1)$ for $[h_i]_2^{P_1}$ and $[h_i]_2^{P_2}$
- Step 7: $2(\ell + 1)$ for $[h'_i]_2^{P_1}$ and $[h'_i]_2^{P_2}$
- Step 8: $2(\ell + 1)(2 + \log_2 \ell)$ for $[h'_i]_{N_2}^{P_1}$ and $[h'_i]_{N_2}^{P_2}$
- Step 9: $2(2 + \log_2 \ell)$ for $[f]_{N_2}^{P_1}$ and $[f]_{N_2}^{P_2}$
- Step 10: 2ℓ for $[f]_N^{P_1}$ and $[f]_N^{P_2}$

Summing these leads directly to the aforementioned figure for the overall communication requirement of our protocol: $7\ell \log_2 \ell + 26\ell + 11 \log_2 \ell + 32$ bits.

5 Protocol Variations

In this section, we address a couple alternatives for slight changes which may be made to our protocol to alter its complexity or expand its generality. The first such proposed alteration allows for a trade-off in complexity which could be desirable in some settings, trading the elimination of a computational round for a slightly increased communication complexity. The other suggests a manner in which it is possible to generalize our three-party protocol to any number of parties greater than three with only slight increase in communication costs.

5.1 Variation for Alternative Complexity

If round complexity is of greater concern than overall communication costs, we offer a trade-off which can be implemented with respect to our general protocol. There is a one round complexity reduction which is bought at the expense of an increase in the communication complexity. In order to affect this change, the protocol as given previously stands for Steps 1-7. Beginning with Step 8, alterations are necessary. We present the Algorithm 3 which consists of the necessary steps to replace in Algorithm 2. The only other alterations occur in Step 1(d)-(e) where s_a is shared in $\mathbb{Z}_N$ rather than $\mathbb{Z}_{N_2}$. In addition, N is now required to be odd. Finally, Steps 10 and 11 are no longer necessary, and the protocol ceases with completion after the finish of Step 9.

Algorithm 3: Replacement steps for alternative complexity for $\mathrm{SC}(\langle P_1, a\rangle, \langle P_2, b\rangle, \langle P_3, \perp\rangle) \rightarrow (\langle P_1, [f]_N^{P_1}\rangle, \langle P_2, [f]_N^{P_2}\rangle)$

Input: Public info: N and N_2, $N > N_2$, N is an odd integer and the modulus of the secret sharing scheme, ℓ is the required bitwidth for the domain of a and b, $N > 2^\ell$. N_2 is a prime such that $\lceil \log_2 \ell \rceil + 1 < \log_2 N_2 < \lceil \log_2 \ell \rceil + 2$, $0 \leq i \leq \ell$, and $j \in \{1, 2\}$

Output: f is secretly shared between P_1 and P_2. $f = 1$ if $a \geq b$; otherwise, $f = 0$

8 P_3

(a) $h'_i \leftarrow [h'_i]_2^{P_1} + [h'_i]_2^{P_2}$
(b) Generate $[h'_i]_N^{P_j}$
(c) Send $[h'_i]_N^{P_j}$ to P_j

9 P_j

(a) $[h'_i]_N^{P_1} \leftarrow r'_i - [h'_i]_N^{P_1}$, if $r'_i = 1$
(b) $[h'_i]_N^{P_2} \leftarrow -[h'_i]_N^{P_2}$, if $r'_i = 1$
(c) $[s'_a]_N^{P_j} \leftarrow \sum_i [h'_i]_N^{P_j}$
(d) $[f]_N^{P_j} \leftarrow [s_a]_N^{P_j} - [s'_a]_N^{P_j}$
(e) $[f]_N^{P_1} \leftarrow [f]_N^{P_1} + 1$
(f) $[f]_N^{P_j} \leftarrow 2^{-1}[f]_N^{P_j}$

The communications required in this version of the protocol are as follows:

- Step 1: $3(\ell+1)$ bits for r_i, r'_i, and $[a_i]_2^{P_2}$, 1 for r'', $(\ell+1)(2+log_2\ell)$ for τ_i, $1+\log_2\ell$ for π, and ℓ for $[s_a]_N^{P_2}$ the total is $\ell\log_2\ell+6\ell+2\log_2\ell+7$ bits
- Step 2: $\ell+1$ for $[b_i]_2^{P_1}$
- Step 3: $2(\ell+1)$ for $[e_i]_2^{P_1}$ and $[e_i]_2^{P_2}$
- Step 4: $2(\ell+1)(2+\log_2\ell)$ for $[e_i]_{N_2}^{P_1}$ and $[e_i]_{N_2}^{P_2}$
- Step 5: $2(\ell+1)(2+\log_2\ell)$ for $[v_i]_{N_2}^{P_1}$ and $[v_i]_{N_2}^{P_2}$
- Step 6: $2(\ell+1)$ for $[h_i]_2^{P_1}$ and $[h_i]_2^{P_2}$
- Step 7: $2(\ell+1)$ for $[h'_i]_2^{P_1}$ and $[h'_i]_2^{P_2}$
- Step 8: $2\ell(\ell+1)$ for $[h'_i]_N^{P_1}$ and $[h'_i]_N^{P_2}$

The result of this alteration results in a complexity of $2\ell^2+5\ell\log_2\ell+23\ell+6\log_2\ell+22$. There is some change in the groups used for shared values, but the functionality and overall procedure is changed in no substantive manner for the considerations of security and correctness. Therefore, we do not introduce detailed complexity and security discussion as we did for the primary version of our protocol. All the arguments are essentially the same.

5.2 Variation for Shared and Bit-decomposed Inputs

Similar to the previously referenced related protocol from [12], we can, with slight alteration, accept values which already exist in a shared and bitwise decomposed state as inputs to our protocol rather than only privately held values from two parties. This can allow arbitrarily many parties to be shareholders and still perform the comparison with minimal additional communication cost. Additive secret sharing is required due to its ability to recombine incomplete sets of shares into a share which is correct, and uniformly random, for a set of fewer parties. Recall that under additive secret sharing, shares of a value x are constructed, for m parties and modulus N, according to the following equation:

$$[x]_N^{P_m} = x - [x]_N^{P_1} - \cdots - [x]_N^{P_{m-1}}$$

where all $[x]_N^{P_1}$ through $[x]_N^{P_{m-1}}$ are randomly selected elements of $\mathbb{Z}_N$ and $[x]_N^{P_m}$ satisfies the equation. Once these shares are distributed, all m parties hold an equal share of the secret, and all m are required to reconstruct the secret. If, however, P_3 through P_m were to send their shares to P_2, then P_2 could sum them with its own share and hold a new share which would allow for P_2 and P_1 to compute functions without interaction from the rest of the parties. At the end of the desired functionality, shares can be redistributed to the remaining parties in a manner similar, though inverse, to the process by which they were combined. The share held by P_2 has gained no information since it is still uniform random, and both security and correctness have been maintained. It is this property which additive secret sharing possesses, and Shamir's scheme does not, that we exploit to maintain efficiency of our scheme in this variation. The steps, 1-5 and 11, in need of alterations are given in Algorithm 4. The messages exchanged in this version of the protocol are as follows:

- Step 1: $4(\ell+1)+1+(\ell+1)(2+\log_2 \ell)+1+\log_2 \ell+2\ell(m-2)$ for q_i, r_i, r'_i, r'', τ_i, π, and $[a'_i]_2^{P_1}$, as well as $[a_i]_2^{P_k}$ and $[b_i]_2^{P_k}$, where $k = 3 \dots m$
- Step 2: $\ell+1$ for $[a_i]_2^{P_2}$
- Step 3: $2(\ell+1)$ for $[e_i]_2^{P_1}$ and $[e_i]_2^{P_2}$
- Step 4: $4(\ell+1)(2+\log_2 \ell)$ for $[e_i]_{N_2}^{P_1}, [e_i]_{N_2}^{P_2}, [a'_i]_{N_2}^{P_1}$, and $[a'_i]_{N_2}^{P_2}$
- Step 5: $2(\ell+1)(2+\log_2 \ell)$ for $[v_i]_{N_2}^{P_1}$ and $[v_i]_{N_2}^{P_2}$
- Step 6: $2(\ell+1)$ for $[h_i]_2^{P_1}$ and $[h_i]_2^{P_2}$
- Step 7: $2(\ell+1)$ for $[h'_i]_2^{P_1}$ and $[h'_i]_2^{P_2}$
- Step 8: $2(\ell+1)(2+\log_2 \ell)$ for $[h'_i]_{N_2}^{P_1}$ and $[h'_i]_{N_2}^{P_2}$
- Step 9: $2(2+\log_2 \ell)$ for $[f]_{N_2}^{P_1}$ and $[f]_{N_2}^{P_2}$
- Step 10: 2ℓ for $[f]_N^{P_1}$ and $[f]_N^{P_2}$
- Step 11: $2\ell(m-2)$ for $[f]_N^{P_{j,k}}$, where $k = 3 \dots m$ and $j = 1, 2$

This leads directly to a total communication cost of $9\ell \log_2 \ell + 4m\ell + 25\ell + 12 \log_2 \ell + 35$ bits. Even though we are operating on already bit decomposed and shared values, our protocol is still very efficient in comparison to others due to the fact that we share our bit decomposed values in $\mathbb{Z}_2$ as opposed to $\mathbb{Z}_N$. This boosts efficiency through immediate savings in communication, but it is also the means by which we can compute `xor` locally without requiring secure multi-party multiplications. In the following we present Table 4, which clearly demonstrates that not only is our proposed protocol of lower asymptotic complexity in both m and ℓ, and lower complexity in terms of required overall rounds, but it is also lower in the coefficients of the dominating terms, thus presenting a substantial reduction in required communication costs.

Table 4. Overall round and communication complexity for secure comparison protocols

Presented in	Type	Overall Rounds	Bits transmitted overall
[12]	A	44	$m(m-1)(184\ell^2 \log_2 \ell + 209\ell^2)$
[13]	A	15	$m(m-1)(279\ell^2 + 5\ell)$
[7]	A	10	$m(m-1)(153\ell^2 + 432\ell \log_2 \ell + 24\ell)$
[7]	R	8	$m(m-1)(27\ell^2 + 36\ell \log_2 \ell + 5\ell)$
[9]	R	9	$m(m-1)(10(\ell(\ell+\kappa)) + 3\ell)$
This paper Section 5.2	A	5	$9\ell \log_2 \ell + 4m\ell + 25\ell + 12 \log_2 \ell + 35$

Algorithm 4: Replacement steps for shared, bit-decomposed inputs for $\text{SC}(\langle P_1, [a_i]_2^{P_1}, [b_i]_2^{P_1} \rangle \dots \langle P_m, [a_i]_2^{P_m}, [b_i]_2^{P_m} \rangle) \rightarrow (\langle P_1, [f]_N^{P_1} \rangle \dots \langle P_m, [f]_N^{P_m} \rangle)$

Input: Public info: N and N_2, $N > N_2$, N is an integer and the modulus of the secret sharing scheme, ℓ is the required bitwidth for the domain of a and b which are bitwise shared among the m parties, $N > 2^\ell$. N_2 is a prime such that $\lceil \log_2 \ell \rceil + 1 < \log_2 N_2 < \lceil \log_2 \ell \rceil + 2$, $0 \leq i \leq \ell$, and $j \in \{1, 2\}$

Output: f is secretly shared between all $P_1 \dots P_m$. $f = 1$ if $a \geq b$; otherwise, $f = 0$

1 P_1

(a) $[a_{i+1}]_2^{P_1} = [a_i]_2^{P_1}$ for $i = \ell - 1, \dots, 0$
(b) $[b_{i+1}]_2^{P_1} = [b_i]_2^{P_1}$ for $i = \ell - 1, \dots, 0$
(c) $[a_0]_2^{P_1} \leftarrow 0$
(d) $[b_0]_2^{P_1} \leftarrow 0$
(e) Generate $q_i, r_i, r'_i \in_R \mathbb{Z}_2$, $r'' \in_R \mathbb{Z}_2$, $\tau_i \in_R \mathbb{Z}^*_{N_2}$, and random shift π
(f) $[a'_i]_2^{P_1} \leftarrow q_i - [a_i]_2^{P_1}$, if $q_i = 1$
(g) Send q_i, r_i, r'_i, r'', τ_i and π to P_2, and send $[a'_i]_2^{P_1}$ to P_3

P_k $(k = 3 \dots m)$

(a) Send $[a_i]_2^{P_k}$ and $[b_i]_2^{P_k}$ to P_2

2 P_2

(a) $[a_{i+1}]_2^{P_2} = \sum_{k=2}^{m} [a_i]_2^{P_k}$, for $i = \ell - 1, \dots, 0$
(b) $[b_{i+1}]_2^{P_2} = \sum_{k=2}^{m} [b_i]_2^{P_k}$, for $i = \ell - 1, \dots, 0$
(c) $[a_0]_2^{P_2} \leftarrow 1$
(d) $[b_0]_2^{P_2} \leftarrow 0$
(e) Send $[a_i]_2^{P_2}$ to P_3

3 P_j

(a) $[e_i]_2^{P_j} \leftarrow [a_i]_2^{P_j} + [b_i]_2^{P_j}$
(b) $[e_i]_2^{P_1} \leftarrow r_i - [e_i]_2^{P_1}$, if $r_i = 1$
(c) Send $[e_i]_2^{P_j}$ to P_3

4 P_3

(a) $e_i \leftarrow [e_i]_2^{P_1} + [e_i]_2^{P_2}$
(b) Generate $[e_i]_{N_2}^{P_j}$
(c) $a'_i \leftarrow [a'_i]_2^{P_1} + [a_i]_2^{P_2}$
(d) Generate $[a'_i]_{N_2}^{P_j}$
(e) Send $[e_i]_{N_2}^{P_j}$ and $[a'_i]_{N_2}^{P_j}$ to P_j

---

5 P_j

(a) $[e_i]_{N_2}^{P_1} \leftarrow r_i - [e_i]_{N_2}^{P_1}$, if $r_i = 1$
(b) $[e_i]_{N_2}^{P_2} \leftarrow -[e_i]_{N_2}^{P_2}$, if $r_i = 1$
(c) $[a_i]_{N_2}^{P_1} \leftarrow q_i - [a_i]_{N_2}^{P_1}$, if $q_i = 1$
(d) $[a_i]_{N_2}^{P_2} \leftarrow -[a_i]_{N_2}^{P_2}$, if $q_i = 1$
(e) $[s_a]_{N_2}^{P_j} \leftarrow \sum_i [a_i]_{N_2}^{P_j}$
(f) $[\gamma'_\ell]_{N_2}^{P_j} \leftarrow [e_\ell]_{N_2}^{P_j}$
(g) $[\gamma'_i]_{N_2}^{P_j} \leftarrow [\gamma'_{i+1}]_{N_2}^{P_j} + [e_i]_{N_2}^{P_j}$, for $i = \ell - 1, \ldots, 0$
(h) $[\gamma_\ell]_{N_2}^{P_j} \leftarrow [\gamma'_\ell]_{N_2}^{P_j}$
(i) $[\gamma_i]_{N_2}^{P_j} \leftarrow [\gamma'_{i+1}]_{N_2}^{P_j} + [\gamma'_i]_{N_2}^{P_j}$, for $i = \ell - 1, \ldots, 0$
(j) $[\gamma_i]_{N_2}^{P_1} \leftarrow [\gamma_i]_{N_2}^{P_1} - 1$
(k) $[u_i]_{N_2}^{P_j} \leftarrow \tau_i [\gamma_i]_{N_2}^{P_j}$
(l) $[v_i]_{N_2}^{P_j} \leftarrow \text{Shift}_\pi \left([u_i]_{N_2}^{P_j} \right)$
(m) Send $[v_i]_{N_2}^{P_j}$ to P_3

11 P_j

(a) $[f]_N^{P_1} \leftarrow r'' - [f]_N^{P_1}$, if $r'' = 1$
(b) $[f]_N^{P_2} \leftarrow -[f]_N^{P_2}$, if $r'' = 1$
(c) Generate $[f]_N^{P_{j,k}} \in_R \mathbb{Z}_N$, for $k = 3, \ldots, m$
(d) $[f]_N^{P_j} \leftarrow [f]_N^{P_j} - \sum_{k=3}^{m} [f]_N^{P_{j,k}}$
(e) Send $[f]_N^{P_{j,k}}$ to P_k, for $k = 3, \ldots, m$

P_k $(k = 3, \ldots, m)$

(a) $[f]_N^{P_k} \leftarrow [f]_N^{P_{1,k}} + [f]_N^{P_{2,k}}$

---

6 Conclusion

Our contribution in this paper amounts to a proposed protocol for secure multi-partly computation, specifically comparison, which can be implemented very efficiently in a secret sharing scheme. Using dynamic groups for share creation as well as protocol asymmetry, we have significantly reduced both the local computation as well as the communication complexity for this vital operation (multiple orders of magnitude less than other extant protocols as well as a reduction of order). It is our hope that our contributions thus far may be of immediate benefit to the academic and research communities for immediate use and efficiency improvement of many other protocols and programs dependent on a secure multi-party comparison operator.

Our future work in this area includes an extended presentation of our protocols to include security guarantees for different adversary models. We also plan to formally present the generalization of our protocol to function in the more general setting of values which have already been secretly shared rather than those privately and locally held at the initialization of the protocol, or those which have already been bit decomposed in the secret sharing scheme. Furthermore, we will investigate the necessary strategies to make our design applicable for Shamir's secret sharing scheme.

References

1. Veugen, T., Blom, F., de Hoogh, S.J., Erkin, Z.: Secure comparison protocols in the semi-honest model. IEEE Journal of Selected Topics in Signal Processing **9**(7) (2015) 1217–1228
2. Yao, A.C.C.: Protocols for secure computations. In: Foundations of Computer Science. Volume 82., IEEE (1982) 160–164
3. Franklin, M.K., Reiter, M.K.: Secure auction systems (April 25 2000) US Patent 6,055,518.
4. Hamlen, K., Kantarcioglu, M., Khan, L., Thuraisingham, B.: Security issues for cloud computing. Optimizing information security and advancing privacy assurance: new technologies **150** (2012)
5. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the ACM April 18-20, 1967, spring joint computer conference, ACM (1967) 483–485
6. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, ACM (1989) 201–209
7. Reistad, T.I., Toft, T.: Secret sharing comparison by transformation and rotation. In: International Conference on Information Theoretic Security, Springer (2007) 169–180
8. Bogdanov, D.: How to securely perform computations on secret-shared data. Mater's Thesis (2007)
9. Reistad, T., Toft, T.: Linear, constant-rounds bit-decomposition. In: International Conference on Information Security and Cryptology, Springer (2009) 245–257
10. Toft, T.: Sub-linear, secure comparison with two non-colluding parties. In: International Workshop on Public Key Cryptography, Springer (2011) 174–191

11. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: International Colloquium on Automata, Languages, and Programming, Springer (2013) 645–656
12. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Theory of Cryptography Conference, Springer (2006) 285–304
13. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: International Workshop on Public Key Cryptography, Springer (2007) 343–360
14. Yao, A.C.C.: How to generate and exchange secrets. In: Foundations of Computer Science, IEEE (1986) 162–167
15. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al.: Fairplay-secure two-party computation system. In: USENIX Security Symposium. Volume 4., San Diego, CA, USA (2004)
16. Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: a system for secure multiparty computation. In: Proceedings of the 15th ACM conference on Computer and communications security, ACM (2008) 257–266
17. Henecka, W., Sadeghi, A.R., Schneider, T., Wehrenberg, I., et al.: Tasty: tool for automating secure two-party computations. In: Proceedings of the 17th ACM conference on Computer and communications security, ACM (2010) 451–462
18. Songhori, E.M., Hussain, S.U., Sadeghi, A.R., Schneider, T., Koushanfar, F.: Tinygarble: Highly compressed and scalable sequential garbled circuits. In: Security and Privacy, 2015, IEEE (2015) 411–428
19. Pullonen, P., Siim, S.: Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In Brenner, M., Christin, N., Johnson, B., Rohloff, K., eds.: Financial Cryptography and Data Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2015) 172–183
20. Blake, I.F., Kolesnikov, V.: One-round secure comparison of integers. Journal of Mathematical Cryptology **3**(1) (2009) 37–68
21. Damgård, I., Geisler, M., Kroigard, M.: Homomorphic encryption and secure comparison. International Journal of Applied Cryptography **1**(1) (2008) 22–31
22. Fischlin, M.: A cost-effective pay-per-multiplication comparison method for millionaires. Topics in Cryptology—CT-RSA 2001 (2001) 457–471
23. Kerschbaum, F., Biswas, D., de Hoogh, S.: Performance comparison of secure comparison protocols. In: Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on, IEEE (2009) 133–136
24. Barker, E.: Recommendation for key management—part 1: General (revision 4). http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf (2016)
25. Reistad, T.I.: Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values (2009)
26. Crépeau, C., Savvides, G., Schaffner, C., Wullschleger, J.: Information-theoretic conditions for two-party secure function evaluation. In: EUROCRYPT, Springer (2006) 538–554
27. Goldreich, O.: Foundations of cryptography: volume 1, basic tools. Cambridge university press (2001)
28. Chebyshev, P.L.: Mémoire sur les nombres premiers. J. Math. Pures Appl. **17** (1852) 366–390
29. Ramanujan, S.: A proof of bertrand's postulate. Journal of the Indian Mathematical Society **11**(181-182) (1919) 27

30. Erdos, P.: Beweis eines satzes von tschebyschef. Acta Scientifica Mathematica **5** (1932) 194–198
31. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM (1988) 1–10