

Secret Sharing Comparison by Transformation and Rotation

Tord Ingolf Reistad¹ and Tomas Toft^{2*}

¹ NTNU, Dept. of Telematics,
N-7491 Trondheim, Norway.
tordr@item.ntnu.no

² University of Aarhus, Dept. of Computer Science,
DK-8200 Aarhus N, Denmark.
tomas@daimi.au.dk

Abstract. Given any linear secret sharing scheme with a multiplication protocol, we show that a given set of players holding shares of two values $a, b \in \mathbb{Z}_p$ for some prime p , it is possible to compute a sharing *result* such that $result = (a < b)$ with only two rounds and 5ℓ invocations of the multiplication protocol online, where $\ell = \log(p)$.

Moreover, most of the work required is independent of $[a]$ and $[b]$ and may be performed in advance in a pre-processing phase before the inputs become available. This is important for practical implementations of multiparty computations, where one can have a set-up phase. The proposed protocol is also an improvement in that respect as it uses fewer rounds and less invocations of the multiplication protocol than previous solutions.

The protocol is unconditionally secure against active/adaptive adversaries and when the underlying secret sharing scheme has these properties.

1 Introduction

In multiparty computation (MPC) a number of parties P_1, \dots, P_n have *private* inputs for some function that they wish to evaluate. However, they are mutually mistrusting and do not wish to share their inputs with anyone. A great deal of work has been done on unconditionally secure constant round MPC with honest majority including but not limited to [1–4], indeed it has been demonstrated that any function can be computed securely using circuit based protocols.

However, when considering concrete applications, it is often more efficient to focus on secure arithmetic e.g. in the field \mathbb{Z}_p for some odd prime p , which may then be used to simulate integer arithmetic. Unfortunately, many such applications require non-arithmetic operations as well; this provides motivation for constructing specialized, efficient, constant-rounds protocols for primitive tasks (from the application designers view).

* Supported by Simap

Access to the binary representation of values allows many operations to be performed relatively cheaply. Although constant-round bit-decomposition is possible as demonstrated by Damgård et al. [5], it is less efficient than desired. Primitives may be used repeatedly and any improvement in the primitives will lead to a similar improvement in an overall application. This work focuses on one such primitive, namely comparison (less-than-testing) of secret shared values, i.e. obtaining a sharing of a bit stating whether one value is larger than another without leaking any information.

Related work and contribution Much research has focused on secure comparison in various settings as it is a quite useful primitive, though focus is often on some concrete application e.g. auctions. Often the setting differs from the present [6–8]. Through bit-decomposition, Damgård et al. provided the first constant rounds comparison in the present setting [5] – this required $\mathcal{O}(\ell \log(\ell))$ secure multiplications where $\ell = \log(p)$. Comparison was later improved by Nishide and Ohta [9] who reduced the complexity to $\mathcal{O}(\ell)$ multiplications.

A common streak in all of the above solutions is that the binary representation of the values is considered. Thus, unless a radically different approach is taken, improving on the $\mathcal{O}(\ell)$ bound does not seem feasible. The present work builds on sub-protocols and ideas of [5] and [9] and aims at reducing the constants hidden under big- \mathcal{O} , particularly with regard to online round complexity, i.e. when pre-processing independent of the actual values is excluded. Using ideas from [10] which considers comparison of bitwise stored values based on homomorphic encryption in a two-party setting, we construct a protocol which is extremely round efficient.

Table 1 compares the present solution to those of Damgård et al. and Nishide and Ohta. Type A refers to comparison of arbitrary values $[a], [b] \in \mathbb{Z}_p$, while R denotes values of restricted size, $[a], [b] < \lfloor \frac{p}{4} \rfloor$. When using \mathbb{Z}_p to simulate integer computation, it is not unreasonable to choose p a few bits larger to accommodate this assumption.

With regard to the restricted comparison, in contrast to earlier papers (and the general comparison), it is assumed that two rather than four attempts are needed to generate random bitwise shared values. The probability of failing is less than $1/2$, thus retrying yields a result in expected two attempts (implying an *expected* constant rounds protocol), whereas previous work considered multiple attempts in parallel in order to avoid rerunning on failed attempts (i.e. truly constant round but with negligible probability of not obtaining a result). This issue occurs when p may be arbitrary, if it is chosen sensibly, e.g. as a Mersenne prime, the failure probability may be reduced to negligible in the bit-length of p .

The structure of this article Sections 2 and 3 introduce the setting as well as a number of primitives required. Most of these are well-known, and are included in order to provide detailed analysis of our protocol. Section 4 takes a high-level view of the computation required; it reduces the problem of comparison to a more manageable form similar to previous work. Section 5 introduces the

Table 1. Complexities of comparison protocols

Presented in	Type	Rounds		Multiplications	
		overall	online	overall	online
[5]	A	44	37	$209\ell + 184\ell \log_2(\ell)$	$21\ell + 56\ell \log_2(\ell)$
[9]	A	15	8	$279\ell + 5$	$15\ell + 5$
This paper	A	12	4	$84\ell + 78 \log_2(\ell) + 17$	$18\ell + 5$
[5]	R	44	37	$171\ell + 184\ell \log_2(\ell)$	$21\ell + 56\ell \log_2(\ell)$
[9]	R	13	6	$55\ell + 5$	$5\ell + 1$
This paper	R	10	2	$23\ell + 26 \log_2(\ell) + 4$	5ℓ
This paper	R	8	2	$26\ell + 36 \log_2(\ell) + 6$	5ℓ

DGK comparison protocol. Section 6 shows how the DGK algorithm can be used to create random bitwise shared elements. Sections 7, 8 and 9 modifies DGK to avoid leaking information. Finally Sect. 10 gives an overall analysis and conclusion.

Acknowledgments We would like to thank Jesper Buus Nielsen helping us come up with the idea of hiding which way the comparison is done, instead of hiding the result. We are also grateful to the anonymous reviewers for their helpful comments and Prof. Ivan Damgård and Prof. Stig Frode Mjøelsnes for their support.

2 Preliminaries

We assume a linear secret sharing scheme with a multiplication protocol allowing values of the prime field \mathbb{Z}_p , $\ell = \lceil \log(p) \rceil$, to be shared among n parties. As an example, consider Shamir's scheme along with the protocols of Ben-Or et al. (or the improved protocols of Gennaro et al.) [11–13]. The properties of the scheme are inherited, i.e. if this is unconditionally secure against active/adaptive adversaries then so are the protocols proposed. In addition to sharing values and performing secure arithmetic of \mathbb{Z}_p , the parties may reveal (reconstruct) shared values, doing this ensures that the value becomes known by *all* parties.

We use $[a]$ to denote a secret sharing of $a \in \mathbb{Z}_p$. Secure computation is written using an infix notation. For shared values $[a]$ and $[b]$, and constant $c \in \mathbb{Z}_p$, computation of sums will be written as $[a] + c$ and $[a] + [b]$, while products will be written $c[a]$ and $[a][b]$. The initial three follow by the linearity of the scheme, while the fourth represents an invocation of the multiplication protocol.

Sharings of bits, $[b] \in \{0, 1\} \subset \mathbb{Z}_p$ will also be considered. Boolean arithmetic is written using infix notation, though it must be realized using field arithmetic. Notably xor of two bits is constructed as $[b_1] \oplus [b_2] = [b_1] + [b_2] - 2[b_1][b_2]$ which is equivalent.

Values may also be *bitwise shared*, written $[a]_B$. Rather than having a sharing of a value itself, $[a]$, sharings of the bits of the binary representation of a are

given, i.e. $[a_0], \dots, [a_{\ell-1}] \in \{0, 1\}$ such that

$$[a] = \sum_{i=0}^{\ell-1} 2^i [a_i]$$

for $\ell = \lceil \log(p) \rceil$, with the sum being viewed as occurring over the integers. Note that $[a]$ is easily obtained from $[a]_B$ by the linearity of the scheme.

When considering complexity, the focus will be on communication. Computation will be disregarded in the sense that polynomial time suffices. Similar to other work, focus will be placed on the number of invocations of the multiplication protocol as this is considered the most costly of the primitives. Addition and multiplication by constants require no interaction and is considered costless. The complexity of sharing and revealing is seen as negligible compared to that of multiplication and is ignored.

It is assumed that invocations of the multiplication protocol parallelize arbitrarily – multiplications are executed in parallel when possible. Round complexity is formally rounds-of-multiplications; rounds for reconstruction are disregarded as in other work.

3 Simple Primitives

This section introduces a number of simple primitives required below. Most of these sub-protocols are given in [5] but are repeated here in order to provide a detailed analysis as well as for completeness. Most of these are related to the generation of random values unknown to all parties. It is important to note that these may fail, however, this does not compromise the privacy of the inputs – failure simply refers to the inability to generate a proper random value (which is detected). Generally the probability of failure will be of the order $1/p$, which for simplicity will be considered negligible, see [5] for further discussion.

Random element generation A sharing of a uniformly random, unknown value $[r]$ may be generated by letting all parties share a uniformly random value. The sum of these is uniformly random and unknown to all, even in the face of an active adversary. The complexity of this is assumed to be equivalent to an invocation of the multiplication protocol.

Random non-zero values A uniformly random, non-zero value may be obtained by generating two random values, $[r]$ and $[s]$ and revealing the product. If $rs = 0$ the protocol fails, however if not, $[r]$ is guaranteed non-zero and unknown as it is masked by $[s]$. The complexity is three multiplications in two rounds.

Random bits The parties may create an uniformly random bit $[b] \in \{0, 1\}$. As described in [5] the parties may generate a random value $[r]$ and reveal its square, r^2 . If this is 0 the protocol fails, otherwise they compute $[b] = 2^{-1}((\sqrt{r^2})^{-1})[r] + 1$ where $\sqrt{r^2}$ is defined such that $0 \leq \sqrt{r^2} \leq \frac{p-1}{2}$. The complexity is two multiplications in two rounds.

Unbounded Fan-In Multiplications It is possible to compute prefix-products of arbitrarily many non-zero values in constant rounds using the method of Bar-Ilan and Beaver [1]. Given $[a_1], \dots, [a_k] \in \mathbb{Z}_p^*$, $[a_{1,i}] = [\prod_{j=1}^i a_j]$ may be computed for $i \in \{1, 2, \dots, k\}$ as follows.

Let $r_0 = 1$ and generate k random non-zero values $[r_1], \dots, [r_k]$ as described above, however, in parallel with the multiplication of $[r_j]$ and the mask $[s_j]$, compute $[r_{j-1}s_j]$ as well. Then $[r_{j-1}r_j^{-1}]$ may be computed at no cost as $[r_{j-1}s_j] \cdot (r_j s_j)^{-1}$. Each $[a_j]$ is then multiplied onto $[r_{j-1}r_j^{-1}]$ and all these are revealed. We then have

$$[a_{1,i}] = \prod_{j=1}^i (a_j r_{j-1} r_j^{-1}) \cdot [r_j] .$$

Privacy of the $[a_j]$ is ensured as they are masked by the $[r_j^{-1}]$, and complexity is $5k$ multiplications in three rounds. Regarding the preparation of the masking values as pre-processing, the online complexity is k multiplications in a single round.

Constructing a unary counter Given a secret shared number $[a] < m < p - 1$ for some known m , we must construct a vector of shared values $[V]$ of size m containing all zeros except for the a 'th entry which should be 1. This may be constructed similarly to the evaluation of symmetric Boolean functions of [5].

Let $[A] = [a + 1]$ and for $i \in \{0, 1, \dots, m - 1\}$ consider the set of functions: $f_i : \{1, 2, \dots, m + 1\} \rightarrow \{0, 1\}$ mapping everything to 0 except for $i + 1$ which is mapped to 1. The goal is to evaluate these on input A . By Lagrange interpolation, each may be replaced by a *public* m -degree polynomial over \mathbb{Z}_p . Using a prefix-product, $[A], \dots, [A^m]$ may be computed; once this is done, the computation of the entries of $[V]$ is costless. Thus, complexity is equivalent to prefix-product of m terms.

4 A High-level View of Comparison

The overall idea behind the proposed protocol is similar to that of other comparison protocols, including that of [9]. The problem of comparing two secret shared numbers is first transformed to a comparison between shared values where sharings of the binary representations are present; this greatly simplifies the problem. The main difference between here and previous solutions is that we consider inputs of marginally bounded size: $[a], [b] < \lfloor \frac{p-1}{4} \rfloor$. This assumption reduces complexity but may be dropped as in previous work; the section concludes with a sketch of the required alterations. Once the problem is transformed to comparison of bitwise represented values, we may apply the novel secure computation of the following sections.

Our goal is to compute a bit $[result] = ([a] < [b])$. The transformation of the problem proceeds as follows. First we consider the comparison of $[a'] = 2[a] < \frac{p-1}{2}$ and $[b'] = 2[b] + 1 < \frac{p-1}{2}$ rather than $[a]$ and $[b]$. This does not alter the

result, however, it ensures that the compared values are not equal, which will be required below.

As $[a'], [b'] < \frac{p-1}{2}$, comparing them is equivalent to determining whether $[z] = [a'] - [b'] \in \{1, 2, \dots, (p-1)/2\}$. In turn this is equivalent to determining the least significant bit of $2[z]$, written $[(2z)_0]$. If $[z] < (p-1)/2$, then $2[z] \in \mathbb{Z}_p$ is even; if not then a reduction modulo p occurs and it is odd.

The computation of $[(2z)_0]$ is done by first computing and revealing $[c] = 2[z] + [r]_B$, where $[r]_B$ is uniformly random. Noting that $[(2z)_0]$ depends only on the least significant bits of c and $[r]_B$ (c_0 and $[r_0]$) and whether an overflow (a modulo p reduction) has occurred in the computation of c . The final result is simply the xor of these three Boolean values.

Determining overflow is equivalent to computing $c < [r]_B$. The remainder of the article describes this computation. Privacy is immediate and follows from the security of the secret sharing scheme. Seeing c does not leak information, as $[r]_B$ is uniformly random then so is c . The full complexity analysis is postponed until Sect. 10.

Unbounded $[a]$ and $[b]$ If it is not ensured that $[a], [b] < \lfloor \frac{p-1}{4} \rfloor$ then further work must be performed in order to perform a comparison. Computing bits $[t_{=}] = [a = b]$ (e.g. as described in [9]), as well as $[t_a] = [a < (p-1)/2]$, $[t_b] = [b < (p-1)/2]$, and $[t_\delta] = [a - b < (p-1)/2]$ allows the final result to be determined by a small Boolean circuit which translates to a simple secure computation over \mathbb{Z}_p . The result is either immediate (if $[a] = [b]$) or may be determined by the three latter bits, see [9] for details.

5 The DGK Comparison Protocol

In [10] Damgård et al. proposed a two-party protocol for comparison based on a homomorphic encryption scheme. They consider a comparison between a publicly known and a bitwise encrypted value with the output becoming known. Though their setting is quite different, ideas may be applied in the present. Initially, we consider comparison of two unknown, ℓ -bit, bitwise shared values $[a]_B$ and $[b]_B$, determining $[a] < [b]$ is done by considering a vector $[E] = [e_{\ell-1}], \dots, [e_0]$ of length ℓ with

$$[e_i] = [s_i] \left(1 + [a_i] - [b_i] + \sum_{j=i+1}^{\ell-1} ([a_j] \oplus [b_j]) \right) \quad (1)$$

where the $[s_i]$ are uniformly random non-zero values.

Theorem 1 (DGK). *Given two bitwise secret shared values $[a]_B$ and $[b]_B$, the vector $[E]$, given by equation 1 will contain uniformly random non-zero values and at most one 0. Further, a 0 occurs exactly when $[a] < [b]$.*

Proof. The larger of two values is characterized by having a 1 at the most significant bit where the numbers differ. Letting m denote the most significant differing bit-position, the intuition behind the $[e_i]$ is that $[e_{\ell-1}], \dots, [e_{m+1}]$ will equal their respective $[s_i]$, and $[e_{m-1}], \dots, [e_0]$ will also be uniformly random and non-zero, as for $i < m$ $1 \leq \sum_{j=i+1}^{\ell-1} ([a_j] \oplus [b_j]) \leq \ell$. Finally, $[e_m]$ will then be 0 exactly when $[b_m]$ is 1, i.e. when $[b]$ is larger than $[a]$, otherwise it is random as well. \square

Though the non-zero elements of vector $[E]$ are random, the location of the zero leaks information. In [10] this is solved by the two-party nature of the protocol – one party knows the decryption key, while the other permutes the $[e_i]$. The multi-party setting here requires this to be performed using arithmetic.

6 Creating Random Bitwise Shared Values

Based on the comparison of Sect. 5, a random, bitwise shared value $[r]_B$ may be generated such that it is less than some k -bit bound m . This is accomplished by generating the k bits $[r_{k-1}], \dots, [r_0]$ and verifying that this represents a value less than m . Noting that information is only leaked when a 0 occurs in the $[e_i]$, ensuring that this coincides with the discarding of $[r]_B$ if it is too large suffices. Based on the above, the k $[e_i]$ are computed as

$$[e_i] = [s_i] \left(1 + (m-1)_i - [r_i] + \sum_{j=i+1}^{k-1} ((m-1)_j \oplus [r_j]) \right) \quad (2)$$

where $(m-1)_i$ denotes the i 'th bit of $m-1$. By Theorem 1 this will contain a 0 exactly when $m-1 < [r]_B \Leftrightarrow [r]_B \geq m$.

The complexity of generating random bitwise shared values consists of the generation of the k bits and the k masks, $[s_{k-1}], \dots, [s_0]$, plus the k multiplications needed to perform the masking. Overall this amounts to $4k$ multiplications in three rounds as the $[r_i]$ and $[s_i]$ may be generated in parallel (and the $[s_i]$ are not verified as being non-zero, which increases the probability of failure marginally). The probability of success depends heavily on m , though it is at least $1/2$. Note that this may be used to generate uniformly random elements of \mathbb{Z}_p .

7 Avoiding Information Leaks

Though Sect. 4 has reduced the problem of general comparison to that of comparing a public value to a bitwise shared one, $c < [r]_B$, two problems remain with regard to the DGK-protocol in the present setting. First off, the result is stored as the existence of a 0 in a vector $[E]$ of otherwise uniformly random non-zero values, rather than as a single shared bit $[result] \in \{0, 1\}$ which is required to conclude the computation. Second, the position of the 0 in $[E]$ (if it exists) leaks information.

Hiding the result The former of the two problems may be solved by “masking the result” thereby allowing it to be revealed. Rather than attempting to hide the result, the comparison will be hidden. Formally, a uniformly random value $[s] \in \{-1, 1\}$ is generated, this is equivalent to generating a random bit. The $[e_i]$ of equation 1 are then replaced by

$$[e_i] = [s_i] \left(1 + (c_i - [r_i])[s] + \sum_{j=i+1}^{\ell-1} (c_j \oplus [r_j]) \right) .$$

As $[z]$ is non-zero, we have $c \neq [r]_B$, and thus the desired result is the one obtained, xor’ed with $-2^{-1}([s] - 1)$. If the comparison was flipped ($[s] = -1$) so is the output. Note that the observation made here implies that when considering comparison, it suffices to consider public results: Shared results are immediately available by randomly swapping the inputs.

Hiding the location of the 0 In order to hide the location of the 0, the $[e_i]$ must be permuted, however, as the non-zero entries are uniformly random, it suffices to *shift* $[E]$ by a unknown, random amount $v \in \{0, 1, \dots, \ell - 1\}$. I.e. constructing a vector $[\tilde{E}]$ with $[\tilde{e}_i] = [e_{(i+v) \bmod \ell}]$.

Shifting an ℓ -term vector by some unknown amount $[v]$ may be done by encoding $[v]$ in a *shifting-vector* $[W]$ of length ℓ with entries

$$[w_i] = \begin{cases} 0 & i \neq [v] \\ 1 & i = [v] \end{cases}$$

Note that this is *exactly* the unary counter of Sect. 3. Given this, an entry of the shifted vector may be computed as:

$$[\tilde{e}_i] = [e_{(i+v) \bmod \ell}] = \sum_{j=0}^{\ell-1} [e_j] \cdot [w_{(j+i) \bmod \ell}] , \quad (3)$$

however, this implies quadratic complexity overall. Thus, rather than shifting the $[e_i]$, the $[\tilde{e}_i]$ will be computed based on already shifted values, i.e. we will consider $[\tilde{r}_i] = [r_{(i+v) \bmod \ell}]$ and $[\tilde{c}_i] = c_{(i+[v]) \bmod \ell}$ along with shifted sums of xor’s (the masks $[s_i]$ need not be shifted). This leaves us with two distinct problems:

1. Obtain the shifted bits of $[r]_B$ and c .
2. Obtain the shifted sums of the xor’ed bits

The solutions to these problems are described in the following sections.

8 Shifting Bits

The first thing to note is that securely shifting *known* values is costless, though naturally the result is shared. This is immediate from equation 3, thus the shifted bits of c , $[\tilde{c}_{\ell-1}], \dots, [\tilde{c}_0]$, are available at no cost.

Regarding $[r]$, shifting its bits is as difficult as shifting the $[e_i]$. Thus an alternative strategy must be employed, namely generating “shifted” bits and constructing the *unshifted* $[r]$ from these. Let $[v]$ denote the shifting-value and $[W]$ the shifting vector, and compute an ℓ -entry bit-vector $[K]$ as

$$[k_i] = 1 - \sum_{j=0}^i [w_j] \quad (4)$$

at no cost. Clearly $[k_i] = 1$ for $i < [v]$ and $[k_i] = 0$ for $i \geq [v]$. $[r]$ may then be computed based on random “shifted” bits as

$$[r] = [2^{-v}] \cdot \left(\sum_{i=0}^{\ell-1} 2^i [\tilde{r}_i] (1 + [k_i](2^\ell - 1)) \right),$$

where $[2^{-v}]$ is computed by the costless $\sum_{i=0}^{\ell-1} 2^{-i} [w_i]$. It can be verified that the above is correct, assuming that the bits represent a value in \mathbb{Z}_p . $[2^{-v}]$ performs the “unshifting,” while the final parenthesis handles “underflow.”

It remains to verify that the bits represent a shifted value of \mathbb{Z}_p . This may be done by performing the exact same computation as needed for the full comparison, i.e. shifting the bits of $p-1$, computing the shifted sums of xors and masking these. The shifting value $[v]$ is “hard-coded” into $[r]$, but it may be “reused” for this, as no information is leaked when no 0 occurs, i.e. when $[r] < p$. When information is leaked, the value is discarded anyway.

As generating $[r]$ and the shifted bits $[\tilde{r}_i]$ involves creating a uniformly random value $[v] \in \{0, 1, \dots, \ell-1\}$ as described in Sect. 6. This is 3 rounds and $8 \log_2(\ell)$ multiplications as we perform two attempts. $[W]$ may be computed from $[v]$ in an additional round and $5 \log_2(\ell)$ multiplications as pre-processing parallelizes. The shifted bits of $[r]$ and the masks $[s_i]$ may be generated concurrently using 3ℓ multiplications. These masks will not be ensured non-zero as this is unlikely to occur, this results in a slightly larger probability of aborting. Checking $[r] < p$ adds three rounds and 5ℓ multiplications, providing a total of 7 rounds, $8\ell + 13 \log_2(\ell)$ multiplications. The protocol fails with probability less than $1/2$, thus two attempts are performed in parallel giving a total of 7 rounds and $16\ell + 26 \log_2(\ell)$ multiplications.

Computing $[r]$ based on the shifted bits uses an additional round and ℓ multiplications. Concurrently, a sharing of the least significant bit of $[r]$ may be obtained using ℓ multiplications as described by equation 3. This value is needed for the overall computation.

9 Shifting the Sums of Xor’s

It remains to describe how to “shift” the sums of the xor’s, $\sum_{j=i+1}^{\ell-1} [r_j] \oplus c_j$. The shifted xor’s $[\tilde{c}_i \oplus \tilde{r}_i]$ may be computed directly based on the previous section. The sum, however, consists of the terms from the *current* index to the *shifted* position

of the most significant bit, which may or may not “wrap around.” Focusing on a single $[\tilde{c}_i]$, we consider two cases: $i < [v]$ and $i \geq [v]$. Naturally, $[v]$ is unknown, however, the correct case is determined by $[k_i]$ of above.

When $i < [v]$ the terms have been shifted more than i , thus we must sum from $i + 1$ to $[v] - 1$, this equals:

$$[\sigma_i^<] = \sum_{j=i+1}^{\ell-1} [k_j] \cdot ([\tilde{r}_j \oplus \tilde{c}_j]) .$$

Multiplying by the $[k_j]$ ensures that only the relevant terms are included. For $i \geq [v]$ the terms are shifted less than i , thus the most significant bit does not pass the i 'th entry. The sum must therefore be computed from i to $\ell - 1$ and from 0 to $[v] - 1$. Viewing this as the sum of all terms *minus* the sum from $[v]$ to i we get

$$[\sigma_i^>] = \left(\sum_{j=0}^{\ell-1} [\tilde{r}_j \oplus \tilde{c}_j] \right) - \left(\sum_{j=0}^i (1 - [k_j]) \cdot [\tilde{r}_j \oplus \tilde{c}_j] \right) .$$

Thus, as the computation of the two candidates $[\sigma_i^<]$ and $[\sigma_i^>]$ for each position i is linear, computing all sums simply require selecting the correct one for each entry based on $[k_i]$:

$$[k_i][\sigma_i^<] + (1 - [k_i])[\sigma_i^>] .$$

This requires 2 rounds and 4ℓ multiplications, however, noting that

$$[k_j] \cdot ([\tilde{r}_j] \oplus [\tilde{c}_j]) = [k_j \tilde{r}_j] + [k_j][\tilde{c}_j] - 2[k_j \tilde{r}_j][\tilde{c}_j] ,$$

the $[k_j \tilde{r}_j]$ can be pre-processed (in 1 round, ℓ multiplications). This reduces the online complexity of calculating the sum to 1 round and 4ℓ multiplications.

10 Overall Analysis and Conclusion

Pre-processing Initially a uniformly random value $[v] \in \{0, 1, \dots, \ell - 1\}$ must be created. This value is used to construct a random shifting vector $[W]$ of length ℓ . Concurrently the *shifted* bits of $[r]$ are generated. It must then be verified that $[r] < p$. Once this is done the values $[r]$ and $[r_0]$ may be computed. In addition, the non-zero masks $[s_i]$, the sign bit $[s]$ must be generated, and $[s_i \cdot s]$, $[s_i \cdot s \cdot \tilde{r}_i]$ and $[s_i \cdot k_i]$ and $[s \cdot r_0]$ computed.

The overall cost of creating the $[v]$, $[W]$ and shifted bits $[\tilde{r}_i]$ is 7 rounds, $16\ell + 26 \log_2(\ell)$ multiplications. In addition, creating $[r]$, $[r_0]$, $[s_i]$, $[s]$, $[s_i \cdot s]$, $[s_i \cdot s \cdot \tilde{r}_i]$, and $[s_i \cdot k_i]$ adds another round and $7\ell + 4$ multiplications (as $[k_i \tilde{r}_i]$ is already computed). The total pre-processing complexity is therefore 8 rounds $23\ell + 26 \log_2(\ell) + 4$ multiplications.

Round complexity may be further reduced. $[v]$ and $[W]$ may be created in parallel, also $[r]$ and $[r_0]$ may be computed before $[r] < p$ is verified. The complexity of pre-processing is thereby reduced to 6 rounds, though $26\ell + 36 \log_2(\ell) + 6$ multiplications are required.

Online The computation of $c = [r] + 2(2[a] - (2[b] + 1))$ where $[a], [b] < \lfloor \frac{\ell}{4} \rfloor$ is costless. The online computation of the $[\tilde{e}_i]$ is

$$[\tilde{e}_i] = [s_i] \left(1 + ([\tilde{c}_i] - [\tilde{r}_i]) [s] + \left([k_i][\sigma_i^{\geq}] + (1 - [k_i]) [\sigma_i^{\leq}] \right) \right) ,$$

however, this may be rephrased to reduce complexity:

$$[s_i] + [s_i \cdot s][\tilde{c}_i] - [s_i \cdot s \cdot \tilde{r}_i] + \left(([s_i \cdot k_i][\sigma_i^{\geq}] + ([s_i] - [s_i \cdot k_i]) [\sigma_i^{\leq}]) \right) . \quad (5)$$

This implies two rounds and 5ℓ multiplications: First $[\sigma_i^{\leq}]$ and $[\sigma_i^{\geq}]$ are determined using 2ℓ multiplications, then three parallel multiplications are performed for each of the ℓ instances of equation 5. Letting e denote if a 0 was encountered in the $[\tilde{e}_i]$, the final result is

$$[result] = [r_0] \oplus c_0 \oplus ((-2^{-1}([s] - 1)) \oplus e) = [r_0 \oplus (-2^{-1}([s] - 1))] \oplus c_0 \oplus e .$$

which is costless. The overall complexity is therefore $28\ell + 26 \log_2(\ell) + 4$ multiplications in 10 rounds or 8 rounds $31\ell + 36 \log_2(\ell) + 6$ if the pre-processing round complexity is optimised as described above.

References

1. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Rudnicki, P., ed.: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, New York, ACM Press (1989) 201–209
2. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st Annual Symposium on Foundations of Computer Science, Las Vegas, Nevada, USA, IEEE Computer Society Press (November 12–14, 2000) 294–304
3. Cramer, R., Damgård, I.: Secure distributed linear algebra in a constant number of rounds. In Kilian, J., ed.: Advances in Cryptology - Crypto 2001, Berlin, Springer-Verlag (2001) 119–136 Lecture Notes in Computer Science Volume 2139.
4. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Proceedings of ICALP 2002, Berlin, Springer-Verlag (2002) 244–256 Lecture Notes in Computer Science Volume 2380.
5. Damgård, I., Fitz, M., Kiltz, E., Nielsen, J., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi, S., Rabin, T., eds.: Theory of Cryptography. Volume 3876 of Lecture Notes in Computer Science (LNCS)., Springer (2006) 285–304
6. Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., Toft, T.: Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications. BRICS Report Series RS-05-18, BRICS (2005) <http://www.brics.dk/RS/05/18/>.
7. Fischlin, M.: A cost-effective pay-per-multiplication comparison method for millionaires. In Naccache, D., ed.: Topics in Cryptology – CT-RSA 2001. Volume 2020 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2001) 457–471
8. Schoenmakers, B., Tuyls, P.: Practical two-party computation based on the conditional gate. In Lee, P.J., ed.: Advances in Cryptology – ASIACRYPT 2004. Volume 3329 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2004) 119–136
9. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: PKC 2007: 10th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany (2007)
10. Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for on-line auctions. In: ACISP 07: 12th Australasian Conference on Information Security and Privacy. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany (2007) Forthcoming.
11. Shamir, A.: How to share a secret. Communications of the ACM **22**(11) (1979) 612–613
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for noncryptographic fault-tolerant distributed computations. In: 20th Annual ACM Symposium on Theory of Computing, ACM Press (1988) 1–10
13. Gennaro, R., Rabin, M., Rabin, T.: Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM Press (1998) 101–111