Communication-Efficient (Client-Aided) Secure Two-Party Protocols and Its Application

Satsuya Ohata[1] and Koji Nuida[1,2]

[1] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
satsuya.ohata@aist.go.jp
[2] The University of Tokyo, Tokyo, Japan nuida@mist.i.u-tokyo.ac.jp

Abstract. Secure multi-party computation (MPC) allows a set of parties to compute a function jointly while keeping their inputs private. Compared with the MPC based on garbled circuits, some recent research results show that MPC based on secret sharing (SS) works at a very high speed. Moreover, SS-based MPC can be easily vectorized and achieve higher throughput. In SS-based MPC, however, we need many communication rounds for computing concrete protocols like equality check, less-than comparison, etc. This property is not suited for large-latency environments like the Internet (or WAN). In this paper, we construct semi-honest secure communication-efficient two-party protocols. The core technique is *Beaver triple extension*, which is a new tool for treating multi-fan-in gates, and we also show how to use it efficiently. We mainly focus on reducing the number of communication rounds, and our protocols also succeed in reducing the number of communication bits (in most cases). As an example, we propose a less-than comparison protocol (under practical parameters) with *three* communication rounds. Moreover, the number of communication bits is also 38.4% fewer. As a result, total online execution time is 56.1% shorter than the previous work adopting the same settings. Although the computation costs of our protocols are more expensive than those of previous work, we confirm via experiments that such a disadvantage has small effects on the whole online performance in the typical WAN environments.

1 Introduction

Secure multi-party computation (MPC) [26, 14] allows a set of parties to compute a function f jointly while keeping their inputs private. More precisely, the $N(\geq 2)$ parties, each holding private input x_i for $i \in [1, N]$, are able to compute the output $f(x_1, \cdots, x_N)$ without revealing their private inputs x_i. Some recent research showed there are many progresses in the research on MPC based on secret sharing (SS) and its performance is dramatically improved. SS-based MPC can be easily vectorized and suitable for parallel executions. We can obtain large throughput in SS-based MPC since we have no limit on the size of vectors. This is a unique property on SS-based MPC, and it is compatible with the SIMD operations like mini-batch training in privacy-preserving machine learning. We cannot enjoy this advantage in the MPC based on garbled circuits (GC) or homomorphic encryption (HE). The most efficient MPC scheme so far is three-party computation (3PC) based on 2-out-of-3 SS (e.g., [1, 7]). In two-party computation (2PC), which is the focus of this paper, we need fewer hardware resources than 3PC. Although it does not work at high speed since we need heavy pre-computation, we can mitigate this problem by adopting slightly new MPC models like client/server-aided models that we denote later.

In addition to the advantage as denoted above, the amount of data transfer in online phase is also small in SS-based MPC than GC/HE-based one. However, the number of communication rounds we need for computation is large in SS-based MPC. We need one interaction between computing parties when we compute an arithmetic multiplication gate or a boolean AND gate, which is time-consuming when processing non-linear functions since it is difficult to make the circuit depth shallow. This is a critical disadvantage in real-world privacy-preserving applications since there are non-linear functions we frequently use in practice like equality check, less-than comparison, max value extraction, activation functions in machine learning, etc. In most of the previous research, however, this problem has not been seriously treated. This is because they assumed there is (high-speed) LAN connection between computing parties. Under such environments, total online execution time we need for processing non-linear functions is small even if we need many interactions between computing parties since the communication latency is usually very short (typically ≤ 0.5ms). This assumption is somewhat strange in practice, as the use of LAN suggests that MPC is executed on the network that is maintained by the same administrator/organization. In that case, it is not clear if the requirement for SS that parties do not collude is held or not. Hence, it looks more suitable to assume non-local networks like WAN. However, large communication latency in WAN becomes the performance bottleneck in SS-based MPC. We find by our experiments that the time caused by the communication latency occupies more than 99% in some cases for online total execution time. To reduce the effect of the large communication latency, it is important to develop SS-based MPC with fewer communication rounds. In other words, we should put in work to make the circuit shallower to improve the concrete efficiency of SS-based MPC.

1.1 Related Work

MPC Based on Secret Sharing There are many research results on SS-based MPC. For example, we have results on highly-efficient MPC (e.g., [1, 7]), concrete tools or the toolkit (e.g., [9, 23, 4, 22]), mixed-protocol framework [10, 24, 19], application to privacy-preserving machine learning or data analysis (e.g., [21, 24, 19]), proposal of another model for speeding up the pre-computation [18, 21], etc. As denoted previously, however, we have not been able to obtain good experimental results for computing large circuits over WAN environments. For example, [21] denoted the neural network training on WAN setting is not practical yet.

MPC Based on Garbled Circuit or Homomorphic Encryption There are also many research results on GC/HE-based MPC. For example, we have results on the toolkit (e.g., [17]), encryption switching protocols [16, 8], privacy-preserving machine learning (e.g., [5, 12, 15]), etc. Recently, we have many research results on GC for more than three parties (e.g., [20, 28]). Note that it is difficult to improve the circuit size on standard boolean GC [27], which is a bottleneck on GC-based MPC. Moreover, [3, 6] proposed the GC-based MPC for WAN environments and showed the benchmark using AES, etc. Even if we adopt the most efficient GC [27] with 128-bit security, however, we need to send at least 256-bit string per an AND gate. This is two orders of magnitude larger than SS-based MPC. We construct the round-efficient protocol while keeping data traffic small.

1.2 Our Contribution

There are two main contributions in this paper. First, we propose the method for treating multi-fan-in gates in semi-honest secure SS-based 2PC and show how to use them efficiently. Second, we propose many round-efficient protocols and show their performance evaluations via experiments. We explain the details of them as follows:

1. We propose the method for treating multi-fan-in $\mathsf{MULT}/\mathsf{AND}$ gates over $\mathbb{Z}_{2^n}$ and some techniques for reducing the communication rounds of protocols. Our N-fan-in gates are based on the extension of Beaver triples, which is a technique for computing standard 2-fan-in gates. In our technique, however, we have a disadvantage that the computation costs and the memory costs are exponentially increased by N; that is, we have to limit the size of N in practice. On the other hand, we can improve the costs of communication. More concretely, we can compute arbitrary N-fan-in $\mathsf{MULT}/\mathsf{AND}$ with one communication round and the amount of data transfer is also improved. Moreover, we show performance evaluation results on above multi-fan-in gates via experiments. More concretely, see Sections 3 and 5.1.
2. We propose round-efficient protocols using multi-fan-in gates. We need fewer interactions for our protocols between computing parties in online phase than previous ones. When we use shares over $\mathbb{Z}_{2^{32}}$, compared with the previous work [4], we reduce the communication rounds as follows: $\mathsf{Equality}$: $(5 \to 2)$, $\mathsf{Comparison}$: $(7 \to 3)$, and Max for 3 elements:$(18 \to 4)$. Moreover, we show the performance evaluation results on our protocols via experiments. From our experiments, we find the computation costs for multi-fan-in gates and protocols based on them have small effects on the whole online performance in the typical WAN environments. We also implement an application (a privacy-preserving exact edit distance protocol for genome strings) using our protocols. More concretely, see Sections 4, 5.2, and 5.3.

2 Preliminaries

2.1 Syntax for Secret Sharing

A 2-out-of-2 secret sharing ($(2,2)$-SS) scheme over $\mathbb{Z}_{2^n}$ consists of two algorithms: Share and $\mathsf{Reconst}$. Share takes as input $x \in \mathbb{Z}_{2^n}$, and outputs $(\llbracket x \rrbracket_0, \llbracket x \rrbracket_1) \in \mathbb{Z}_{2^n}^2$, where the bracket notation $\llbracket x \rrbracket_i$ denotes the share of the i-th party (for $i \in \{0,1\}$). We denote $\llbracket x \rrbracket = (\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$ as their shorthand. $\mathsf{Reconst}$ takes as input $\llbracket x \rrbracket$, and outputs x. For arithmetic sharing $\llbracket x \rrbracket^{\mathsf{A}} = (\llbracket x \rrbracket_0^{\mathsf{A}}, \llbracket x \rrbracket_1^{\mathsf{A}})$ and boolean sharing $\llbracket x \rrbracket^{\mathsf{B}} = (\llbracket x \rrbracket_0^{\mathsf{B}}, \llbracket x \rrbracket_1^{\mathsf{B}})$, we consider power-of-two integers n (e.g. $n = 64$) and $n = 1$, respectively.

2.2 Secure Two-Party Computation Based on (2,2)-Additive Secret Sharing

Here, we explain how to compute arithmetic $\mathsf{ADD}/\mathsf{MULT}$ gates on $(2,2)$-additive SS. We use the standard $(2,2)$-additive SS scheme, defined by

- $\mathsf{Share}(x)$: randomly choose $r \in \mathbb{Z}_{2^n}$ and let $\llbracket x \rrbracket_0^{\mathsf{A}} = r$ and $\llbracket x \rrbracket_1^{\mathsf{A}} = x - r \in \mathbb{Z}_{2^n}$.
- $\mathsf{Reconst}(\llbracket x \rrbracket_0^{\mathsf{A}}, \llbracket x \rrbracket_1^{\mathsf{A}})$: output $\llbracket x \rrbracket_0^{\mathsf{A}} + \llbracket x \rrbracket_1^{\mathsf{A}}$.

We can compute fundamental operations; that is, $\mathsf{ADD}(x,y) := x + y$ and $\mathsf{MULT}(x,y) := xy$. $\llbracket z \rrbracket \leftarrow \mathsf{ADD}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ can be done locally by just adding each party's shares on x and on y. $\llbracket w \rrbracket \leftarrow \mathsf{MULT}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ can be done in various ways. We will use the standard method based on Beaver triples (BT) [2]. Such a triple consists of $\mathsf{bt}_0 = (a_0, b_0, c_0)$ and $\mathsf{bt}_1 = (a_1, b_1, c_1)$ such that $(a_0 + a_1)(b_0 + b_1) = (c_0 + c_1)$. Hereafter, a, b, and c denote $a_0 + a_1$, $b_0 + b_1$, and $c_0 + c_1$, respectively. We can compute these BT in offline phase. In this protocol, each i-th party P_i ($i \in \{0,1\}$) can compute the multiplication share $\llbracket z \rrbracket_i = \llbracket xy \rrbracket_i$ as follows: (1) P_i first compute $(\llbracket x \rrbracket_i - a_i)$ and $(\llbracket y \rrbracket_i - b_i)$. (2) P_i sends them to P_{1-i}. (3) P_i reconstruct $x' = x - a$ and $y' = y - b$. (4) P_0 computes $\llbracket z \rrbracket_0 = x'y' + x'b_0 + y'a_0 + c_0$ and P_1 computes $\llbracket z \rrbracket_1 = x'b_1 + y'a_1 + c_1$. Here, $\llbracket z \rrbracket_0$ and $\llbracket z \rrbracket_1$ calculated as above procedures are valid shares of xy; that is, $\mathsf{Reconst}(\llbracket z \rrbracket_0, \llbracket z \rrbracket_1) = xy$. We abuse notations and write the ADD and MULT protocols simply as $\llbracket x \rrbracket + \llbracket y \rrbracket$ and $\llbracket x \rrbracket \cdot \llbracket y \rrbracket$, respectively. Note that similarly to the ADD protocol, we can also locally compute multiplication by constant c, denoted by $c \cdot \llbracket x \rrbracket$.

We can easily extend above protocols to boolean gates. By converting $+$ and $-$ to $\oplus$ in arithmetic ADD and MULT protocols, we can obtain XOR and AND protocols, respectively. We can construct NOT and OR protocols from the properties of these gates. When we compute $\mathsf{NOT}(\llbracket x \rrbracket_0^{\mathsf{B}}, \llbracket x \rrbracket_1^{\mathsf{B}})$, P_0 and P_1 output $\neg \llbracket x \rrbracket_0^{\mathsf{B}}$ and $\llbracket x \rrbracket_1^{\mathsf{B}}$, respectively. When we compute $\mathsf{OR}(\llbracket x \rrbracket, \llbracket y \rrbracket)$, we compute $\neg \mathsf{AND}(\neg \llbracket x \rrbracket, \neg \llbracket y \rrbracket)$. We abuse notations and write the XOR, AND, NOT, and OR protocols simply as $\llbracket x \rrbracket \oplus \llbracket y \rrbracket$, $\llbracket x \rrbracket \wedge \llbracket y \rrbracket$, $\neg \llbracket x \rrbracket$ (or $\overline{\llbracket x \rrbracket}$), and $\llbracket x \rrbracket \vee \llbracket y \rrbracket$, respectively.

2.3 Semi-Honest Security and Client-Aided Model

In this paper, we consider simulation-based security notion in the presence of semi-honest adversaries (for 2PC) as in [13]. As described in [13], composition theorem for the semi-honest model holds; that is, any protocol is privately computed as long as its subroutines are privately computed.

In this paper, we adopt client-aided model [21, 22] (or server-aided model [18]) for 2PC. In this model, a client (other than computing parties) generates and distributes shares of secrets. Moreover, the client also generates and distributes some necessary BTs to the computing parties. This improves the efficiency of offline computation dramatically since otherwise computing parties would have to generate BTs by themselves jointly via heavy cryptographic primitives like homomorphic encryption or oblivious transfer. The only downside for this model is the restriction that any computing party is assumed to not collude with the client who generates BTs for keeping the security.

3 Core Tools for Round-Efficient Protocols

In this section, we propose a core tool for round-efficient 2PC that we call "Beaver triple extension (BTE)". Moreover, we explain some techniques for precomputation to reduce the communication rounds in online phase.

3.1 N-fan-in MULT/AND via N-Beaver Triple Extension

N-Beaver Triple Extension Let N be a positive integer. Let $\mathcal{M} = \mathbb{Z}_M$ for some M (e.g., $M = 2^n$). Write $[1, N] = \{1, 2, \dots, N\}$. We define a client-aided protocol for generating N-BTE as follows:

1. Client randomly chooses $[\![a_{\{\ell\}}]\!]_0$ and $[\![a_{\{\ell\}}]\!]_1$ from $\mathcal{M}$ ($\ell = 1, \dots, N$). Let $a_{\{\ell\}} \leftarrow [\![a_{\{\ell\}}]\!]_0 + [\![a_{\{\ell\}}]\!]_1$. For each $I \subseteq [1, N]$ with $|I| \geq 2$, by setting $a_I \leftarrow \prod_{\ell \in I} a_{\{\ell\}}$, client randomly chooses $[\![a_I]\!]_0 \in \mathcal{M}$ and sets $[\![a_I]\!]_1 \leftarrow a_I - [\![a_I]\!]_0$.
2. Client sends all the $[\![a_I]\!]_0$ to P_0 and all the $[\![a_I]\!]_1$ to P_1.

Note that, in the protocol above, the process of randomly choosing $[\![a_I]\!]_0$ and then setting $[\![a_I]\!]_1 \leftarrow a_I - [\![a_I]\!]_0$ is equivalent to randomly choosing $[\![a_I]\!]_1$ and then setting $[\![a_I]\!]_0 \leftarrow a_I - [\![a_I]\!]_1$. Therefore, the roles of P_0 and P_1 are symmetric.

Multiplication Protocol For $\ell = 1, \dots, N$, let $([\![x_\ell]\!]_0, [\![x_\ell]\!]_1)$ be given shares of ℓ-th secret input value $x_\ell \in \mathcal{M}$. The protocol for multiplication is constructed as follows:

1. Client generates and distributes N-BTE $([\![a_I]\!]_0)_I$ and $([\![a_I]\!]_1)_I$ to the two parties as described above.
2. For $k = 0, 1$, P_k computes $[\![x'_\ell]\!]_k \leftarrow [\![x_\ell]\!]_k - [\![a_{\{\ell\}}]\!]_k$ for $\ell = 1, \dots, N$ and sends those $[\![x'_\ell]\!]_k$ to P_{1-k}.
3. For $k = 0, 1$, P_k computes $x'_\ell \leftarrow [\![x'_\ell]\!]_{1-k} + [\![x'_\ell]\!]_k$ for $\ell = 1, \dots, N$.
4. P_0 outputs $[\![y]\!]_0$ given by

$$[\![y]\!]_0 \leftarrow \prod_{\ell=1}^{N} x'_\ell + \sum_{\emptyset \neq I \subseteq [1,N]} \left(\prod_{\ell \in [1,N] \setminus I} x'_\ell \right) [\![a_I]\!]_0$$

while P_1 outputs $[\![y]\!]_1$ given by

$$[\![y]\!]_1 \leftarrow \sum_{\emptyset \neq I \subseteq [1,N]} \left(\prod_{\ell \in [1,N] \setminus I} x'_\ell \right) [\![a_I]\!]_1 \ .$$

We can prove the correctness and semi-honest security of this protocol. Due to the page limitation, we show the proofs in the full version of this paper.

3.2 Discussion on Beaver Triple Extension

We can achieve the same functionality of N-$\mathsf{MULT/AND}$ by using 2-$\mathsf{MULT/AND}$ multiple times and there are some trade-offs between these two strategies. In the computation of N-fan-in $\mathsf{MULT/AND}$ using N-BTE, the memory consumption and computation cost increase exponentially with N. Therefore, we have to put a restriction on the size of N and concrete settings change optimal N. In this paper, we use N-$\mathsf{MULT/AND}$ for $N \leq 9$ to construct round-efficient protocols. On the other hand, N-fan-in $\mathsf{MULT/AND}$ using N-BTE needs fewer communication costs. Notably, the number of communication rounds of our protocol does not depend on N and this improvement has significant effects on practical performances in WAN settings. Because of the problems on the memory/computation costs we denoted above, however, there is a limitation for the size of N. When we use L-fan-in $\mathsf{MULT/AND}$ ($L \leq N$) gates, we need $\frac{\lceil \log N \rceil}{\lfloor \log L \rfloor}$ communication rounds for computing N-fan-in $\mathsf{MULT/AND}$.

Damgård et al. [9] also proposed how to compute N-fan-in gates in a round-efficient manner using Lagrange interpolation. Each of their scheme and ours has its merits and demerits. Their scheme has an advantage over memory consumption and computational costs; that is, their N-fan-in gates do not need exponentially large memory and computation costs. On the other hand, their scheme needs two communication rounds to compute N-fan-in gates for any N and requires the share spaces to be $\mathbb{Z}_p$ (p: prime). A 2PC scheme over $\mathbb{Z}_{2^n}$ is sometimes more efficient than one over $\mathbb{Z}_p$ when we implement them using low-level language (e.g., C++) since we do not have to compute remainders modulo 2^n for all arithmetic operations.

3.3 More Techniques for Reducing Communication Rounds

On Weights At Most One We consider the plain input x that all bits are 0, or only a single bit is 1 and others are 0. In this setting, we can compute the share representing whether all the bits of x are 0 or not without communications between P_0 and P_1. More concretely, we can compute it by locally computing XOR for all bits on each share. This technique is implicitly used in the previous work [4] for constructing an arithmetic overflow detection protocol $\mathsf{Overflow}$, which is an important building block for constructing less-than comparison and more. We show more skillful use of this technique for constructing $\mathsf{Overflow}$ to avoid heavy computation in our protocols. More concretely, see Section 4.2.

Arithmetic Blinding We consider the situation that two clients who have secrets also execute computation (i.e., an input party is equal to a computing party), which is the different setting from client-aided 2PC. During the multiplication protocol, both P_0 and P_1 obtain $x - a$ and $y - b$. Here, P_0 finds a and P_1 finds b since P_0 and P_1 know the value of x and y, respectively. Therefore, it does not matter if P_0 and P_1 previously know the corresponding values; that is, P_0 can send b_0 to P_1 and P_1 can send a_1 to P_0 in the pre-computation phase. This operation does not cause security problems. By above pre-processing, P_0 and P_1 can directly send $x - a$ and $y - b$ in the multiplication protocol, respectively. As a result, we can reduce the amount of data transfer in the multiplication protocol. Even in the client-aided 2PC setting, this situation appears in the boolean-to-arithmetic conversion protocol. More concretely, see Section 4.3.

Trivial Sharing We consider the setting that an input party is not equal to a computing party, which is the same one as standard client-aided 2PC. In this situation, we can use the share $[\![b]\!]_i$ $(i \in \{0, 1\})$ itself as a secret value for computations by considering another party has the share $[\![0]\!]_{1-i}$. Although we find this technique in the previous work [4], we can further reduce the communication rounds of two-party protocols by combining this technique and BTE. More concretely, see Section 4.3.

4 Communication-Efficient Protocols

In this section, we show round-efficient 2PC protocols using BTE and the techniques in Section 3.3. For simplicity, in this section, we set a share space to $\mathbb{Z}_{2^{16}}$ and use N-fan-in gates $(N \leq 5)$ to explain our proposed protocols. Although we omit the protocols over $\mathbb{Z}_{2^{32}}/\mathbb{Z}_{2^{64}}$ due to the page limitation, we can obtain the protocols with the same communication rounds with $\mathbb{Z}_{2^{16}}$ by using 7 or less fan-in AND over $\mathbb{Z}_{2^{32}}$ and 9 or less fan-in AND over $\mathbb{Z}_{2^{64}}$. We omit the correctness of the protocols adopting the same strategy in the previous work [4].

4.1 Equality Check Protocol and Its Application

An equality check protocol $\mathsf{Equality}([\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}})$ outputs $[\![z]\!]^{\mathsf{B}}$, where $z = 1$ iff $x = y$. We start from the approach by [4] and focus on reducing communication rounds. In $\mathsf{Equality}$, roughly speaking, we first compute $t = x - y$ and then check if all bits of t are 0 or not. If all the bits of t are 0, it means $t = x - y = 0$. We show our two-round $\mathsf{Equality}$ as in Algorithm 1: In our strategy, more generally, we need $\frac{\lceil \log n \rceil}{\lfloor \log L \rfloor}$ communication rounds for executing $\mathsf{Equality}$ when we set the share space to $\mathbb{Z}_{2^n}$ and use N-OR $(N \leq L)$. By using our $\mathsf{Equality}$, we can also obtain a three-round round-efficient table lookup protocol TLU. We show the construction of our TLU in the full version of this paper.

Algorithm 1 Our Proposed Equality

```
Functionality: $[\![z]\!]^{\mathsf{B}} \leftarrow \mathsf{Equality}([\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}})$
Ensure: $[\![z]\!]^{\mathsf{B}}$, where $z = 1$ iff $x = y$.
1: $P_0$ and $P_1$ locally compute $[\![t]\!]^{\mathsf{A}}_0 = [\![x]\!]^{\mathsf{A}}_0 - [\![y]\!]^{\mathsf{A}}_0$ and $[\![t]\!]^{\mathsf{A}}_1 = [\![y]\!]^{\mathsf{A}}_1 - [\![x]\!]^{\mathsf{A}}_1$, respectively.
2: $P_i$ $(i \in \{0,1\})$ locally extend $[\![t]\!]^{\mathsf{A}}_i$ to binary and see them as boolean shares; that
   is, $P_i$ obtain $[[\![t[15]]\!]^{\mathsf{B}}_i, \cdots, [\![t[0]]\!]^{\mathsf{B}}_i]$.
3: $P_i$ compute $[\![t'[j]]\!]^{\mathsf{B}} \leftarrow 4\text{-}\mathsf{OR}([\![t[4j]]\!]^{\mathsf{B}}, [\![t[4j+1]]\!]^{\mathsf{B}}, [\![t[4j+2]]\!]^{\mathsf{B}}, [\![t[4j+3]]\!]^{\mathsf{B}})$
   for $j \in [0, \cdots, 3]$.
4: $P_i$ compute $[\![t'']\!]^{\mathsf{B}} \leftarrow 4\text{-}\mathsf{OR}([\![t'[0]]\!]^{\mathsf{B}}, [\![t'[1]]\!]^{\mathsf{B}}, [\![t'[2]]\!]^{\mathsf{B}}, [\![t'[3]]\!]^{\mathsf{B}})$.
5: $P_i$ compute $[\![z]\!]^{\mathsf{B}} = \neg[\![t'']\!]^{\mathsf{B}}$.
6: return $[\![z]\!]^{\mathsf{B}}$.
```

Algorithm 2 Our Proposed MSNZB

```
Functionality: $[\![z]\!]^{\mathsf{B}} \leftarrow \mathsf{MSNZB}([\![x]\!]^{\mathsf{B}})$
Ensure: $[\![z]\!]^{\mathsf{B}} = [[\![z[15]]\!]^{\mathsf{B}}, \cdots, [\![z[0]]\!]^{\mathsf{B}}]$, where $z[j] = 1$ for the largest value $j$ such that
   $x[j] = 1$ and $z[k] = 0$ for all $j \neq k$.
1: $P_i$ $(i \in \{0,1\})$ set $[\![t[j]]\!]^{\mathsf{B}}_i = [\![x[j]]\!]^{\mathsf{B}}_i$ for $j \in [3, 7, 11, 15]$. Then $P_i$ parallelly compute
   $[\![t[j]]\!]^{\mathsf{B}} \leftarrow 2\text{-}\mathsf{OR}([\![x[j]]\!]^{\mathsf{B}}, [\![x[j+1]]\!]^{\mathsf{B}})$ for $j \in [2, 6, 10, 14]$,
   $[\![t[j]]\!]^{\mathsf{B}} \leftarrow 3\text{-}\mathsf{OR}([\![x[j]]\!]^{\mathsf{B}}, [\![x[j+1]]\!]^{\mathsf{B}}, [\![x[j+2]]\!]^{\mathsf{B}})$ for $j \in [1, 5, 9, 13]$, and
   $[\![t[j]]\!]^{\mathsf{B}} \leftarrow 4\text{-}\mathsf{OR}([\![x[j]]\!]^{\mathsf{B}}, [\![x[j+1]]\!]^{\mathsf{B}}, [\![x[j+2]]\!]^{\mathsf{B}}, [\![x[j+3]]\!]^{\mathsf{B}})$ for $j \in [0, 4, 8, 12]$.
2: $P_i$ compute $[\![t'[j]]\!]^{\mathsf{B}}_i = [\![t[j]]\!]^{\mathsf{B}}_i$ for $j \in [3, 7, 11, 15]$ and compute
   $[\![t'[j]]\!]^{\mathsf{B}}_i = [\![t[j]]\!]^{\mathsf{B}}_i \oplus [\![t[j+1]]\!]^{\mathsf{B}}_i$ for $j \in [0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14]$.
3: $P_i$ locally compute $[\![s[j]]\!]^{\mathsf{B}}_i = \bigoplus_{k=4j}^{4j+3} [\![t'[k]]\!]^{\mathsf{B}}_i$ for $j \in [1, 2, 3]$.
4: $P_i$ compute $[\![z[j]]\!]^{\mathsf{B}}_i = [\![t'[j]]\!]^{\mathsf{B}}_i$ for $j \in [12, \cdots, 15]$. Then $P_i$ parallelly compute
   $[\![z[j]]\!]^{\mathsf{B}} \leftarrow 2\text{-}\mathsf{AND}([\![t'[j]]\!]^{\mathsf{B}}, \neg[\![s[3]]\!]^{\mathsf{B}})$ for $j \in [8, \cdots, 11]$,
   $[\![z[j]]\!]^{\mathsf{B}} \leftarrow 3\text{-}\mathsf{AND}([\![t'[j]]\!]^{\mathsf{B}}, \neg[\![s[2]]\!]^{\mathsf{B}}, \neg[\![s[3]]\!]^{\mathsf{B}})$ for $j \in [4, \cdots, 7]$, and
   $[\![z[j]]\!]^{\mathsf{B}} \leftarrow 4\text{-}\mathsf{AND}([\![t'[j]]\!]^{\mathsf{B}}, \neg[\![s[1]]\!]^{\mathsf{B}}, \neg[\![s[2]]\!]^{\mathsf{B}}, \neg[\![s[3]]\!]^{\mathsf{B}})$ for $j \in [0, \cdots, 3]$.
5: return $[\![z]\!]^{\mathsf{B}} = [[\![z[15]]\!]^{\mathsf{B}}, \cdots, [\![z[0]]\!]^{\mathsf{B}}]$.
```

4.2 Overflow Detection Protocol and Applications

An arithmetic overflow detection protocol Overflow has many applications and is also a core building block of less-than comparison protocol. The same as the approach by [4], we construct Overflow via the most significant non-zero bit extraction protocol MSNZB. We first explain how to construct MSNZB efficiently and then show two-round Overflow.

A protocol for extracting the most significant non-zero bit ($\mathsf{MSNZB}([\![x]\!]^{\mathsf{B}} = [[\![x[15]]\!]^{\mathsf{B}}, \cdots, [\![x[0]]\!]^{\mathsf{B}}])$) finds the position of the first "1" of the x and outputs such a boolean share vector $[\![z]\!]^{\mathsf{B}} = [[\![z[15]]\!]^{\mathsf{B}}, \cdots, [\![z[0]]\!]^{\mathsf{B}}]$. In [4], we used a "prefix-OR" operation for executing MSNZB. On the other hand, in our construction, we first separate a bit string into some blocks and compute in-block MSNZB. Then, we compute correct MSNZB for x via in-block MSNZB. We show our two-round MSNZB as in Algorithm 2. Based on the above MSNZB, we can construct an arithmetic overflow detection protocol $\mathsf{Overflow}([\![x]\!]^{\mathsf{A}}, k)$. This protocol outputs $[\![z]\!]^{\mathsf{B}}$, where $z = 1$ iff the condition $([\![x]\!]^{\mathsf{A}}_0 \mod 2^k + [\![x]\!]^{\mathsf{A}}_1 \mod 2^k) \geq 2^k$ holds. We also start from the approach by [4]. In their Overflow,

Algorithm 3 Our Proposed Overflow

```
Functionality: $[\![z]\!]^{\mathsf{B}} \leftarrow \mathsf{Overflow}([\![x]\!]^{\mathsf{A}}, k)$
Ensure: $[\![z]\!]^{\mathsf{B}}$, where $z = 1$ iff $([\![x]\!]^{\mathsf{A}}_0 \mod 2^k) + ([\![x]\!]^{\mathsf{A}}_1 \mod 2^k) \geq 2^k$.
 1: $P_0$ locally extends $([\![x]\!]^{\mathsf{A}}_0 \mod 2^k)$ to binary and obtains
    $[\![d]\!]^{\mathsf{B}}_0 = [[\![d[15]]\!]^{\mathsf{B}}_0, \cdots, [\![d[0]]\!]^{\mathsf{B}}_0]$. $P_1$ also locally extends $(-[\![x]\!]^{\mathsf{A}}_1 \mod 2^k)$ to binary
    and obtains $[\![d]\!]^{\mathsf{B}}_1 = [[\![d[15]]\!]^{\mathsf{B}}_1, \cdots, [\![d[0]]\!]^{\mathsf{B}}_1]$.
 2: $P_i$ $(i \in \{0,1\})$ set $[\![t[j]]\!]^{\mathsf{B}}_i = [\![d[j]]\!]^{\mathsf{B}}_i$ for $j \in [3,7,11,15]$. Then $P_i$ parallelly compute
    $[\![t[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{2\text{-}OR}([\![d[j]]\!]^{\mathsf{B}}, [\![d[j+1]]\!]^{\mathsf{B}})$ for $j \in [2,6,10,14]$,
    $[\![t[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{3\text{-}OR}([\![d[j]]\!]^{\mathsf{B}}, [\![d[j+1]]\!]^{\mathsf{B}}, [\![d[j+2]]\!]^{\mathsf{B}})$ for $j \in [1,5,9,13]$, and
    $[\![t[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{4\text{-}OR}([\![d[j]]\!]^{\mathsf{B}}, [\![d[j+1]]\!]^{\mathsf{B}}, [\![d[j+2]]\!]^{\mathsf{B}}, [\![d[j+3]]\!]^{\mathsf{B}})$ for $j \in [0,4,8,12]$.
 3: $P_i$ compute $[\![t'[j]]\!]^{\mathsf{B}}_i = [\![t[j]]\!]^{\mathsf{B}}_i$ for $j \in [3,7,11,15]$ and compute
    $[\![t'[j]]\!]^{\mathsf{B}}_i = [\![t[j]]\!]^{\mathsf{B}}_i \oplus [\![t[j+1]]\!]^{\mathsf{B}}_i$ for $j \in [0,1,2,4,5,6,8,9,10,12,13,14]$.
 4: $P_i$ locally compute $[\![w[j]]\!]^{\mathsf{B}}_i = \bigoplus_{k=4j}^{4j+3} [\![t'[k]]\!]^{\mathsf{B}}_i$ for $j \in [1,2,3]$.
 5: $P_0$ sets $[\![u[j]]\!]^{\mathsf{B}}_0 = 0$ for $j \in [0,\cdots,15]$ and
    $P_1$ sets $[\![u[j]]\!]^{\mathsf{B}}_1 = [\![d[j]]\!]^{\mathsf{B}}_1$ for $j \in [0,\cdots,15]$.
 6: $P_i$ parallelly compute
    $[\![v[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{2\text{-}AND}([\![t'[j]]\!]^{\mathsf{B}}, [\![u[j]]\!]^{\mathsf{B}})$ for $j \in [12,\cdots,15]$,
    $[\![v[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{3\text{-}AND}([\![t'[j]]\!]^{\mathsf{B}}, [\![u[j]]\!]^{\mathsf{B}}, \neg[\![w[3]]\!]^{\mathsf{B}})$ for $j \in [8,\cdots,11]$,
    $[\![v[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{4\text{-}AND}([\![t'[j]]\!]^{\mathsf{B}}, [\![u[j]]\!]^{\mathsf{B}}, \neg[\![w[2]]\!]^{\mathsf{B}}, \neg[\![w[3]]\!]^{\mathsf{B}})$ for $j \in [4,\cdots,7]$, and
    $[\![v[j]]\!]^{\mathsf{B}} \leftarrow \mathsf{5\text{-}AND}([\![t'[j]]\!]^{\mathsf{B}}, [\![u[j]]\!]^{\mathsf{B}}, \neg[\![w[1]]\!]^{\mathsf{B}}, \neg[\![w[2]]\!]^{\mathsf{B}}, \neg[\![w[3]]\!]^{\mathsf{B}})$ for $j \in [0,\cdots,3]$.
 7: $P_i$ locally compute $[\![z]\!]^{\mathsf{B}}_i = \bigoplus_{\ell=0}^{15} [\![v[\ell]]\!]^{\mathsf{B}}_i$.
 8: $P_i$ compute $[\![z]\!]^{\mathsf{B}} = \neg[\![z]\!]^{\mathsf{B}}$.
 9: If $[\![x]\!]^{\mathsf{A}}_1 = 0$, then $P_1$ locally computes $[\![z]\!]^{\mathsf{B}}_1 = [\![z]\!]^{\mathsf{B}}_1 \oplus 1$
10: return $[\![z]\!]^{\mathsf{B}}$.
```

we check whether or not there exists 1 in $u = (-[\![x]\!]_1 \mod 2^k)$ at the same position of MSNZB on $d = (([\![x]\!]_0 \mod 2^k) \oplus (-[\![x]\!]_1 \mod 2^k))$. Even if we apply our two-round MSNZB in this section, we need three communication rounds for their Overflow since we need one more round to check the above condition using 2–AND. Here, we consider further improvements by combining MSNZB and 2–AND; that is, we increase the fan-in of AND on the step 4 in Algorithm 2 and push the computation of 2–AND into that step as in Algorithm 3: Moreover, we can construct one-round Overflow for small shares spaces (in practice). We show a concrete construction in the full version of this paper.

We have many applications of Overflow like less-than comparison Comparison, which is a building block of the maximum value extraction protocol. In particular, thanks to the round-efficient Overflow, we can obtain a three-round Comparison. Morita et al. [22] proposed a constant (= five)-round Comparison using multi-fan-in gates that works under the shares over $\mathbb{Z}_p$ [9]. Our Comparison is more round-efficient than theirs under the parameters we consider in this paper.

4.3 Boolean-to-Arithmetic Conversion Protocol and Extensions

A boolean-to-arithmetic conversion protocol $\mathsf{B2A}([\![x]\!]^{\mathsf{B}})$ outputs $[\![z]\!]^{\mathsf{A}}$, where $z = x$. In (1-bit) boolean shares, there are four cases; that is, $([\![x]\!]^{\mathsf{B}}_0, [\![x]\!]^{\mathsf{B}}_1) = (0,0)$, $(0,1), (1,0), (1,1)$. Even if we consider these boolean shares as arithmetic ones,

Algorithm 4 Our Proposed $\mathsf{B2A}$

```
Functionality: $[\![z]\!]^{\mathsf{A}} \leftarrow \mathsf{B2A}([\![x]\!]^{\mathsf{B}})$
Ensure: $[\![z]\!]^{\mathsf{A}}$, where $z = x$.
1: In pre-computation phase, the client randomly chooses $a, b \in \mathbb{Z}_{2^{16}}$, computes $c = ab$, chooses a randomness $r \in \mathbb{Z}_{2^{16}}$, and sets $(c_0, c_1) = (r, c - r)$. Then the client sends $(a, c_0)$ and $(b, c_1)$ to $P_0$ and $P_1$, respectively.
2: $P_i$ $(i \in \{0,1\})$ set $[\![x]\!]^{\mathsf{A}}_i = [\![x]\!]^{\mathsf{B}}_i$.
3: $P_0$ computes $x' = [\![x]\!]^{\mathsf{A}}_0 - a$ and $P_1$ computes
   $x'' = [\![x]\!]^{\mathsf{A}}_1 - b$. Then they send them to each other.
4: $P_0$ computes $[\![z]\!]^{\mathsf{A}}_0 = [\![x]\!]^{\mathsf{A}}_0 - 2(x'x'' + x'' \cdot a + c_0)$ and
   $P_1$ computes $[\![z]\!]^{\mathsf{A}}_1 = [\![x]\!]^{\mathsf{A}}_1 - 2(x' \cdot b + c_1)$.
5: return $[\![z]\!]^{\mathsf{A}}$
```

it works well in the first three cases; that is, $0 \oplus 0 = 0 + 0$, $0 \oplus 1 = 0 + 1$, and $1 \oplus 0 = 1 + 0$. However, $1 \oplus 1 \neq 1 + 1$ and we have to correct the output of this case. Based on this idea and the technique in Section 3.3 (trivial sharing), [4] proposed the construction of $\mathsf{B2A}$. In their protocol, we use a standard arithmetic multiplication protocol and need one communication round. In the setting of client-aided 2PC, however, $\mathsf{B2A}$ satisfies the condition that input party is equal to the computing party. Therefore, we can apply the techniques in Section 3.3 (arithmetic blinding) and construct more efficient $\mathsf{B2A}$ as in Algorithm 4: Although the number of communication rounds is the same as in [4], our protocol is more efficient. First, the data transfer in online phase is reduced from $2n$-bits to n-bits. Moreover, the number of randomnesses we need in pre-computation is reduced from five to three, and the data amount for sending from the client to P_0 and P_1 is reduced from $3n$-bits to $2n$-bits.

We can extend the above idea and obtain protocols like $\mathsf{BX2A}$: $[\![b]\!]^{\mathsf{B}} \times [\![x]\!]^{\mathsf{A}} = [\![bx]\!]^{\mathsf{A}}$, $\mathsf{BC2A}$: $[\![b]\!]^{\mathsf{B}} \times [\![c]\!]^{\mathsf{B}} = [\![bc]\!]^{\mathsf{A}}$, and $\mathsf{BCX2A}$: $[\![b]\!]^{\mathsf{B}} \times [\![c]\!]^{\mathsf{B}} \times [\![x]\!]^{\mathsf{A}} = [\![bcx]\!]^{\mathsf{A}}$. These protocols are useful when we construct a round-efficient maximum value extraction protocol (and its variants) in Section 4.4.

$\mathsf{BX2A}$: $[\![b]\!]^{\mathsf{B}} \times [\![x]\!]^{\mathsf{A}} = [\![bx]\!]^{\mathsf{A}}$ We usually need to compute the multiplication of a boolean share $[\![b]\!]^{\mathsf{B}}$ and an arithmetic one $[\![x]\!]^{\mathsf{A}}$ (e.g., TLU, ReLU function in neural networks). We call this protocol $\mathsf{BX2A}$ in this paper. [19] proposed one-round $\mathsf{BX2A}$ under the $(2,3)$-replicated SS, such construction in 2PC has not been known. By almost the same idea as $\mathsf{B2A}$, we can construct one-round $\mathsf{BX2A}$ in 2PC as follows:

1. P_i $(i \in \{0,1\})$ set $[\![b]\!]^{\mathsf{A}}_i = [\![b]\!]^{\mathsf{B}}_i$.
2. P_0 sets $[\![b']\!]^{\mathsf{A}}_0 = [\![b]\!]^{\mathsf{B}}_0$ and $[\![b'']\!]^{\mathsf{A}}_0 = 0$, and P_1 sets $[\![b']\!]^{\mathsf{A}}_1 = 0$ and $[\![b'']\!]^{\mathsf{A}}_1 = [\![b]\!]^{\mathsf{B}}_1$.
3. P_i compute
$$[\![s]\!]^{\mathsf{A}}_i \leftarrow 2\text{-}\mathsf{MULT}([\![b]\!]^{\mathsf{A}}, [\![x]\!]^{\mathsf{A}})$$
$$[\![t]\!]^{\mathsf{A}}_i \leftarrow 3\text{-}\mathsf{MULT}([\![b']\!]^{\mathsf{A}}, [\![b'']\!]^{\mathsf{A}}, [\![x]\!]^{\mathsf{A}}).$$
4. P_i computes $[\![z]\!]^{\mathsf{A}}_i = [\![s]\!]^{\mathsf{A}}_i - 2[\![t]\!]^{\mathsf{A}}_i$.

Here, we denote this computation as $[\![bx - 2b_0b_1x]\!]^{\mathsf{A}}$.

BC2A: $[\![\boldsymbol{b}]\!]^{\mathsf{B}} \times [\![\boldsymbol{c}]\!]^{\mathsf{B}} = [\![\boldsymbol{bc}]\!]^{\mathsf{A}}$ Almost the same idea as BX2A, we can compute $[\![b]\!]^{\mathsf{B}} \times [\![c]\!]^{\mathsf{B}} = [\![bc]\!]^{\mathsf{A}}$ (BC2A) with one communication round. We use this protocol in 3-Argmax/3-Argmin in Section 4.4. We can construct one-round BC2A by computing

$$[\![bc - 2b_0b_1 - 2c_0c_1 + 2b_0\overline{c_0}b_1\overline{c_1} + 2\overline{b_0}c_0\overline{b_1}c_1]\!]^{\mathsf{A}}.$$

We need 2-MULT and 4-MULT for this protocol.

BCX2A: $[\![\boldsymbol{b}]\!]^{\mathsf{B}} \times [\![\boldsymbol{c}]\!]^{\mathsf{B}} \times [\![\boldsymbol{x}]\!]^{\mathsf{A}} = [\![\boldsymbol{bcx}]\!]^{\mathsf{A}}$ Almost the same idea as the above protocols, we can also compute $[\![b]\!]^{\mathsf{B}} \times [\![c]\!]^{\mathsf{B}} \times [\![x]\!]^{\mathsf{A}} = [\![bc]\!]^{\mathsf{A}}$ (BCX2A) with one communication round. We use this protocol in Max/Min in Section 4.4. We can construct one-round BC2A by computing

$$[\![bcx - 2b_0b_1x - 2c_0c_1x + 2b_0\overline{c_0}b_1\overline{c_1}x + 2\overline{b_0}c_0\overline{b_1}c_1x]\!]^{\mathsf{A}}.$$

We need 3-MULT and 5-MULT for this protocol.

4.4 The Maximum Value Extraction Protocol and Extensions

The maximum value extraction protocol $\mathsf{Max}([\![\boldsymbol{x}]\!]^{\mathsf{A}})$ outputs $[\![z]\!]^{\mathsf{A}}$, where z is the largest value in $\boldsymbol{x}$. We first explain the case of Max for three elements (3-Max), which is used for computing edit distance, etc. We denote a j-th element of $\boldsymbol{x}$ as $x[j]$; that is, $\boldsymbol{x} = [x[0], x[1], x[2]]$. We start from a standard tournament-based construction. If the condition $x[0] < x[1]$ holds, $x' = x[1]$. Otherwise, $x' = x[0]$. By repeating the above procedure once more using $[\![x']\!]^{\mathsf{A}}$ and $[\![x[2]]\!]^{\mathsf{A}}$, we can extract the maximum value among $\boldsymbol{x}$. In this strategy, we need 16 $(= (6+1+1) \times 2)$ communication rounds, and 8 $(= (3+1) \times 2)$ communication rounds even if we apply our three-round Comparison (in Section 4.2) and BX2A (in Section 4.3). This is mainly because we cannot parallelly execute Comparison. To solve this disadvantage, we first check the magnitude relationship for all elements using Comparison. Then we extract the maximum value. Based on these ideas, we show our 3-Max as in Algorithm 5: Although the computation costs obviously increased, this is four-round 3-Max by applying our Comparison and BCX2A. Based on the above idea, we can also obtain the minimum value extraction protocol, argument of the maximum/minimum extraction protocols, and (argument of) the maximum/minimum value extraction protocols with $N(> 3)$ inputs. We show the construction of these protocols in the full version of this paper.

5 Performance Evaluation

We demonstrate the practicality of our arithmetic/boolean gates and protocols. We implemented 2PC simulators and performed all benchmarks on a single

Algorithm 5 Our Proposed 3-Max

```
Functionality: $[\![z]\!]^{\mathsf{A}} \leftarrow \mathsf{Max}([\![\boldsymbol{x}]\!]^{\mathsf{A}})$
Ensure: $[\![z]\!]^{\mathsf{A}}$, where $z$ is the largest element in $\boldsymbol{x}$.
1: $P_i$ ($i \in \{0,1\}$) parallelly compute
   $[\![c_{01}]\!]^{\mathsf{B}} \leftarrow \mathsf{Comparison}([\![x[0]]\!]^{\mathsf{A}}, [\![x[1]]\!]^{\mathsf{A}})$,
   $[\![c_{02}]\!]^{\mathsf{B}} \leftarrow \mathsf{Comparison}([\![x[0]]\!]^{\mathsf{A}}, [\![x[2]]\!]^{\mathsf{A}})$, and
   $[\![c_{12}]\!]^{\mathsf{B}} \leftarrow \mathsf{Comparison}([\![x[1]]\!]^{\mathsf{A}}, [\![x[2]]\!]^{\mathsf{A}})$.
2: $P_i$ compute $[\![c_{10}]\!]_i^{\mathsf{B}} = \neg[\![c_{01}]\!]_i^{\mathsf{B}}$, $[\![c_{20}]\!]_i^{\mathsf{B}} = \neg[\![c_{02}]\!]_i^{\mathsf{B}}$, and $[\![c_{21}]\!]_i^{\mathsf{B}} = \neg[\![c_{12}]\!]_i^{\mathsf{B}}$.
3: $P_i$ parallelly compute
   $[\![t[0]]\!]_i^{\mathsf{A}} \leftarrow \mathsf{BCX2A}([\![c_{10}]\!]^{\mathsf{B}}, [\![c_{20}]\!]^{\mathsf{B}}, [\![x[0]]\!]^{\mathsf{A}})$,
   $[\![t[1]]\!]_i^{\mathsf{A}} \leftarrow \mathsf{BCX2A}([\![c_{01}]\!]^{\mathsf{B}}, [\![c_{21}]\!]^{\mathsf{B}}, [\![x[1]]\!]^{\mathsf{A}})$, and
   $[\![t[2]]\!]_i^{\mathsf{A}} \leftarrow \mathsf{BCX2A}([\![c_{02}]\!]^{\mathsf{B}}, [\![c_{12}]\!]^{\mathsf{B}}, [\![x[2]]\!]^{\mathsf{A}})$.
4: $P_i$ compute $[\![z]\!]_i^{\mathsf{A}} = \Sigma_{j=0}^{2}[\![t[j]]\!]_i^{\mathsf{A}}$.
5: return $[\![z]\!]^{\mathsf{A}}$.
```

laptop computer with Intel Core i7-6700K 4.00GHz and 64GB RAM. We implemented simulators using Python 3.7 with Numpy v1.16.2 and vectorized all gates/protocols. We assumed 10MB/s (= 80000bits/ms) bandwidth and 40ms RTT latency as typical WAN settings, and calculate the data transfer time (DTT) and communication latency (CL) using these values. We adopted the client-aided model; that is, we assumed in our experiments that clients generate BTE in their local environment without using HE/OT.

5.1 Performance of Basic Gates

Here we show experimental results on N-AND. We set $N = [2, \cdots, 9]$ and 1 to $10^6 (= 1000000)$ batch in our experiments. Here we mainly show the experimental results on the cases of 1/1000/1000000 batch. The results are as in Table 1 and Figure 1: We find (1) the pre-computation time, online computation time, and data transfer time are exponentially growing up with respect to N; (2) the dominant part in online total execution time is WAN latency especially in the case of small batch. If we compute $N(> 2)$-AND using multiple 2-AND gates, we need two or more communication rounds. Therefore, our scheme is especially suitable for the 2PC with relatively small batch (e.g., $\leq 10^5$) as it yields low WAN latency.

5.2 Performance of Our Protocols

Here we show experimental results on our proposed protocols ($\mathsf{Equality}$, $\mathsf{Comparison}$, and 3-Max). We implemented the baseline protocols [4] and our proposed ones in Section 4. Same as the evaluation of N-AND, we mainly show the results of our experiments over $\mathbb{Z}_{2^{32}}$ with 1/1000/1000000 batch in Table 2 and Figure 2 (relations between batch size and online execution time). Same as the cases with N-AND, WAN latency is the dominant part of the online total execution time. In relatively small batch ($\leq 10^4$), all our protocols are faster than baseline

	pre-comp. time (ms)	online comp. time (ms)	# of comm. bits (bit)	data trans. time (ms)	# of comm. rounds	comm. latency (ms)	online total exec. time (ms)
	0.015	0.019	2	2.5×10^{-5}	1	40	40.0
2-AND	2.39	0.033	2×10^{3}	2.5×10^{-2}	1	40	40.1
	2439	19.4	2×10^{6}	25.0	1	40	84.4
	0.041	0.032	3	3.75×10^{-5}	1	40	40.0
3-AND	4.80	0.053	3×10^{3}	3.75×10^{-2}	1	40	40.1
	4899	33.1	3×10^{6}	37.5	1	40	110.6
	0.067	0.055	4	5.0×10^{-5}	1	40	40.1
4-AND	9.04	0.091	4×10^{3}	5.0×10^{-2}	1	40	40.1
	9383	62.8	4×10^{6}	50.0	1	40	152.8
	0.11	0.089	5	6.25×10^{-5}	1	40	40.1
5-AND	17.2	0.16	5×10^{3}	6.25×10^{-2}	1	40	40.2
	17700	111.7	5×10^{6}	62.5	1	40	214.2
	0.20	0.16	6	7.5×10^{-5}	1	40	40.2
6-AND	33.0	0.28	6×10^{3}	7.5×10^{-2}	1	40	40.4
	34059	203.0	6×10^{6}	75.0	1	40	318.0
	0.38	0.32	7	8.75×10^{-5}	1	40	40.3
7-AND	64.3	0.53	7×10^{3}	8.75×10^{-2}	1	40	40.6
	66123	370.8	7×10^{6}	87.5	1	40	498.3
	0.76	0.64	8	1.0×10^{-4}	1	40	40.6
8-AND	125.1	1.06	8×10^{3}	1.0×10^{-1}	1	40	41.2
	129553	700.7	8×10^{6}	100.0	1	40	840.7
	1.63	1.39	9	1.125×10^{-4}	1	40	41.4
9-AND	245.2	2.25	9×10^{3}	1.125×10^{-1}	1	40	42.4
	255847	1346	9×10^{6}	112.5	1	40	1498.5

Table 1. Evaluation on N-AND with 1(upper)/1000(middle)/1000000(lower) batch.

ones in the online total execution time since ours require fewer communication rounds. For example in Comparison with 1 batch, we need more online computation time than the baseline one. However, communication costs are smaller. As a result, our Comparison is 56.1% faster than baseline one (280.6ms $\rightarrow$ 122.1ms) in our WAN settings. As already mentioned, our protocols are not suitable for a (extremely) large batch since the computation cost is larger than baseline ones.

5.3 Application: Privacy-Preserving (Exact) Edit Distance

We implemented a privacy-preserving edit distance protocol using our protocols (Equality, B2A, and 3-Min). Unlike many previous works on approximate edit distance (e.g., [25]), here we consider the exact edit distance. We computed an edit distance between two length-L genome strings (S_0 and S_1) via standard dynamic programming (DP). It appears four characters in the strings; that is, A, T, G, and C. In DP-matrix, we fill the cell $x[i][j]$ by the following rule:

$$x[i][j] = \text{3-Min}([x[i-1][j]+1, x[i][j-1]+1, x[i-1][j-1]+e])$$

Here, $e = 0$ if the condition $S_0[i] = S_1[j]$ holds, and otherwise $e = 1$. We can compute e using Equality (two rounds) and B2A (one round). To reduce the total online execution time, we calculate the edit distance as follows:

1. We parallelly compute e for all cells and store them in advance. This procedure requires three communication rounds.
2. Diagonal cells in DP-matrix are independent with each other. Therefore, we can parallelly compute these cells $x[d][0], x[d-1][1], \cdots, x[0][d]$ (for each d) to reduce the communication rounds.

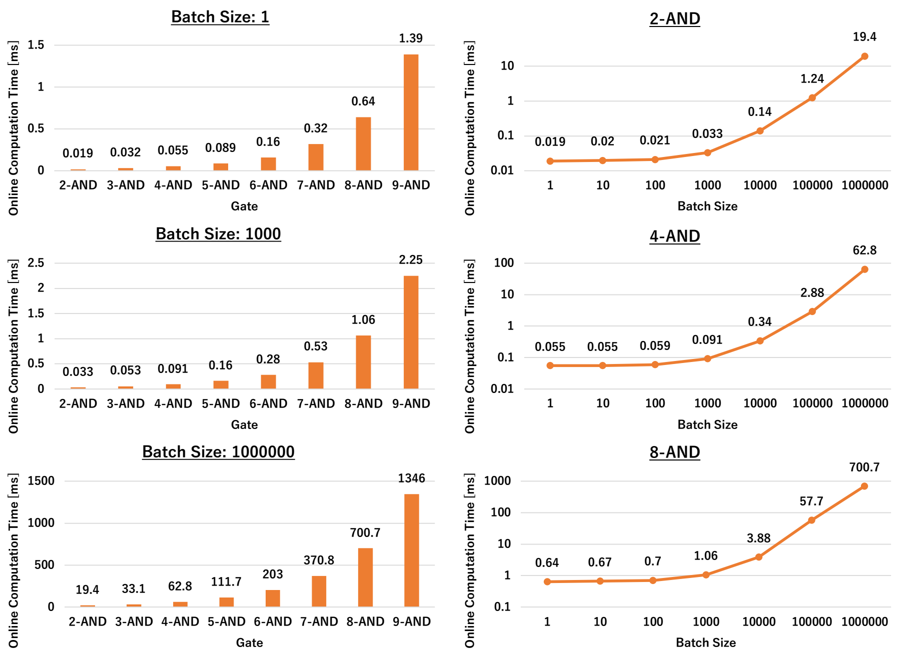

Fig. 1. Relations between N (fan-in number), batch size, and online computation time for N-AND: we show the relations between N and online computation time with 1/1000/1000000 batch (left), and show the relations between batch size and online computation time for 2/4/8-AND (right).

By applying the above techniques, we can compute exact edit distance for two length-L strings with $3+4(2L-1) = (8L-1)$ communication rounds. We used the arithmetic shares and protocols over $\mathbb{Z}_{2^{16}}$ in our experiments. The experimental results are as in Table 3: As we can see from the experimental results, most of the online total execution time is occupied by the communication latency.

Acknowledgements. This work was partly supported by JST CREST JP-MJCR19F6 and the Ministry of Internal Affairs and Communications Grant Number 182103105.

References

1. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 805–817 (2016)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. pp. 420–432 (1991)

	pre-comp. time (ms)	online comp. time (ms)	# of comm. bits (bit)	data trans. time (ms)	# of comm. rounds	comm. latency (ms)	online total exec. time (ms)
Equality	0.15	0.18	62	7.75×10^{-4}	5	200	200.2
(1 batch)	**0.76**	**0.52**	**38**	$\mathbf{4.75 \times 10^{-4}}$	**2**	**80**	**80.5**
Comparison	1.5	0.54	970	1.21×10^{-2}	7	280	280.6
(1 batch)	**3.9**	**2.1**	**712**	$\mathbf{8.9 \times 10^{-3}}$	**3**	**120**	**122.1**
3-Max	3.1	1.2	2196	2.75×10^{-2}	18	720	721.2
(1 batch)	**9.7**	**2.3**	**3960**	$\mathbf{4.95 \times 10^{-2}}$	**4**	**160**	**162.3**
Equality	74.7	0.61	62×10^3	0.78	5	200	201.4
(10^3 batch)	**500.5**	**1.1**	$\mathbf{38 \times 10^3}$	**0.48**	**2**	**80**	**80.9**
Comparison	1398	8.25	970×10^3	12.1	7	280	300.4
(10^3 batch)	**2745**	**11.6**	$\mathbf{712 \times 10^3}$	**8.9**	**3**	**120**	**140.5**
3-Max	2891	17.5	2196×10^3	27.5	18	720	765.0
(10^3 batch)	**8635**	**36.3**	$\mathbf{3960 \times 10^3}$	**49.5**	**4**	**160**	**245.8**
Equality	77574	761.4	62×10^6	780	5	200	1741
(10^6 batch)	**500617**	**1233**	$\mathbf{38 \times 10^6}$	**480**	**2**	**80**	**1793**
Comparison	1445847	13895	970×10^6	12100	7	280	26275
(10^60 batch)	**2799437**	**20748**	$\mathbf{712 \times 10^6}$	**8900**	**3**	**120**	**29768**
3-Max	2956155	28252	2196×10^6	27500	18	720	56472
(10^6 batch)	**8571664**	**69935**	$\mathbf{3960 \times 10^6}$	**49500**	**4**	**160**	**119595**

Table 2. Evaluation of our protocols over $\mathbb{Z}_{2^{32}}$ for $1/10^3/10^6$ batches. In each cell, we show our experimental results on the baseline (upper) and ours (lower).

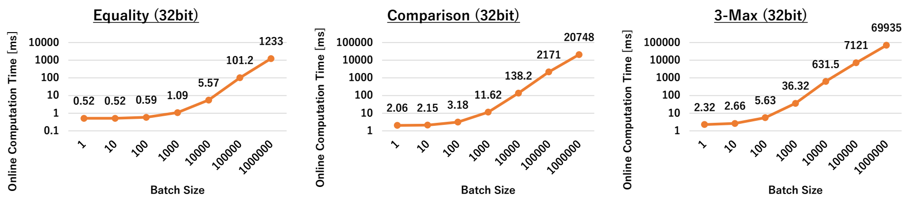

Fig. 2. Relations between batch size and online computation/execution time of the protocols over $\mathbb{Z}_{2^{32}}$.

3. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 578–590 (2016)
4. Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multi-party computation for data mining applications. Int. J. Inf. Sec. **11**(6), 403–418 (2012)
5. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015)
6. Byali, M., Joseph, A., Patra, A., Ravi, D.: Fast secure computation for small population over the internet. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 677–694 (2018)
7. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. pp. 34–64

string length	**pre-comp. time (s)**	**online comp. time (s)**	**data trans. time (s)**	**comm. latency (s)**	**online total exec. time (s)**
4	0.04	0.01	4.0×10^{-4}	1.24	1.25
8	0.14	0.02	1.4×10^{-3}	2.52	2.54
16	0.57	0.04	5.7×10^{-3}	5.08	5.13
32	2.2	0.10	2.3×10^{-2}	10.2	10.3
64	8.1	0.22	9.2×10^{-2}	20.4	20.7
128	33.4	0.54	3.7×10^{-1}	40.9	41.8
256	135.7	1.5	1.5	84.9	84.9
512	534.1	4.8	5.9	163.8	174.5
1024	2262	16.0	23.4	327.6	367.0

Table 3. Experimental results of privacy-preserving exact edit distance with 2^{ℓ}-length two strings ($\ell = [2, \cdots, 10]$).

(2018)
8. Couteau, G., Peters, T., Pointcheval, D.: Encryption switching protocols. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. pp. 308–338 (2016)
9. 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, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings. pp. 285–304 (2006)
10. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015)
11. Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017 (2017)
12. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. pp. 201–210 (2016)
13. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
14. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 218–229 (1987)
15. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 1651–1669 (2018)
16. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: How to combine homomorphic encryption and garbled circuits - improved circuits and computing the minimum distance efficiently. In: International Workshop on Signal Processing in the EncryptEd Domain (SPEED'09) (2009)

17. Liu, X., Deng, R.H., Choo, K.R., Weng, J.: An efficient privacy-preserving outsourced calculation toolkit with multiple keys. IEEE Trans. Information Forensics and Security **11**(11), 2401–2414 (2016)
18. Mohassel, P., Orobets, O., Riva, B.: Efficient server-aided 2pc for mobile phones. PoPETs **2016**(2), 82–99 (2016)
19. Mohassel, P., Rindal, P.: Aby^3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 35–52 (2018)
20. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 591–602 (2015)
21. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 19–38 (2017)
22. Morita, H., Attrapadung, N., Teruya, T., Ohata, S., Nuida, K., Hanaoka, G.: Constant-round client-aided secure comparison protocol. In: Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II. pp. 395–415 (2018)
23. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings. pp. 343–360 (2007)
24. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018. pp. 707–721 (2018)
25. Schneider, T., Tkachenko, O.: EPISODE: efficient privacy-preserving similar sequence queries on outsourced genomic databases. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019. pp. 315–327 (2019)
26. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986. pp. 162–167 (1986)
27. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. pp. 220–250 (2015)
28. Zhu, R., Cassel, D., Sabry, A., Huang, Y.: NANOPI: extreme-scale actively-secure multi-party computation. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 862–879 (2018)