Secure Comparison and Interval Test Protocols based on Three-party MPC

Wataru Fujii[1], Keiichi Iwamura[1] and Masaki Inamura[1,2]
[1]*Graduate School of Engineering, Tokyo University of Science, Tokyo, Japan*
[2]*Center for Research and Collaboration, Tokyo Denki University, Tokyo, Japan*

Keywords: Secret Sharing, Multiparty Computation, Secure Comparison, Secure Interval Test.

Abstract: Multiparty Computation (MPC) is a technology that enables computations to be performed without exposing private data. Three main approaches are employed to construct an MPC: secret-sharing schemes, homomorphic encryption, and garbled circuit. Although secret-sharing based MPC involves lower communication cost generally, it requires more round communications for concrete protocols such as comparisons or interval tests. Herein, we propose a five-round secure comparison protocol and a five-round interval test protocol using a shuffling protocol based on a three-party secret-sharing scheme. Additionally, we compare our protocols with existing protocols based on rounds and multiplications.

1 INTRODUCTION

Recently, multiparty computation (MPC) has attracted significant attention as a technique to obtain statistics without exposing private data. In general, an MPC is a computation between N parties $P_1,\ldots,P_N$ having private inputs $x_1,\ldots,x_N$. These parties compute outputs $(y_1,\ldots,y_N) = f(x_1,\ldots,x_N)$ such that P_i derives only output y_i.

Three main techniques have been developed for MPC: secret-sharing schemes, homomorphic encryption, and garbled circuit. In general, secret-sharing-based MPC involves relatively lower computational and communication costs but requires more round communications for concrete protocols such as comparisons or interval tests. Therefore, it is important for secret-sharing-based MPC to reduce the round communications of these protocols. Herein, we focus on Shamir's (2,3) secret-sharing (Shamir, 1979)-based MPC, which is an algorithm that divides a secret s into three shares. The original secret s can be restored by collecting two of three shares.

Damgård et al. demonstrated the first constant-round comparison protocol based on secret-sharing schemes (Damgård et al., 2006); Nishide and Ohta proposed more efficient constant-round protocols such as comparisons or interval tests (Nishide and Ohta, 2007) by improving the idea thereof. Furthermore, Reistad and Toft proposed an eight-round comparison protocol based on the limited compared values (Reistad and Toft, 2007).

Herein, we propose a five-round secure comparison protocol based on Shamir's (2,3) secret-sharing scheme using a shuffling protocol (Laur et al., 2011; Chida et al., 2019). Moreover, we construct a five-round interval test protocol using our proposed comparison protocol and the interval test protocol of (Nishide and Ohta, 2007).

Related Works. Many studies regarding secure constant-round comparison protocols based on secret-sharing schemes have been conducted, e.g., (Damgård et al., 2006; Nishide and Ohta, 2007; Reistad and Toft, 2007; Reistad, 2009). Moreover, studies regarding logarithmic-round secure comparison protocols have been introduced (Garay et al., 2007; Catrina and De Hoogh, 2010). Recently, Morita et al. proposed an efficient-round secure comparison protocol based on the client-server model, which enables communication rounds to be reduced by the precomputation of clients (Morita et al., 2018).

2 PRELIMINARIES

In this section, we introduce some notations and known techniques. Herein, we denote p to represent an odd prime number, l the bit length of p, and Z_p a prime field. In addition, we assume that all protocols are computed over Z_p.

Fujii, W., Iwamura, K. and Inamura, M.
Secure Comparison and Interval Test Protocols based on Three-party MPC.
DOI: 10.5220/0009161406980704
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 698-704
ISBN: 978-989-758-399-5; ISSN: 2184-4356

2.1 Shamir's (2,3) Secret-sharing Scheme

Let $s \in Z_p$ be the secret shared by three parties P_1, P_2, P_3. Every parity P_i has a unique identification value $x_i \in Z_p \backslash \{0\}$.

Share. A dealer builds a random polynomial $f(x)$ with a randomly chosen $a \in Z_p$, as follows:

$$f(x) = s + ax \bmod p$$

and sends a share $[s]_i = f(x_i)$ to P_i.

Reveal. A restorer collects two or more shares of s and recovers s by reconstructing the random polynomial $f(x)$.

2.2 Addition, Multiplication and XOR

We assume that three parties possess two secret shared values a, b as $[a] = ([a]_1, [a]_2, [a]_3)$, $[b] = ([b]_1, [b]_2, [b]_3)$, and $c \in Z_p$, which is a value known to all parties. $[a+b]$, $[a]+c$, and $c[a]$ can be computed locally by computing $[a]_i + [b]_i$, $c + [a]_i$, and $c[a]_i$, respectively. Therefore, these operations do not require communication.

Meanwhile, the multiplication of shared values $[a]$, $[b]$ requires the parties to communicate with each other. We assume that the parties perform the multiplication proposed by (Gennaro et al., 1998). According to (Gennaro et al., 1998), when the number of parties is n, the multiplication requires $n(n-1)$ shared values to be sent in parallel. In our settings, six shared values must be sent in parallel.

The XOR operation of two secret shared bits $[b_1]$, $[b_2] \in \{0,1\} \subset Z_p$ is constructed as $[b_1] \oplus [b_2] = [b_1] + [b_2] - 2[b_1][b_2]$. $[b_1] + [b_2]$ can be performed locally and the communication costs of this operation equal those of the multiplication. When b_2 is a public value, this operation do not require communication.

2.3 Complexity

Similar to other studies, we used two metrics: round complexity and number of multiplication invocations, to evaluate the communication costs of the protocols. The round complexity means the numbers of sequential rounds of the multiplications. If we compute $[a][b]$ and $[c][d]$, two operations can be performed in parallel. Therefore, the complexity is one round and two multiplications.

We assume that the complexity of sharing and revealing is negligible compared with that of multiplication, as in other studies. However, we determine the complexity of the shuffling protocols. For more details, see Section 5.4.

3 SUBPROTOCOLS

In this section, we introduce a number of subprotocols. These protocols are related to the generation of random values.

Random Number Sharing (RNS). This protocol, described in Algorithm 1, outputs a sharing $[r]$ of a uniformly random, unknown value $r \in Z_p$ (Damgård et al., 2006). The complexity is one round and one multiplication.

Algorithm 1: RNS Protocol.

Output: $[r]$ ($r \in Z_p$).
1: P_i shares a uniformly random shared value $[r.i]$.
2: $[r] \leftarrow [r.1] + [r.2] + [r.3]$.
3: Output $[r]$.

Random Non-zero Number Sharing (RNNS). This protocol, described in Algorithm 2, outputs a sharing $[r]$ of a uniformly random, unknown non-zero value $r \in Z_p \backslash \{0\}$ (Damgård et al., 2006). The complexity is two rounds and three multiplications.

Algorithm 2: RNNS Protocol.

Output: $[r]$ ($r \in Z_p \backslash \{0\}$).
1: $[r], [s] \leftarrow 2 \times RNS$.
2: $[v] \leftarrow [r] \times [s]$
3: $v \leftarrow \text{Reveal}([v])$.
4: If $v = 0$, abort. Otherwise, output $[r]$.

Random Bit Sharing (RBS). This protocol, described in Algorithm 3, outputs a sharing $[b]$ of a uniformly random unknown bit $b \in \{0,1\}$ (Damgård et al., 2006). The complexity is two rounds and two multiplications.

Algorithm 3: RBS Protocol.

Output: $[b]$ ($b \in \{0,1\}$).
1: $[r] \leftarrow RNS$.
2: $[a] \leftarrow [r] \times [r]$
3: $a \leftarrow \text{Reveal}([a])$.
4: $a.1, a.2 \leftarrow \sqrt{a}$
5: $c \leftarrow \min(a.1, a.2)$.
6: If $c = 0$, abort. Otherwise, proceed as follows
7: $[d] \leftarrow [r]/c$. ▷ $d \in \{-1, 1\}$
8: $[b] \leftarrow ([d]+1)/2$.
9: Output $[b]$.

Random Bitwise Values Sharing (RBVS). This protocol, described in Algorithm 4, outputs bitwise sharings $[r_0],\ldots,[r_{l-1}]$ of a uniformly random, unknown value r such that $0 \leq r = \sum_{i=0}^{l-1} 2^i r_i < p$. In Algorithm 4, $(p-1)_i$ denotes the i'th bit of $p-1$, and $r < p$ occurs only when a vector $\vec{d}$ does not contain any 0 elements. We assume that two attempts are required to compute $r < p$ for an arbitrary odd p, as in (Reistad and Toft, 2007; Reistad, 2009). Therefore, the complexity is three rounds and $8l$ multiplications. When $p = 2^l - c$, where c is a small integer, only one attempt is required, and the multiplications can be reduced to $2l$. For more details, see (Reistad and Toft, 2007; Reistad, 2009).

Algorithm 4: RBVS Protocol.

Output: $[r_0],\ldots,[r_{l-1}]$, $(r_i \in \{0,1\})$.
1: For $i = 0,\ldots,l-1$ in parallel do
   $[r_i] \leftarrow RBS$
   $[s_i] \leftarrow RNS$.
2: For $i = 0,\ldots,l-1$ in parallel do
   $[d_i] = [s_i](1 + (p-1)_i - [r_i] + \sum_{j=i+1}^{l-1}((p-1)_j \oplus [r_j]))$.
3: $[\vec{d}] = ([d_0],\ldots,[d_{l-1}])$.
4: $\vec{d} \leftarrow \text{Reveal}([\vec{d}])$.
5: If a vector $\vec{d}$ contains a 0 element then abort. Otherwise, proceed as follows.
6: Output $[r_0],\ldots,[r_{l-1}]$.

4 EXISTING COMPARISON PROTOCOL

The secure comparison protocol is a protocol that computes a shared bit $[a < b]$ from shared values $[a]$,$[b] \in Z_p$. Reistad and Toft proposed an eight-round comparison protocol based on compared values that were limited to less than $\frac{p-1}{2}$ (Reistad and Toft, 2007; Reistad, 2009).

Their protocols comprise two subprotocols. The first subprotocol converts the comparison $[a < b]$ into a comparison $[r] > c$, where $[r]$ is a random bitwise-shared value, and c is a value known to all parties.

The second subprotocol computes the comparison $[r] > c$. This subprotocol is based on the homomorphic encryption comparison protocol (Damgård et al., 2007).

4.1 Conversion of the Comparison

This protocol, described in Algorithm 5, has inputs $[a]$, $[b] < \frac{p-1}{2}$ and outputs $([a] < [b]) = c_o \oplus [r_0] \oplus ([r] > c)$, where c_o is the least-significant bit of c and $[r_0]$ is the least-significant bit of $[r]$. For more details on this protocol, see (Reistad and Toft, 2007; Reistad, 2009).

Algorithm 5: Conversion Protocol.

Input: $[a]$, $[b] < (p-1)/2$.
Output: $([a] < [b]) = c_o \oplus [r_0] \oplus ([r] > c)$.
1: $([r_0],\ldots,[r_{l-1}]) \leftarrow RBVS$.
2: $[r] \leftarrow \sum_{i=0}^{l-1} 2^i [r_i]$.
3: $[c] \leftarrow 2([a] - [b]) + [r]$.
4: $c \leftarrow \text{Reveal}([c])$.
5: Output $([a] < [b]) = c_o \oplus [r_0] \oplus ([r] > c)$.

The cost of the conversion protocol equals to one RBVS protocol and one XOR operation. Therefore, the complexity is four rounds and $8l+1$ multiplications. When $p = 2^l - c$, where c is a small integer, the multiplications can be reduced to $2l+1$.

4.2 Computing $([r] > c)$

Given random bitwise-shared values $([r_0],\ldots,[r_{l-1}])$ and a value c, vector $[\vec{e}] = ([e_0],\ldots,[e_{l-1}])$ is computed as follows:

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

In equation (1), $[s_i]$ are uniformly random non-zero values and $[s]$ is a uniformly random shared value of $s \in \{-1,1\}$. By revealing vector $[\vec{e}]$, we can obtain the comparison $([r] > c)$ as follows. It is noteworthy that $e = 1$ if vector $\vec{e}$ has a 0 element, and $e = 0$ otherwise.

$$([r] > c) = e \oplus \left(-\frac{[s]-1}{2}\right) \quad (2)$$

When vector $\vec{e}$ does not have any 0 elements, each e_i is masked by s_i, and e_i does not leak any information regarding shares. However, the equation $r_i = c_i$ holds for $i > m$ when $e_m = 0$ exists in vector $\vec{e}$. Therefore, information regarding r is leaked.

To solve this problem, the parties compute a vector $[\tilde{e}_i]$ shifted by a random unknown value $v \in \{0,\ldots,l-1\}$, as follows:

$$[\tilde{e}_i] = [e_{(i+v) \pmod l}] \quad (3)$$

In equation (3), the location of the 0 is hidden by v. Therefore, revealing $[\tilde{e}_i]$ does not leak any information regarding r.

The round complexity of computing $([r] > c)$ is five rounds. In total, the comparison protocol contains nine rounds. However, one round can be reduced by computing equation (1) before $[r] < p$ is verified. Thus, the round complexity can be reduced to eight rounds.

5 OUR PROTOCOLS

5.1 Reducing the Round Complexity

In existing comparison protocols, the computation of $([r] > c)$ requires five of eight rounds. This is because the computation of a shifted vector requires many rounds. Hence, we present two proposals as follows:

- We compute $[\vec{e}]$ with fewer rounds.
- For permuting $[\vec{e}]$, we apply a shuffling protocol instead of computing a shifted vector.

By using these two proposals, we can construct a five-round secure comparison protocol.

5.2 Computing $[\vec{e}]$ with Fewer Rounds

More round-efficient vector $[\vec{e}] = ([e_0], \ldots, [e_{l-1}])$ are shown below. This equation is based on the homomorphic encryption comparison protocol (Veugen, 2012), which is the improved protocol of (Damgård et al., 2007).

$$[e_i] = [s_i] \left([s] + (c_i - [r_i]) + 3 \sum_{j=i+1}^{l-1} (c_j \oplus [r_j]) \right) \quad (4)$$

Equation (4) requires only one round and l multiplications in contrast to equation (1), which requires two rounds and $2l$ multiplications.

Security. As in equation (1), When vector $\vec{e}$ does not have any 0 elements, each e_i is masked by s_i, and e_i does not leak any information regarding shares. However, the equation $r_i = c_i$ holds for $i > m$ when $e_m = 0$ exists in vector $\vec{e}$, and information regarding r is leaked. Thus, $[e_i]$ must be permuted.

5.3 Subprotocols for Shuffling Protocol

In this section, we describe (2,2)-additive secret sharing and share conversion, which are subprotocols of a shuffling protocol. The shuffling protocol cannot be constructed efficiently on Shamir's (2,3) secret-sharing scheme. Thus, we execute the shuffling protocol after converting shares from Shamir's (2,3) secret sharing to (2,2)-additive secret sharing.

5.3.1 (2,2)-Additive Secret-sharing Scheme

For the sake of simplicity, we assume that the secret $s \in Z_p$ is shared by two parities P_1, P_2. We denote $[\![s]\!]_i$ as P_i's share of s and $[\![s]\!] = ([\![s]\!]_1, [\![s]\!]_2)$ as the shorthand.

Share. randomly choose $r \in Z_p$; let $[\![s]\!]_1 = r$ and $[\![s]\!]_2 = s - r$.

Reveal. Output $s = [\![s]\!]_1 + [\![s]\!]_2$.

5.3.2 Share Conversion

Share conversion from Shamir's (2,3) secret sharing to (2,2)-additive secret sharing can be performed without communication (Cramer et al., 2005). Let λ_i (for $i = 1, 2$) be coefficients of the Lagrange interpolation and compute as follows:

$$[\![s]\!]_i = \lambda_i [s]_i \quad (5)$$

$$\lambda_1 = -\frac{x_2}{x_1 - x_2}$$

$$\lambda_2 = -\frac{x_1}{x_2 - x_1}$$

5.4 Shuffling Protocol

Notations and Definitions. We define the notations used in the shuffling protocol as follows:

- π: a random permutation.
- π_{12}: a random permutation shared by P_1, P_2.
- π_3: a random permutation known to only P_3.
- $\pi \circ \vec{s}$: l vectors permuted by π.
- $\pi \circ [\![\vec{s}]\!]_i$: P_i's (2,2) - additive secret shared l vectors permuted by π.
- $\vec{\beta}_{12}$: l vectors of random values less than p shared by P_1, P_2.

Random Permutation. Herein, random permutation π is written as follows:

$$\pi = \begin{pmatrix} 0 & 1 & \ldots & l-1 \\ \pi(0) & \pi(1) & \ldots & \pi(l-1) \end{pmatrix}$$

When $\vec{x}$ represents l vectors, $\pi \circ \vec{x}$ denotes that $\vec{x}$ is randomly permuted by π, that is, $\vec{x} = (x_0, \ldots, x_{l-1})$ are rearranged into $(\acute{x}_0, \ldots, \acute{x}_{l-1})$ such that $\acute{x}_j = x_{\pi(j)}$. Similarly, $\pi \circ [\![\vec{x}]\!]_i$ denotes that $[\![\vec{x}]\!]_i = ([\![x_0]\!]_i, \ldots, [\![x_{l-1}]\!]_i)$

are reordered into $([\![\acute{x}_0]\!]_i, \ldots, [\![\acute{x}_{l-1}]\!]_i)$ such that $\acute{x}_j = x_{\pi(j)}$.

Algorithm. We describe the shuffling protocol in Algorithm 6, which is the protocol based on (Laur et al., 2011; Chida et al., 2019). This protocol has inputs of shared vector $[\vec{e}] = ([e_0], \ldots, [e_{l-1}])$ and outputs randomly permuted vector $\pi \circ \vec{e} = \pi_3 \circ \pi_{12} \circ \vec{e}$.

Algorithm 6: Shuffling Protocol.

Input: $[\vec{e}] = ([e_0], \ldots, [e_{l-1}])$.
Output: $\pi \circ \vec{e} = \pi_3 \circ \pi_{12} \circ \vec{e}$.
1: P_1 shares π_{12} and $\vec{\beta}_{12}$ with P_2.
2: Convert $[\vec{e}]$ into $[\![\vec{e}]\!]_1$, $[\![\vec{e}]\!]_2$.
3: P_1 sends $[\![\vec{\acute{e}}]\!]_1 = \pi_{12} \circ [\![\vec{e}]\!]_1 - \vec{\beta}_{12}$ to P_3.
4: P_2 sends $[\![\vec{\acute{e}}]\!]_2 = \pi_{12} \circ [\![\vec{e}]\!]_2 + \vec{\beta}_{12}$ to P_3.
5: P_3 computes $\pi_3 \circ ([\![\vec{\acute{e}}]\!]_1 + [\![\vec{\acute{e}}]\!]_2) = \pi_3 \circ \pi_{12} \circ \vec{e}$.
6: P_3 outputs $\pi \circ \vec{e} = \pi_3 \circ \pi_{12} \circ \vec{e}$.

Security. π_3 is known to only P_3, and π_{12} is shared by only P_1, P_2. Therefore, $\pi = \pi_3 \circ \pi_{12}$ is the unknown random permutation, and $\pi \circ \vec{e}$ does not leak any information regarding r because $\vec{e}$ is randomly permuted by π.

Complexity. Similar to other studies, we ignore the complexity of sharing and revealing. However, we determine the communication costs of sending some vectors and random permutations. One multiplication cost is equivalent to sending six values. Therefore, the complexity of sending one value is one round and $1/6$ multiplications. $[\![\vec{\acute{e}}]\!]_1$, $[\![\vec{\acute{e}}]\!]_2$ and $\vec{\beta}_{12}$ are l vectors and π_{12} is a $2 \times l$ matrix. Thus, this protocol requires $5l \times 1/6 = 5l/6$ multiplications. π_{12} and $\vec{\beta}_{12}$ can be shared in parallel, whereas $[\![\vec{\acute{e}}]\!]_1$ and $[\![\vec{\acute{e}}]\!]_2$ can be sent in parallel. Thus, the round complexity is two rounds.

However, the sharing of π_{12} and $\vec{\beta}_{12}$ can be executed simultaneously with other protocols in advance because π_{12} and $\vec{\beta}_{12}$ are random values unrelated to the inputs. Therefore, the actual round complexity is one.

5.5 Secure Comparison Protocol

We describe our secure comparison protocol in Algorithm 7. This protocol has inputs $[a]$, $[b] < \frac{p-1}{2}$ and outputs $[a < b]$, where $(a < b) = 1$ if $a < b$ and $(a < b) = 0$ otherwise.

Complexity. The complexities of our comparison protocol are as follows: three rounds and $8l$ multiplications for the RBVS protocol, two rounds and $3l$ multiplications for $[s]_i$, two rounds and two multiplications for $[s]$, one round and l multiplications for $[e]_i$, one round and $5l/6$ multiplications for the shuffling protocol, and one round and one multiplication for $c_o \oplus [r_0] \oplus ([r] > c)$. The total complexity is 10 rounds and $(12 + 5/6)l + 3$ multiplications. However, the computations of $[s]_i$ and $[s]$ can be executed simultaneously with RBVS protocols in advance because $[s]_i$ and $[s]$ are random values unrelated to the inputs. Therefore, we can ignore these round complexities. In addition, one round can be reduced by computing $[e]_i$ before $[r] < p$ is verified. We assume that two attempts are required to compute $r < p$ for an arbitrary odd p. Therefore, we require an additional of $4l + 2$ multiplications for two sets of $[e]_i$, $[s]_i$, and $[s]$. In total, the complexity is five rounds and $(16 + 5/6)l + 5$ multiplications.

Algorithm 7: Secure Comparison Protocol.

Input: $[a], [b] < \frac{p-1}{2}$.
Output: $[a < b]$
1: $([r_0], \ldots, [r_{l-1}]) \leftarrow RBVS$.
2: $[r] \leftarrow \sum_{i=0}^{l-1} 2^i [r_i]$.
3: $[c] \leftarrow 2([a] - [b]) + [r]$.
4: $c \leftarrow \text{Reveal}([c])$.
5: For $i = 0, \ldots, l-1$ in parallel do
$[s_i] \leftarrow RNNS$.
6: Compute $[s]$. ▷ $s \in \{-1, 1\}$
7: For $i = 0, \ldots, l-1$ in parallel do
$[e_i] = [s_i] \left([s] + (c_i - [r_i]) + 3 \sum_{j=i+1}^{l-1} (c_j \oplus [r_j]) \right)$.
8: $[\vec{e}] = ([e_0], \ldots, [e_{l-1}])$.
9: $\pi \circ \vec{e} \leftarrow \text{Shuffling}([\vec{e}])$.
10: If vector $\pi \circ \vec{e}$ contains a 0 element then $e = 1$; otherwise, $e = 0$.
11: $([r] > c) = e \oplus \left(-\frac{[s]-1}{2} \right)$.
12: Output $([a] < [b]) = c_o \oplus [r_0] \oplus ([r] > c)$.

When $p = 2^l - c$, where c is a small integer, only one attempt is required and the multiplications of the RBVS protocol is $2l$. Thus, the multiplications can be reduced to $(6 + 5/6)l + 3$.

5.6 Secure Interval Test Protocol

The secure interval test protocol has inputs of public constants c_1, c_2 and shared value $[a] \in Z_p$; it outputs $[c_1 < a < c_2]$ without revealing $(c_1 < a < c_2)$. We construct a five-round secure interval test protocol using our proposed comparison protocol and the interval test protocol of (Nishide and Ohta, 2007).

Subprotocol. We describe the subprotocol of our secure interval test protocol in Alogrothm 8. This protocol has inputs of bitwise shared value $([r_0], \ldots, [r_{l-1}])$

and a public constant c; it outputs $[c < r]$, where $(c < r) = 1$ if $c < r$ and $(c < r) = 0$ otherwise. The complexity is two rounds and $(4+5/6)l+2$ multiplications, excluding rounds complexities of $[s]$ and $[s]_i$.

Algorithm 8: Subcomparison Protocol.

Input: $([r_0], \ldots, [r_{l-1}])$, a public constant c.
Output: $[c < r]$

1: For $i = 0, \ldots, l-1$ in parallel do
   $[s_i] \leftarrow RNNS$.
2: Compute $[s]$. ▷ $s \in \{-1, 1\}$
3: For $i = 0, \ldots, l-1$ in parallel do
   $[e_i] = [s_i]\left([s] + (c_i - [r_i]) + 3\sum_{j=i+1}^{l-1}(c_j \oplus [r_j])\right)$.
4: $[\vec{e}] = ([e_0], \ldots, [e_{l-1}])$.
5: $\pi \circ \vec{e} \leftarrow$ Shuffling$([\vec{e}])$.
6: If vector $\pi \circ \vec{e}$ contains a 0 element then $e = 1$; otherwise, $e = 0$.
7: $[c < r] = e \oplus \left(-\frac{[s]-1}{2}\right)$.
8: Output $[c < r]$.

Secure Interval Test Protocol. We describe our secure interval test protocol in Algorithm 9. This protocol has inputs of public constants c_1, c_2, and shared value $[a] \in Z_p$; it outputs $[c_1 < a < c_2]$, where $(c_1 < a < c_2) = 1$ if $c_1 < a < c_2$ and $(c_1 < a < c_2) = 0$ otherwise.

Algorithm 9: Secure Interval Test Protocol.

Input: public constants c_1, c_2, shared value$[a]$.
Output: $[c_1 < a < c_2]$

1: $([r_0], \ldots, [r_{l-1}]) \leftarrow RBVS$.
2: $[r] \leftarrow \sum_{i=0}^{l-1} 2^i [r_i]$.
3: $[c] \leftarrow [a] + [r]$.
4: $c \leftarrow$ Reveal$([c])$.
5: if $c_2 \leqq c$ then
   $r_1 \leftarrow c - c_2$ and $r_2 \leftarrow c - c_1$.
6: else if $c \leqq c_1$ then
   $r_1 \leftarrow c + p - c_2$ and $r_2 \leftarrow c + p - c_1$.
7: $[c_1 < a < c_2] \leftarrow [r_1 < r] \times [r < r_2]$.
8: else if $c_1 < c < c_2$ then
   $r_1 \leftarrow c - c_1 - 1$ and $r_2 \leftarrow c + p - c_2 + 1$.
9: $[c_1 < a < c_2] \leftarrow 1 - ([r_1 < r] \times [r < r_2])$.
10: Output $[c_1 < a < c_2]$.

When $c_2 \leqq c$, as shown in Figure 1, the shared bit $[c_1 < a < c_2]$ equals to 1 if $r_1 = (c - c_2) < r < r_2 = (c - c_1)$. Similarly, when $c \leqq c_1$, as shown in Figure 2, the shared bit $[c_1 < a < c_2]$ equals to 1 if $r_1 = (c + p - c_2) < r < r_2 = (c + p - c_1)$. Therefore, the shared bit $[c_1 < a < c_2]$ can be obtained by computing $[r_1 < r] \times [r < r_2]$ in these two cases.

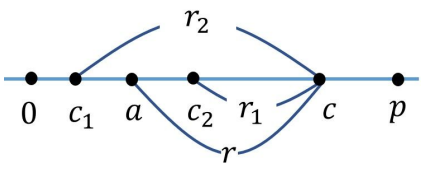

Figure 1: The case of $c_2 \leqq c$.

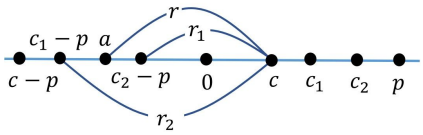

Figure 2: The case of $c \leqq c_1$.

When $c_1 < c < c_2$, as shown in Figure 3, the shared bit $[c_1 < a < c_2]$ equals to 0 if $r_1 = (c - c_1 - 1) < r < r_2 = (c + p - c_2 + 1)$. Therefore, the shared bit $[c_1 < a < c_2]$ can be obtained by computing $1 - [r_1 < r] \times [r < r_2]$ in this case.

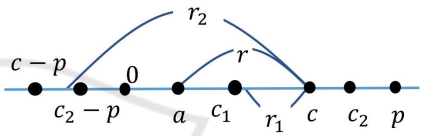

Figure 3: The case of $c_1 < c < c_2$.

Complexity. $[r_1 < r]$ and $[r < r_2]$ can be computed in parallel by Algorithm 8. Thus, the complexities of our interval test are as follows: three rounds and $8l$ multiplications for the RBVS protocol, two rounds and $2 \times ((4+5/6)l+2)$ multiplications for $[r_1 < r]$ and $[r < r_2]$, and one round and one multiplication for $[r_1 < r] \times [r < r_2]$. The total complexity is six rounds and $(16+5/3)l+5$ multiplications. However, one round can be reduced by computing $[e]_i$ before $[r] < p$ is verified, as in the comparison protocol. We assume that two attempts are required to compute $r < p$ for an arbitrary odd p. Therefore, we require an additional of $8l+4$ multiplications for four sets of $[e]_i$, $[s]_i$, and $[s]$. In total, the complexity is five rounds and $(24+5/3)l+9$ multiplications.

When $p = 2^l - c$, where c is a small integer, only one attempt is required and the multiplications of the RBVS protocol is $2l$. Thus, the multiplications can be reduced to $(10+5/3)l+5$.

6 COMPARISON WITH OTHER STUDIES

In this section, we compare our comparison and interval test protocols with existing protocols based on

rounds and multiplications to evaluate the communication costs of the protocols.

In comparison protocols, for the sake of simplicity, we assume that the compared values are restricted to less than $\frac{p-1}{2}$ and $p = 2^l - c$, where c is a small integer. Therefore, only one attempt is required to compute $r < p$ in RBVS protocol. In interval test protocols, inputs are arbitrary values in Z_p, and two attempts are required.

Table 1: Complexities of comparison protocols.

	Rounds	Multiplications
(Damgård et al., 2006)	44	$148l + 188l\log_2 l$
(Nishide and Ohta, 2007)	13	$36l + 1$
(Reistad and Toft, 2007)	8	$20l + 36l\log_2 l + 6$
(Reistad, 2009)	6	$7.5l + 11$
Proposed	5	$(6 + 5/6)l + 3$

Table 2: Complexities of interval test protocols.

	Rounds	Multiplications
(Nishide and Ohta, 2007)	13	$72l + 1$
Proposed	5	$(24 + 5/3)l + 9$

7 CONCLUSIONS

The main results obtained in this study are as follows.

- By using a shuffling protocol, we proposed a five-round secure comparison protocol .
- We constructed a five-round secure interval test protocol by applying our secure comparison protocol.
- We showed that proposed protocols have less communication costs than existing protocols.

In future studies, we will consider methods to further reduce the communication costs of our protocols.

REFERENCES

Catrina, O. and De Hoogh, S. (2010). Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer.

Chida, K., Hamada, K., Ikarashi, D., Kikuchi, R., Kiribuchi, N., and Pinkas, B. (2019). An efficient secure three-party sorting protocol with an honest majority. Cryptology ePrint Archive, Report 2019/695.

Cramer, R., Damgård, I., and Ishai, Y. (2005). Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference*, pages 342–362. Springer.

Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B., and Toft, T. (2006). Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer.

Damgård, I., Geisler, M., and Krøigaard, M. (2007). Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy*, pages 416–430. Springer.

Garay, J., Schoenmakers, B., and Villegas, J. (2007). Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer.

Gennaro, R., Rabin, M. O., and Rabin, T. (1998). Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *podc*, volume 98, pages 101–111. Citeseer.

Laur, S., Willemson, J., and Zhang, B. (2011). Round-efficient oblivious database manipulation. In *Proceedings of the 14th International Conference on Information Security*, ISC'11, pages 262–277, Berlin, Heidelberg. Springer-Verlag.

Morita, H., Attrapadung, N., Teruya, T., Ohata, S., Nuida, K., and Hanaoka, G. (2018). Constant-round client-aided secure comparison protocol. In *ESORICS*.

Nishide, T. and Ohta, K. (2007). Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer.

Reistad, T. I. (2009). Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values. In *SECRYPT*, pages 325–330.

Reistad, T. I. and Toft, T. (2007). Secret sharing comparison by transformation and rotation. In *International Conference on Information Theoretic Security*, pages 169–180. Springer.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Veugen, T. (2012). Improving the dgk comparison protocol. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 49–54. IEEE.