Secure Computation for G-Module and its Applications

Qizhi Zhang [*]
Ant Group
qizhi.zqz@antgroup.com

BingSheng Zhang [*]
Zhejiang University
bingsheng@bingsheng@zju.edu.cn

Lichun Li
Ant Group
lichun.llc@antgroup.com

Shan Yin
Ant Group
yinshan.ys@antgroup.com

Juanjuan Sun
School of mathematical sciences, Tongji university
sunjuan@tongji.edu.cn

Abstract

Secure computation enables two or more parties to jointly evaluate a function without revealing to each other their private input. $\mathbb{G}$-module is an abelian group $\mathbb{M}$, where the group $\mathbb{G}$ acts compatibly with the abelian group structure on $\mathbb{M}$. In this work, we present several secure computation protocols for $\mathbb{G}$-module operations in the online/offline mode. We then show how to instantiate those protocols to implement many widely used secure computation primitives in privacy-preserving machine learning and data mining, such as oblivious cyclic shift, one-round shared OT, oblivious permutation, oblivious shuffle, secure comparison, oblivious selection, DReLU, and ReLU, etc. All the proposed protocols are constant-round, and they are 2X - 10X more efficient than the-state-of-the-art constant-round protocols in terms of communication complexity.

1 Introduction

In a secure computation (a.k.a. multi-party computation) protocol, two or more parties collaboratively evaluate a function and receive its output without revealing their private input to the others. A number of general-purpose secure computation solutions [Yao86, GMW87, BLW08, DPSZ12] have been proposed in the past decades. Recently, an increasing number of practical multi-party computation platforms are tailor-made for privacy-preserving machine-learning and/or data-mining, such as Delphi [MLS$^+$20], SecureML [MZ17], SecureNN [WGC19], Chameleon [RWT$^+$18], and CrypTFlow [KRC$^+$20], etc. Typically, those protocols are deployed in client-server mode [HM00, DI05] with three MPC parties or two MPC parties with an semi-honest assistant third-party (a.k.a. server-aid mode, .e.g., [MOR16, RWT$^+$18]).

In this work, we first investigate the secure computation problem for $\mathbb{G}$-module operations. In mathematics, given a group $\mathbb{G}$, $\mathbb{G}$-module is an abelian group $\mathbb{M}$, on which $\mathbb{G}$ acts compatibly with the abelian group structure on $\mathbb{M}$. We present three secure computation protocols for $\mathbb{G}$-module under different settings. Our protocols are designed in the pre-processing (a.k.a. online-offline) mode, e.g. [DPSZ12], where in the offline phase, the secure computation parties prepare some correlated randomness, e.g. Beaver triples, using offline protocols, say homomorphic encryption or the help of a semi-honest third party [RWT$^+$18]. We then show how to instantiate our protocols to realize some useful concrete secure computation primitives, such as secure comparison and oblivious shuffle. As one important and classical problem, secure comparison (a.k.a. Yao's Millionaires' problem [Yao86]), in which two parties find out whether $x < y$ for private input x, y without disclosing them, receives the most attention [KSS09, GSV07]. Many constructions are proposed in the literature, e.g., [Cou18], and to the best of our knowledge, function secret sharing (FSS) based secure comparison [BGI16] is the most efficient constant-round protocol w.r.t. online communication. Secure comparison is one of the fundamental building blocks in privacy-preserving machine learning. Typically, ReLU and DReLU which internally needs secure comparison, are commonly used in the active layers of deep learning. They are defined by

$$\mathrm{ReLU}(x) := \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{DReLU}(x) := \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

For instance, platforms like [MLS$^+$20, MR18, MZ17, WGC19, RWT$^+$18, KRC$^+$20], all implement and use ReLU and DReLU in certain tasks. In two's complement representation, $\mathrm{DReLU}(x) = -\mathrm{MSB}(x)$, where MSB means Most Significant Bit.

Oblivious shuffle is another handy protocol that is widely used in privacy-preserving database manipulation [LWZ11]. In an oblivious shuffle protocol, the MPC parties hold a shared array $([a_0], \ldots, [a_{n-1}])$, and they want to jointly shuffle to array such that none of the MPC parties know the exactly permutation of the shuffled array. Conventionally, oblivious

[1]These authors contributed equally to this work.

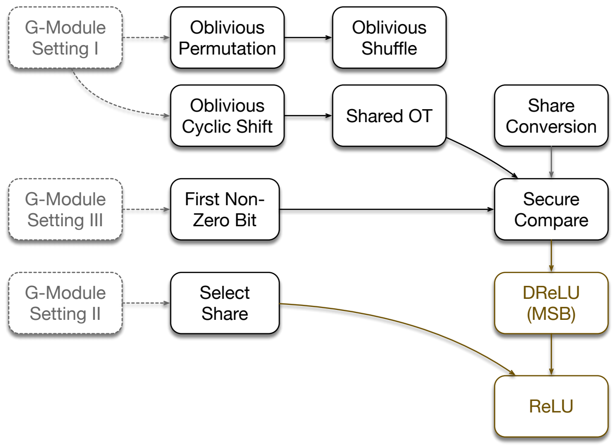

Figure 1: The roadmap of our protocols

sorting networks, e.g., butterfly network, are used to permute the array; however, it would resulting an $O(\log n)$-round protocol. In [LWZ11], the authors proposed a novel oblivious shuffle protocol, where the concept is similar to a mix-net method. Take the 3-party protocol as an example, in LWZ oblivious shuffle [LWZ11], the parties first re-share the array such that only P_2 and P_3 hold the (additive) shares of the array, i.e., P_1's shares are 0. Then, P_2 and P_3 can permute their shares of the array according to the same random permutation $\sigma_1 \in S_n$ that is only known to P_2 and P_3. Note that P_1 is oblivious to σ_1. After that, the parties re-share the permuted array such that only P_3 and P_1 hold the shares; similarly, P_3 and P_1 jointly permute the array according to a random permutation σ_2 that is unknown to P_2; finally, P_1 and P_2 jointly permute the array according to σ_3 that is unknown to P_3. The array has been permuted three times. Since each party is oblivious to at least one permutation, the overall permutation is unknown to all the parties. Although it is a constant-round efficient protocol, it cannot be naturally ported to the two-party setting. This is because each party has to shuffle the array alone without seeing the data in plaintext; usually, public-key cryptographic operations are needed. Very recently, Chase *et al.* [CGP20] proposed a 3-round two-party oblivious shuffle protocol only using symmetric cryptographic operations. In this work, we show the first constant-round oblivious shuffle protocol for two-party computation using additive shares only.

Our Contributions. In this paper, we focus on secure computation for $\mathbb{G}$-module operations, and then gradually develop our oblivious cyclic shift, one-round shared OT, oblivious permutation, oblivious shuffle, secure comparison, oblivious selection, DReLu, and ReLu protocols. Our main contributions can be summarized as follows:

- We introduce a new notion called *secure computation for $\mathbb{G}$-module*, and formalize its functionally $\mathcal{F}_{\mathsf{GM}}$ in the well-known Universal Composibility framework [Can01]. We then present three secure computation protocols that UC-realize $\mathcal{F}_{\mathsf{GM}}$ in three different settings. The communication complexity of our protocol is shown in Table. 1.1. Those protocols may be of independent interest, as they can be further instantiated to many other efficient handy protocols besides the examples mentioned in this paper.

- As an application of secure computation for $\mathbb{G}$-module (setting III), we construct a novel secure comparison protocol $\Pi^{p,n}_{\mathsf{SC}}$. The protocol is designed in the online/offline mode. As depicted in Table. 1.1, to privately compare $x, y \in \mathbb{Z}_{2^n}$, the overall communication (offline + online) is only $(4n+3)\log p + 3n + \log(n+1) + 2$, where $p \geq n+3$ is a prime number. We note that the communication complexity is approximately 10% of that of the state-of-the-art constant-round protocol [BGI16].

- We present a DReLU protocol $\Pi^{p,n}_{\mathsf{DReLU}}$. Compared to the most efficient constand-round DReLU protocol in the literature, CrypTFlow [KRC+20], the online and overall (online + offline) communication of our protocol are approximately 31% and 55% of that of CrypTFlow, respectively. Moreover, the round complexity of our protocol is also much less than that of CrypTFlow. The exact numbers can be found in Table. 1.1.

- As an application of the presented secure computation for $\mathbb{G}$-module (setting I), we show how to realize two-party oblivious permutation protocol $\Pi^{n,\ell}_{\mathsf{O\text{-}Perm}}$, where P_1 inputs permutation $\sigma \in S_n$, where S_n is the symmetric group of degree n, and P_2 inputs $x := (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n_{2^\ell}$, where $x_i \in \mathbb{Z}_{2^\ell}$; after the protocol execution, P_1 and P_2 obtain shares of $\sigma(x)$ in $\mathbb{Z}^n_{2^\ell}$, where $\sigma(x)$ is defined by $\sigma(x)_i = x_{\sigma^{-1}(i)}$. We then construct our two-party obvious shuffle protocol $\Pi^{n,\ell}_{\mathsf{O\text{-}shuffle}}$ using protocol $\Pi^{n,\ell}_{\mathsf{O\text{-}Perm}}$ twice. To the best of our knowledge, this is the first constant-round two-party oblivious shuffle protocol without using any cryptographic operations. Comparing with the state-of-the-art [CGP20], the communication of our protocol is two orders of magnitude less than [CGP20] for big n.

- As an application of secure computation for $\mathbb{G}$-module (setting II), we construct an oblivious selection protocol $\Pi^N_{\mathsf{O\text{-}select}}$. In an oblivious selection protocol, P_1 and P_2 jointly hold shares of $b \in \{0,1\}$ and $x_0, x_1 \in \mathbb{Z}_N$, and they will obtain shares of x_b obliviously after the protocol execution. Compared to the oblivious selection protocol in SecureNN [WGC19], the online and overall (online + offline) communication of our protocol are approximately 41% and 62% of that of SecureNN, respectively. In addition, the round complexity of our protocol is half of SecureNN. The exact numbers can be found in Table. 5.2.

- Finally, on top of $\Pi^{p,n}_{\mathsf{DReLU}}$ and $\Pi^N_{\mathsf{O\text{-}select}}$, we further develop the ReLU protocol Π_{ReLU}. Compare to the ReLU protocols in SecureNN [WGC19] and CrypTFlow [KRC+20], The online communication and overall (online+offline) communication of our ReLU protocol are approximately 31% and 55% of that of the state-of-the-arts. Besides, the round complexity of our protocol is much less. The exact numbers can be found in Table. 7.1.

Our Techniques. The protocol dependency is shown in Fig. 1. We first formally define the $\mathbb{G}$-module UC functionality $\mathcal{F}_{\mathsf{GM}}$ in three different settings. We then UC-realize $\mathcal{F}_{\mathsf{GM}}$ with three protocols in the $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$-hybrid model. We then show how to instantiate $\mathbb{G}$-module (setting III) to develop a *first non-zero bit* protocol $\Pi_{\mathsf{FNZ}}^{p,n}$. In the protocol $\Pi_{\mathsf{FNZ}}^{p,n}$, P_1 and P_2 jointly holds the shares (over $\mathbb{Z}_p$) of a non-zero binary vector, and they obtain the share (over $\mathbb{Z}_n$) of the index of the first non-zero bit. We then show that the $(1,n)$-Shared OT protocol can be instantiated from $\mathbb{G}$-module (setting I). In the $(1,n)$-Shared OT protocol, P_1 inputs messages $(m_0,\ldots,m_{n-1})$, $m_i \in \mathbb{Z}_{2^\ell}$, and P_2 inputs an index $i \in \mathbb{Z}_n$. After the protocol execution, P_1 and P_2 obtain shares of m_i. For completeness, we also recap the share conversion protocol from [Cou18]. On top of $\Pi_{\mathsf{FNZ}}^{p,n}$, $(1,n)$-Shared OT, and share conversion protocols, we develop our constant-round secure comparison protocol $\Pi_{\mathsf{SC}}^{p,n}$. As compared in Table. 6.1, our $\Pi_{\mathsf{SC}}^{p,n}$ protocol is more efficient than the state-of-art.

We then develop our DReLU protocol $\Pi_{\mathsf{DReLU}}^{p,n}$ using the $\Pi_{\mathsf{SC}}^{p,n}$ protocol as a building block. As mentioned before, our DReLU protocol is more efficient than that in SecureNN [WGC19] and CrypTFlow [KRC+20] (cf. Table. 6.2). We instantiate $\mathbb{G}$-module (setting I) to realize the first constant-round two-party oblivious shuffle protocol without using any cryptographic operations. Meanwhile, we instantiate $\mathbb{G}$-module (setting II) to realize the oblivious selection protocol. Comparing to the oblivious selection protocol in SecureNN [WGC19] (It is called select share in SecureNN), our protocol is more efficient in all aspect (cf. Table. 5.2). Finally, we develop the ReLU protocol Π_{ReLU}^{n} using the aforementioned protocols as building blocks. Our ReLU protocol is the more efficient than the state-of-art (cf. Table. 7.1).

2 Preliminaries

Notation. Throughout this paper, we use the following notations and terminologies. Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\emptyset$ denote empty set. When A is a set, $|A|$ stands for the cardinality of A in terms of the number of entries. When S is a set, $s \leftarrow S$ stands for sampling s uniformly at random from S. When A is a randomised algorithm, $y \leftarrow A(x)$ stands for running A on input x with a fresh random coin r. When needed, we denote $y := A(x;r)$ as running A on input x with the explicit random coin r. When A is an abelian group, a is an element in A, we denote $[a]$ as the additive share representation of a, i.e., $[a] := (a_1, a_2)$ such that $a_1 + a_2 = a$, where P_1 hold a_1 and P_2 hold a_2. We abbreviate *probabilistic polynomial time* as PPT. Let $\mathrm{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ be a polynomially-bounded function and negligible function, respectively.

$\mathbb{G}$-Module. Let $(\mathbb{G},\cdot)$ be a finite group with identity element $\mathbf{1}$ and $(\mathbb{M},+)$ be a finite abelian group, we call $\mathbb{M}$ a $\mathbb{G}$-module [HS97] if there is a map:

$$\begin{array}{ccccc} \mathbb{G} & \times & \mathbb{M} & \longrightarrow & \mathbb{M} \\ (g & , & h) & \longmapsto & g\cdot h \end{array}$$

satisfying the following properties:

- $\forall h \in \mathbb{M}$: $\mathbf{1}\cdot h = h$;
- $\forall g_1, g_2 \in \mathbb{G}$, $\forall h \in \mathbb{M}$: $(g_1 \cdot g_2)\cdot h = g_1 \cdot (g_2 \cdot h)$;
- $\forall g \in \mathbb{G}$, $\forall h_1, h_2 \in \mathbb{M}$: $g\cdot(h_1 + h_2) = g\cdot h_1 + g \cdot h_2$.

The image of (g,h) under above map is usually written by $g\cdot h$ or gh, and is called the image of h under the action of g. For a fixed h in $\mathbb{M}$, the set $\{gh : g \in \mathbb{G}\}$ is called the orbit of h under the action of $\mathbb{G}$, or the $\mathbb{G}$-orbit of h , and written as $\mathbb{G}h$.

Under the action of $\mathbb{G}$, $\mathbb{M}$ has $\mathbb{G}$-orbit decomposition, denoted as $\mathbb{M} = \coprod_i \mathbb{M}_i$, where $\mathbb{M}_i := \mathbb{G}\cdot h_i$ can be generated by any element $h_i \in \mathbb{M}_i$ under the group action of $\mathbb{G}$, and $\mathbb{M}_i \cap \mathbb{M}_j = \emptyset$ for $i \neq j$.

Semi-direct product. Let N be an abelian group and H be a group, ϕ be a homomorphism $\phi : H \longrightarrow Aut(N)$, we can construct a new group $H \ltimes N$, called the semidirect product of H and N with respect to ϕ ([SEM03].pp. 75–76), defined as follows:

The underlying set is the Cartesian product $H \times N$. The group operation is determined by the homomorphism ϕ:

$$\begin{array}{ccccc} H\ltimes N & \times & H\ltimes N & \longrightarrow & H\ltimes N \\ (h_1,n_1) & , & (h_2,n_2) & \longmapsto & (h_1h_2, n_1\phi_{h_1}(n_2)) \end{array}$$

for $n_1, n_2 \in N$ and $h_1, h_2 \in H$.

Universal Composbility. Following Canetti's framework [Can01], a protocol is represented as interactive Turing machines (ITMs), each of which represents the program to be run by a participant. Protocols that securely carry out a given task are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. The parties have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-world model*), the ideal process, and the notion of protocol emulation.

Table 1.1: Communication complexity and round complexity of our protocols. (The offline functionalities are realized by a semi-honest server. $\mathbb{G}$ is a group, $\mathbb{M}$ is a $\mathbb{G}$-module. $|\mathbb{G}|$ and $|\mathbb{M}|$ denote the cardinalities of $\mathbb{G}$ and $\mathbb{M}$, respectively. The messages in $(1,n)$ Shared OT $\Pi^{n,l}_{\mathsf{S-OT}}$ are from $\mathbb{Z}_{2^\ell}$. $\Pi^{2,p}_{\mathsf{convert}}$ stands for the share conversion protocol from $\mathbb{Z}_2$ to $\mathbb{Z}_p$. The first non-zero bit protocol $\Pi^{p,n}_{\mathsf{FNZ}}$ takes input from shares of a size-n binary vector over $\mathbb{Z}_p$, where $p \geq n+2$ is a prime number. The secure comparison protocol Π^{n}_{SC} takes input $x, y \in \mathbb{Z}_{2^n}$.)

Protocol	offline com.	online com.	online round	total com.
$\Pi^{\mathrm{I}}_{\mathsf{GM}}$ (cf. Sec. 4)	$\log\lvert\mathbb{M}\rvert$	$\log\lvert\mathbb{G}\rvert+\log\lvert\mathbb{M}\rvert$	1	$\log\lvert\mathbb{M}\rvert+2\log\lvert\mathbb{M}\rvert$
$\Pi^{\mathrm{II}}_{\mathsf{GM}}$ (cf. Sec. 4)	$\log\lvert\mathbb{M}\rvert$	$2\log\lvert\mathbb{G}\rvert+2\log\lvert\mathbb{M}\rvert$	1	$2\log\lvert\mathbb{G}\rvert+3\log\lvert\mathbb{M}\rvert$
$\Pi^{\mathrm{III}}_{\mathsf{GM}}$ (cf. Sec. 4)	$\log\lvert\mathbb{M}\rvert$	$2\log\lvert\mathbb{M}\rvert$	2	$3\log\lvert\mathbb{M}\rvert$
$\Pi^{n,l}_{\mathsf{O-Shift}}$ (cf. Sec. 5)	$n\ell$	$n\ell+\log n$	1	$2n\ell+\log n$
$\Pi^{n,l}_{\mathsf{S-OT}}$ (cf. Sec. 5)	$n\ell$	$n\ell+\log n$	1	$2n\ell+\log n$
$\Pi^{n,l}_{\mathsf{O-Perm}}$ (cf. Sec. 5)	$n\ell$	$n\ell+\log n!$	1	$2n\ell+\log n!$
$\Pi^{n,l}_{\mathsf{O-Shuffle}}$ (cf. Sec. 5)	$2n\ell$	$2n\ell+2\log n!$	2	$4n\ell+2\log n!$
$\Pi^{n,l}_{\mathsf{OT2}}$ (cf. Appendix. D)	ℓ	$n\ell+\log n$	2	$(n+1)\ell+\log n$
$\Pi^{n,l}_{\mathsf{S-OT2}}$ (cf. Appendix. D)	ℓ	$n\ell+\log n$	2	$(n+1)\ell+\log n$
$\Pi^{p,n}_{\mathsf{FNZ}}$ (cf. Sec. 5)	$n\log p^{*}$	$2n\log p$	2	$3n\log p$
$\Pi^{p,n}_{\mathsf{SC}}$ (cf. Sec. 5)	$(2n+1)\log p^{\dagger} +1$	$2(n+1)\log p+3n +\log(n+1)+1$	4	$(4n+3)\log p+3n +\log(n+1)+2$
$\Pi^{p,n}_{\mathsf{DReLU}}$ (cf. Sec. 5)	$(2n-1)\log p^{*} +1$	$2n\log p+ 3n+\log n-2$	4	$(4n-1)\log p+ 3n+\log n-1$
Π^{N}_{OS} for odd N (cf. Sec. 5)	$\log N$	$2(1+\log N)$	1	$2+3\log N$
Π^{N}_{OS} for even N (cf. Sec. 5)	$\log N+1$	$2(2+\log N)$	1	$5+3\log N$
$\Pi^{p,n}_{\mathsf{ReLU}}$ (cf. Sec. 5)	$(2n-1)\log p^{*} +n+2$	$2n\log p+ 5n+\log n+2$	5	$(4n-1)\log p+ 6n+\log n+4$

[*] $p \geq n+2$ is a prime number.
[†] $p \geq n+3$ is a prime number.

3 Security Model

Our security model is based on the Universal Composibility (UC) framework, which lays down a solid foundation for designing and analyzing protocols secure against attacks in an arbitrary *network* execution environment (therefore it is also known as *network aware security model*). Roughly speaking, in the UC framework, protocols are carried out over multiple interconnected machines; to capture attacks, a network adversary $\mathcal{A}$ is introduced, which is allowed to corrupt some machines (i.e., have the full control of all physical parts of some machines); in addition, $\mathcal{A}$ is allowed to partially control the communication tapes of all uncorrupted machines, that is, it sees all the messages sent from and to the uncorrupted machines and controls the sequence in which they are delivered. Then, a protocol ρ is a UC-secure implementation of a functionality $\mathcal{F}$, if it satisfies that for every network adversary $\mathcal{A}$ attacking an execution of ρ, there is another adversary $\mathcal{S}$—known as the simulator—attacking the ideal process that uses $\mathcal{F}$ (by corrupting the same set of machines), such that, the executions of ρ with $\mathcal{A}$ and that of $\mathcal{F}$ with $\mathcal{S}$ makes no difference to any network execution environment.

3.1 The ideal world execution

In the ideal world, P_1 and P_2 only communicate with an ideal functionality $\mathcal{F}_{\mathsf{GM}}$ during the execution. As depicted in Fig. 2, $\mathbb{G}$-module has three settings. In setting I, party P_1 inputs $g \in \mathbb{G}$ and party P_2 inputs $h \in \mathbb{M}$ to $\mathcal{F}_{\mathsf{GM}}$. Upon receiving the input, $\mathcal{F}_{\mathsf{GM}}$ computes $s := g \cdot h$. It then picks random $s_1 \leftarrow \mathbb{M}$ and sets $s_2 := s - s_1$. After that, $\mathcal{F}_{\mathsf{GM}}$ sends the output s_1, s_2 to P_1 and P_2, respectively. In setting II, party P_1 inputs $g_1 \in \mathbb{G}, h_1 \in \mathbb{M}$ and party P_2 inputs $g_2 \in \mathbb{G}, h_2 \in \mathbb{M}$ to $\mathcal{F}_{\mathsf{GM}}$. Upon receiving the input, $\mathcal{F}_{\mathsf{GM}}$ computes $s := g_1 \cdot g_2 \cdot (h_1 + h_2)$. It then pick random $s_1 \leftarrow \mathbb{M}$ and sets $s_2 := s - s_1$. After that $\mathcal{F}_{\mathsf{GM}}$ sends output s_1, s_2 to P_1 and P_2, respectively. In setting III, party P_1 inputs $h_1 \in \mathbb{M}$ and party P_2 inputs $h_2 \in \mathbb{M}$ to $\mathcal{F}_{\mathsf{GM}}$. Upon receiving the input, $\mathcal{F}_{\mathsf{GM}}$ computes $h' := h_1 + h_2$. It then picks random $g \leftarrow \mathbb{G}$ and sets $h := g^{-1} \cdot h'$. After that, $\mathcal{F}_{\mathsf{GM}}$ sends g to P_1 and h to P_2;

3.2 The real/hybrid world execution

The real/hybrid world protocol Π_{GM} uses the offline functionality $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}$. As depicted in Fig. 3, the functionality $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}$ prepares correlated randomness for P_1 and P_2 to facilitate the $\mathbb{G}$-module protocol. More specifically, It consists of the following three different settings. In setting I, $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}$ picks random $u \leftarrow \mathbb{G}$ and $v, w_1 \leftarrow \mathbb{M}$. It then sets $w_2 := u \cdot v - w_1$.

Ideal functionality $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$

This functionality interacts with players $\mathcal{P} := \{P_1, P_2\}$ and the ideal adversary $\mathcal{S}$. It is parameterized with finite groups $\mathbb{G}, \mathbb{M}$.

Setting I:

- Upon receiving $(\textsc{Input-I}, \mathsf{sid}, g)$ from P_1:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_1)$ to the adversary $\mathcal{S}$;
- Upon receiving $(\textsc{Input-I}, \mathsf{sid}, h)$ from P_2:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_2)$ to the adversary $\mathcal{S}$;
- Once both parties have sent their input:
  - Compute $s := g \cdot h$;
  - Pick random $s_1 \leftarrow \mathbb{M}$ and set $s_2 := s - s_1$;
  - Send $(\textsc{Output-I}, \mathsf{sid}, s_1)$ to P_1 and $(\textsc{Output-I}, \mathsf{sid}, s_2)$ to P_2;

Setting II:

- Upon receiving $(\textsc{Input-II}, \mathsf{sid}, g_1, h_1)$ from P_1:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_1)$ to the adversary $\mathcal{S}$;
- Upon receiving $(\textsc{Input-II}, \mathsf{sid}, g_2, h_2)$ from P_2:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_2)$ to the adversary $\mathcal{S}$;
- Once both parties have sent their input:
  - Compute $s := g_1 \cdot g_2 \cdot (h_1 + h_2)$;
  - Pick random $s_1 \leftarrow \mathbb{M}$ and set $s_2 := s - s_1$;
  - Send $(\textsc{Output-II}, \mathsf{sid}, s_1)$ to P_1 and $(\textsc{Output-II}, \mathsf{sid}, s_2)$ to P_2

Setting III:

- Upon receiving $(\textsc{Input-III}, \mathsf{sid}, h_1)$ from P_1:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_1)$ to the adversary $\mathcal{S}$;
- Upon receiving $(\textsc{Input-III}, \mathsf{sid}, h_2)$ from P_2:
  - Send $(\textsc{InputNotify}, \mathsf{sid}, P_2)$ to the adversary $\mathcal{S}$;
- Once both parties have sent their input:
  - Compute $h' := h_1 + h_2$;
  - Pick random $g \leftarrow \mathbb{G}$ and set $h := g^{-1} \cdot h'$;
  - Send $(\textsc{Output-III}, \mathsf{sid}, g)$ to P_1 and $(\textsc{Output-III}, \mathsf{sid}, h)$ to P_2

Figure 2: Ideal functionality $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ for $\mathbb{G}$-module

After that, $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$ sends (u, w_1) to P_1 and (v, w_2) to P_2. In setting II, $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$ picks random $u_1, u_2 \leftarrow \mathbb{G}$ and $v_1, v_2, w_1 \leftarrow \mathbb{M}$. It then sets $w_2 := u_1 \cdot u_2 \cdot (v_1 + v_2) - w_1$. After that, $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$ sends (u_1, v_1, w_1) to P_1 and (u_2, v_2, w_2) to P_2. In setting III, $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$ picks random $u \leftarrow \mathbb{G}$ and $v, w_1 \leftarrow \mathbb{M}$. It then sets $w_2 := u \cdot v - w_1$. After that, $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}$ sends (u, w_1) to P_1 and (v, w_2) to P_2.

4 Secure Computation for $\mathbb{G}$-module

In this section, we provide three secure computation protocols for $\mathbb{G}$-module with different settings. There are all in the preprocessing model. In the following, let $\mathbb{G}$ be a finite group, and $\mathbb{M}$ be a finite $\mathbb{G}$-module.

2PC for $\mathbb{G}$-module (Setting I). In this setting, party P_1 has private input $g \in \mathbb{G}$ and party P_2 has private input $h \in \mathbb{M}$. After

Offline functionality $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$

This functionality interacts with players $\mathcal{P} := \{P_1, P_2\}$ and the adversary $\mathcal{A}$. It is parameterized with finite groups $\mathbb{G}, \mathbb{M}$.

Setting I:

- Upon receiving $(\textsc{Prepare-I}, \mathsf{sid})$ from both parties P_1 and P_2:
  - Pick random $u \leftarrow \mathbb{G}$ and $v, w_1 \leftarrow \mathbb{M}$;
  - Set $w_2 := u \cdot v - w_1$;
  - Send $(\textsc{Prepare-I}, \mathsf{sid}, u, w_1)$ to P_1 and $(\textsc{Prepare-I}, \mathsf{sid}, v, w_2)$ to P_2;

Setting II:

- Upon receiving $(\textsc{Prepare-II}, \mathsf{sid})$ from both parties P_1 and P_2:
  - Pick random $u_1, u_2 \leftarrow \mathbb{G}$ and $v_1, v_2, w_1 \leftarrow \mathbb{M}$;
  - Set $w_2 := u_1 \cdot u_2 \cdot (v_1 + v_2) - w_1$;
  - Send $(\textsc{Prepare-II}, \mathsf{sid}, u_1, v_1, w_1)$ to P_1 and $(\textsc{Prepare-II}, \mathsf{sid}, u_2, v_2, w_2)$ to P_2;

Setting III:

- Upon receiving $(\textsc{Prepare-III}, \mathsf{sid})$ from both parties P_1 and P_2:
  - Pick random $u \leftarrow \mathbb{G}$ and $v, w_1 \leftarrow \mathbb{M}$;
  - Set $w_2 := u \cdot v - w_1$;
  - Send $(\textsc{Prepare-III}, \mathsf{sid}, u, w_1)$ to P_1 and $(\textsc{Prepare-III}, \mathsf{sid}, v, w_2)$ to P_2;

Figure 3: Offline functionality $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ for $\mathbb{G}$-module

the protocol execution, P_1 and P_2 output shares of $g \cdot h \in \mathbb{M}$. The protocol $\Pi_{\mathsf{GM}}^{\mathrm{I}}[\mathbb{G},\mathbb{M}]$ is described in Fig. 4.

Theorem 1 *Let $\mathbb{G}$ be a finite group, and $\mathbb{M}$ be a finite $\mathbb{G}$-module. The protocol $\Pi_{\mathsf{GM}}^{I}[\mathbb{G},\mathbb{M}]$ described in Fig. 4 UC-realizes $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ described in Fig. 2 in the $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$-hybrid model against semi-honest PPT adversaries with static corruption.*

Proof: See Appendix. A.

Efficiency. When $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ is realized by a semi-honest server (a.k.a. server-aid mode, e.g. [RWT+18]), the offline communication of protocol $\Pi_{\mathsf{GM}}^{\mathrm{I}}[\mathbb{G},\mathbb{M}]$ is $\log|\mathbb{G}| + 3\log|\mathbb{M}|$ bits. The communication can be further reduced to $\log|\mathbb{M}|$ bits using PRF (cf. below). The online communication of protocol $\Pi_{\mathsf{GM}}^{\mathrm{I}}[\mathbb{G},\mathbb{M}]$ is $\log|\mathbb{G}| + \log|\mathbb{M}|$ bits, and in one round. Hence, the total communication of protocol $\Pi_{\mathsf{GM}}^{\mathrm{I}}[\mathbb{G},\mathbb{M}]$ is $\log|\mathbb{G}| + 2\log|\mathbb{M}|$ bits.

More specifically, P_1, P_2 can get (u, w_1) and (v, w_2) as follows: The assisting server S, P_1 and P_2 share a PRF:

$$\begin{array}{cccccc} F: K & \times & \mathbb{N} & \longrightarrow & \mathbb{G} \times \mathbb{M} \\ (k & , & t) & \longmapsto & F_k(t) \end{array}$$

where K is the key space. Let S and P_1 share a key k_1, S and P_2 share a key k_2. In the offline phase, for a counter $t \in \mathbb{N}$, S uses $k_1 \in K$ as a PRF key to generate $(u_t, w_{t,1}) \leftarrow F_{k_1}(t)$ and uses $k_2 \in K$ as a PRF key to generate $(\tilde{u}_t, v_t) \leftarrow F_{k_2}(t)$, then computes $w_{t,2} := u_t v_t - w_{t,1}$ and finally sends it to P_2.

Protocol $\Pi^{\mathrm{I}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$

Protocol:

- Upon receiving $(\textsc{Input-I},\mathsf{sid},g)$ from the environment $\mathcal{Z}$, the party P_1 does:
  - Send $(\textsc{Prepare-I},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-I},\mathsf{sid},u,w_1)$;
  - Send $a := g\cdot u^{-1}$ to P_2;
- Upon receiving $(\textsc{Input-I},\mathsf{sid},h)$ from the environment $\mathcal{Z}$, the party P_2 does:
  - Send $(\textsc{Prepare-I},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-I},\mathsf{sid},v,w_2)$;
  - Send $b := h - v$ to P_1;
- Upon receiving $b\in\mathbb{M}$ from P_2, the party P_1 does:
  - Set $s_1 := g\cdot b + a\cdot w_1$;
  - Return $(\textsc{Output-I},\mathsf{sid},s_1)$ to the environment $\mathcal{Z}$ and halt;
- Upon receiving $a\in\mathbb{G}$ from P_1, the party P_2 does:
  - Set $s_2 := a\cdot w_2$;
  - Return $(\textsc{Output-I},\mathsf{sid},s_2)$ to the environment $\mathcal{Z}$ and halt;

Figure 4: Protocol $\Pi^{\mathrm{I}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ for $\mathbb{G}$-module (setting I) in the $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$-hybrid model

P_2 stores $\{w_{t,2}\}_t$. In the online phase, P_1 uses k_1 to generate $(u_t,w_{t,1})\leftarrow F_{k_1}(t)$, and P_2 uses k_2 to generate $(\tilde{u}_t,v_t)\leftarrow F_{k_2}(t)$, and restores $w_{t,2}$.

2PC for $\mathbb{G}$-module (Setting II). In this setting, party P_1 has private input $g_1\in\mathbb{G}, h_1\in\mathbb{M}$ and party P_2 has private input $g_2\in\mathbb{G}, h_2\in\mathbb{M}$. After the protocol execution, P_1 and P_2 output shares of $g_1\cdot g_2\cdot(h_1+h_2)\in\mathbb{M}$. The protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is described in Fig. 5.

Theorem 2 *Let $\mathbb{G}$ be a finite abelian group, and $\mathbb{M}$ be a finite $\mathbb{G}$-module. The protocol $\Pi^{II}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ described in Fig. 5 UC-realizes $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ described in Fig. 2 in the $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$-hybrid model against semi-honest PPT adversaries with static corruption.*

Proof: See Appendix. B.

Efficiency. Similarly, in the server-aid mode, the offline communication of protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is $2\log|\mathbb{G}|+4\log|\mathbb{M}|$ bits, and it can be further reduced to $\log|\mathbb{M}|$ bit using PRF. The online communication of protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is $2(\log|\mathbb{G}|+\log|\mathbb{M}|)$ bits, and in one round. Hence the total communication of protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is $2\log|\mathbb{G}|+3\log|\mathbb{M}|$ bits.

2PC for $\mathbb{G}$-module (Setting III). In this setting, party P_1 has private input $h_1\in\mathbb{M}$ and party P_2 has private input $h_2\in\mathbb{M}$. After the protocol execution, P_1 gets a random element $g_1\in\mathbb{G}$, and P_2 gets $g_2\in\mathbb{G}(h_1+h_2)$ such that $g_1\cdot g_2 = h_1+h_2$. The protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is described in Fig. 6.

Protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$

Protocol:

- Upon receiving $(\textsc{Input-II},\mathsf{sid},g_1,h_1)$ from the environment $\mathcal{Z}$, the party P_1:
  - Send $(\textsc{Prepare-II},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-II},\mathsf{sid},u_1,v_1,w_1)$;
  - Send $a_1 := g_1\cdot u_1^{-1}$ and $b_1 := g_1\cdot(h_1-v_1)$ to P_2;
- Upon receiving $(\textsc{Input-II},\mathsf{sid},g_2,h_2)$ from the environment $\mathcal{Z}$, the party P_2:
  - Send $(\textsc{Prepare-II},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-II},\mathsf{sid},u_2,v_2,w_2)$;
  - Send $a_2 := g_2\cdot u_2^{-1}$ and $b_2 := g_2\cdot(h_2-v_2)$ to P_1;
- Upon receiving $a_2\in\mathbb{G}, b_2\in\mathbb{M}$ from P_2, the party P_1:
  - Set $s_1 := g_1\cdot b_2 + a_1\cdot a_2\cdot w_1$;
  - Return $(\textsc{Output-II},\mathsf{sid},s_1)$ to the environment $\mathcal{Z}$ and halt;
- Upon receiving $a_1\in\mathbb{G}, b_1\in\mathbb{M}$ from P_1, the party P_2:
  - Set $s_2 := g_2\cdot b_1 + a_2\cdot a_1\cdot w_2$;
  - Return $(\textsc{Output-II},\mathsf{sid},s_1)$ to the environment $\mathcal{Z}$ and halt;

Figure 5: Protocol $\Pi^{\mathrm{II}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ for $\mathbb{G}$-module (setting II) in the $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$-hybrid model

Protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$

Protocol:

- Upon receiving $(\textsc{Input-III},\mathsf{sid},h_2)$ from the environment $\mathcal{Z}$, the party P_2 does:
  - Send $(\textsc{Prepare-III},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-III},\mathsf{sid},v,w_2)$;
  - Send $a_2 := h_2 - w_2$ to P_1;
- Upon receiving $(\textsc{Input-III},\mathsf{sid},h_1)$ from the environment $\mathcal{Z}$ and a_2 from P_2, the party P_1 does:
  - Send $(\textsc{Prepare-III},\mathsf{sid})$ to $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, obtaining $(\textsc{Prepare-III},\mathsf{sid},u,w_1)$;
  - Send $a_1 := u^{-1}\cdot(h_1-w_1+a_2)$ and to P_2;
  - Return $(\textsc{Output-III},\mathsf{sid},u)$ to the environment $\mathcal{Z}$ and halt;
- Upon receiving $a_1\in\mathbb{G}$ from P_1, the party P_2 does :
  - Set $h := a_1 + v$;
  - Return $(\textsc{Output-III},\mathsf{sid},h)$ to the environment $\mathcal{Z}$ and halt;

Figure 6: Protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ for $\mathbb{G}$-module (setting III) in the $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$-hybrid model

Theorem 3 *Let $\mathbb{G}$ be a finite group, and $\mathbb{M}$ be a finite $\mathbb{G}$-module. The protocol $\Pi^{III}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ described in Fig. 6 UC-realizes $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ described in Fig. 2 in the $\mathcal{F}^{\mathsf{offline}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$-hybrid model against semi-honest PPT adversaries with static corruption.*

Proof: See Appendix. C.

Efficiency. Similarly, in the servier-aid mode, the offline communication of protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ is $\log|\mathbb{G}|+3\log|\mathbb{M}|$ bits, and it can be further reduced to $\log|\mathbb{M}|$ bits using PRF.

Protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$

Common input: $\mathbb{Z}_n, \mathbb{Z}_{2^\ell}, \Psi := Map(\mathbb{Z}_n, \mathbb{Z}_{2^l})$
P_1's input: $k \in \mathbb{Z}_n$
P_2's input: $\psi \in \Psi$

- P_1 and P_2 jointly invoke protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[\mathbb{Z}_n, \Psi]$, and obtain the shares (ψ_1, ψ_2) of $\textsf{L-Shift}_k(\psi)$.
- P_1 outputs ψ_1, and P_2 outputs ψ_2.

Figure 7: The Oblivious Shift Protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$

Protocol $\Pi^{n,\ell}_{\textsf{S-OT}}$

Common input: $\mathbb{Z}_n, \mathbb{Z}_{2^\ell}, \Psi := Map(\mathbb{Z}_n, \mathbb{Z}_{2^l})$
P_1's input: $i \in \mathbb{Z}_n$
P_2's input: $\psi \in \Psi$

- P_1 and P_2 jointly invoke protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[\mathbb{Z}_n, \Psi]$, and obtain the shares (ψ_1, ψ_2) of $\textsf{L-Shift}_i(\psi)$.
- P_1 outputs $\psi_1(0)$, and P_2 outputs $\psi_2(0)$.

Figure 8: The one-round shared OT protocol $\Pi^{n,\ell}_{\textsf{S-OT}}$

The online communication of protocol $\Pi^{\mathrm{III}}_{\textsf{GM}}[\mathbb{G}, \mathbb{M}]$ is $2\log|\mathbb{M}|$, and in two rounds. Hence the total communication of protocol $\Pi^{\mathrm{III}}_{\textsf{GM}}[\mathbb{G}, \mathbb{M}]$ is $3\log|\mathbb{M}|$.

5 Instantiations and Applications

Oblivious cyclic shift. In an oblivious cyclic shift protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$, P_1 inputs a private offset $k \in \mathbb{Z}_n$, and P_2 inputs a private vector $x := (x_0, \ldots, x_{n-1})$, where $x_i \in \mathbb{Z}_{2^\ell}$. After the protocol execution, P_1 and P_2 obtains shared vector $([x_k], [x_{k+1}], \ldots, [x_{k-1}])$.

The oblivious cyclic shift protocol is an instantiation of $\mathbb{G}$-module (setting I). Let $\Psi := Map(\mathbb{Z}_n, \mathbb{Z}_{2^l})$ be the set consisting of all the maps from $\mathbb{Z}_n$ to $\mathbb{Z}_{2^l}$. For any $k \in \mathbb{Z}_n$, let $\textsf{L-Shift}_k : \Psi \mapsto \Psi$ be the "cyclic left shift" on Ψ, which is defined by $\textsf{L-Shift}_k(\psi)(i) := \psi(i+k)$ with $\psi \in \Psi$ and $i \in \mathbb{Z}_n$. There is a $\mathbb{Z}_n$-module structure on Ψ defined as:

$$\begin{array}{ccccc} \mathbb{Z}_n & \times & \Psi & \longrightarrow & \Psi \\ (k & , & \psi) & \longmapsto & \textsf{L-Shift}_k(\psi) \end{array}$$

Now, we can view protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$ as follows. P_1 inputs $k \in \mathbb{Z}_n$, and P_2 inputs map $\psi \in \Psi$ (represented by a vector $(\psi(0), \ldots, \psi(n-1))$). At the end of the protocol, P_1 and P_2 obtain shared vector representation of $\textsf{L-Shift}_k(\psi)$ in Ψ. As depicted in Fig. 7, we can use the protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[\mathbb{Z}_n, \Psi]$ to realize $\Pi^{n,\ell}_{\textsf{O-shift}}$.

Efficiency. The communication of protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$ is exactly that of the protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[\mathbb{Z}_n \Psi]$. In the server-aid mode, offline: $n\ell$ bits, online: $n\ell + \log n$ bits, 1 round.

One-round $(1,n)$**-shared OT.** In an $(1,n)$-Shared OT protocol $\Pi^{n,\ell}_{\textsf{S-OT}}$, P_1 inputs an index $i \in \mathbb{Z}_n$, and P_2 inputs messages $(x_0, \ldots, x_{n-1})$, $x_i \in \mathbb{Z}_{2^l}$; after the protocol execution, P_1 and P_2 obtain shared $[x_i]$. Conventionally, such a protocol requires two rounds in the online phase. (cf. Appendix. D).

In this paper, we give a one-round $(1,n)$-shared OT protocol. This $(1,n)$-Shared OT protocol is also an instantiation of $\mathbb{G}$-module (setting I). Let $\Psi := Map(\mathbb{Z}_n, \mathbb{Z}_{2^\ell})$ be the set consisting of all the maps from $\mathbb{Z}_n$ to $\mathbb{Z}_{2^l}$. Equivalently, we can view that P_1 inputs $i \in \mathbb{Z}_n$, and P_2 inputs a map $\psi \in \Psi$ (represented by a vector $(\psi(0), \ldots, \psi(n-1))$). At the end of the protocol, they obtain shares of $[\psi(i)]$.

Because $\psi(i) = \textsf{L-Shift}_i(\psi)(0)$, we can realize one-round $(1,n)$-Shared OT using the protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$. The protocol $\Pi^{n,\ell}_{\textsf{S-OT}}$ is depicted in Fig. 8:

Efficiency. The communication of the protocol $\Pi^{n,\ell}_{\textsf{S-OT}}$ is exactly that of the protocol $\Pi^{n,\ell}_{\textsf{O-shift}}$, that is, in the server-aid mode, offline: $n\ell$ bits, online: $n\ell + \log n$ bits, 1 round.

Oblivious permutation. In an obvious permutation protocol $\Pi^{n,\ell}_{\textsf{O-Perm}}$, P_1 inputs permutation $\sigma \in S_n$, where S_n is the symmetric group of degree n, and P_2 inputs $x := (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n_{2^\ell}$, where $x_i \in \mathbb{Z}_{2^l}$; after the protocol execution, P_1 and P_2 obtain shares of $\sigma(x)$ in $\mathbb{Z}^n_{2^\ell}$, where $\sigma(x)$ is defined by $\sigma(x)_i = x_{\sigma^{-1}(i)}$.

There is a S_n-module structure on $\mathbb{Z}^n_{2^\ell}$ defined as follows:

$$\begin{array}{ccccc} S_n & \times & \mathbb{Z}^n_{2^l} & \longrightarrow & \mathbb{Z}^n_{2^\ell} \\ (\sigma & , & x) & \longmapsto & \sigma(x) \end{array}$$

As depicted in Fig. 9, we can use the protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[S_n, \mathbb{Z}^n_{2^\ell}]$ to realize $\Pi^{n,\ell}_{\textsf{O-Perm}}$.

Efficiency. The communication of the protocol $\Pi^{n,\ell}_{\textsf{O-Perm}}$ is exactly that of the protocol $\Pi^{\mathrm{I}}_{\textsf{GM}}[S_n, \mathbb{Z}^n_{2^\ell}]$. In the server-aid mode, offline: $n\ell$ bits, online: $n\ell + \log n! < n\ell + n\log n$ bits, 1 round.

Oblivious shuffle. In an obvious shuffle protocol $\Pi^{n,\ell}_{\textsf{O-shuffle}}$, P_1 and P_2 holds a shared vector $x := (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n_{2^\ell}$. They want to oblivious shuffle x, and obtain shared $[\sigma(x)]$, where $\sigma \in S_n$ is oblivious to both P_1 and P_2.

Intuitively, oblivious shuffle can be achieved by applying the oblivious permutation $\Pi^{n,\ell}_{\textsf{O-Perm}}$ twice. At the beginning of the protocol, suppose P_1 holds shares $x^{(1)} := (x^{(1)}_0, \ldots, x^{(1)}_{n-1})$ and P_2 holds shares $x^{(2)} := (x^{(2)}_0, \ldots, x^{(2)}_{n-1})$ such that $x^{(1)}_i + x^{(2)}_i = x_i$. First, P_1 picks a random permutation $\sigma_1 \in S_n$; P_1 and P_2 invoke $\Pi^{n,\ell}_{\textsf{O-Perm}}$ to permute P_2's shares, obtaining shares of $\sigma_1(x^{(2)})$ as $(w^{(1)}, w^{(2)})$. P_1 then resets $w^{(1)} := w^{(1)} + \sigma_1(x^{(1)})$. It is easy to see that $w^{(1)} + w^{(2)} = \sigma_1(x)$. Analogously, P_2 then picks a random permutation $\sigma_2 \in S_n$. P_1 and P_2 invoke $\Pi^{n,\ell}_{\textsf{O-Perm}}$ to permute P_1's shares, obtaining shares of $\sigma_2(w^{(1)})$ as $(z^{(1)}, z^{(2)})$. P_2 then resets $z^{(2)} := z^{(2)} + \sigma_2(w^{(2)})$. Hence, $z^{(1)} + z^{(2)} = \sigma_2(\sigma_1(x))$. Since P_1 is oblivious to σ_2 and P_2 is

Table 5.1: Comparison to exists Oblivious Shuffle protocols

l	Protocol	offline com.	online comm.	online round	total comm.
1	Ours	$2n\ell$	$2\log n! + 2n\ell$	2	$2\log n! + 4n\ell$
1	Secret-Shared Shuffle [CGP20]	0	$3dn\lambda \log T + (d+1)nl$ *	3	$3dn\lambda \log T + (d+1)nl$ *
32	Ours	$64n$	$< 64n + 2n\log n$	2	$< 2n\log n + 128n$
32	Secret-Shared Shuffle [CGP20]	0	$\geq 384n\log n + 64n$	3	$\geq 384n\log n + 64n$
64	Ours	$128n$	$< 128n + 2n\log n$	2	$< 2n\log n + 256n$
64	Secret-Shared Shuffle [CGP20]	0	$\geq 384n\log n + 128n$	3	$\geq 384n\log n + 128n$
128	Ours	$256n$	$< 256n + 2n\log n$	2	$< 2n\log n + 512n$
128	Secret-Shared Shuffle [CGP20]	0	$\geq 384n\log n + 256n$	3	$\geq 384n\log n + 256n$

* $\lambda = 128, d = 2\lceil \frac{\log n}{\log T} \rceil - 1$, and $T = 2, 3, \cdots n$.

oblivious to σ_1, the entire shuffle is oblivious to both parties. The protocol is given in Fig. 10.

Efficiency. The communication of the protocol $\Pi^{n,\ell}_{\text{O-shuffle}}$ is twice of $\Pi^{n,\ell}_{\text{O-Perm}}$, i.e., offline: $2n\ell$ bits, online: $2(n\ell + \log n!) < 2(n\ell + n\log n)$ bits, 2 round.

In Table. 5.1, we compare our $\Pi^{n,\ell}_{\text{O-shuffle}}$ where the secret-shared shuffle protocol in [CGP20]. When n is big, our protocol is much more efficient than theirs. The overall communication (offline+online) of our protocol less than $1/192$ of [CGP20] when $n \longrightarrow \infty$. Besides, our protocol does not use cryptographic operations, while the secret-shared shuffle protocol in [CGP20] heavily uses symmetric cryptography.

Oblivious Selection. Let $x, y \in \mathbb{Z}_N$, $a \in \mathbb{Z}_2$. In an oblivious selection protocol $\Pi^N_{\text{O-select}}$, P_1 and P_2 holds $[x], [y], [a]$, where x, y are additively shared in $\mathbb{Z}_N$ and a is binary shared in $\mathbb{Z}_2$. After the protocol execution, P_1 and P_2 obtains shared $[s]$ over $\mathbb{Z}_N$ which is defined as

$$s := \begin{cases} x & \text{if } a = 1, \\ y & \text{if } a = 0. \end{cases}$$

We call $a \in \mathbb{Z}_2$ the selection bit. Hereby, we first resolve a special case, where $y = 0$. Note that since $s = a(x - y) + y$, general case can be reduced to special case without additional communication. In the following, we show how to instantiate $\mathbb{G}$-module (setting II) to realize the special oblivious selection protocol $\Pi^{N,\text{special}}_{\text{O-select}}$. When $y = 0$, P_1 and P_2 jointly compute the equation

$$ax = \frac{x - (-1)^a x}{2} \quad \text{for } a = 0, 1, \text{ and } x \in \mathbb{Z}_N.$$

Special oblivious selection when N is odd. Let $a = a_1 + a_2 \pmod 2$, $x = x_1 + x_2 \pmod N$ be the shares of a over $\mathbb{Z}_2$ and x over $\mathbb{Z}_N$ respectively, where N is an odd number. We have

$$\begin{aligned}(-1)^a x &= (-1)^{a_1 + a_2}(x_1 + x_2) \\ &= (-1)^{a_1}(-1)^{a_2}(x_1 + x_2) \in \mathbb{Z}_N\end{aligned}$$

Protocol $\Pi^{n,\ell}_{\text{O-Perm}}$

Common input: $S_n, \mathbb{Z}^n_{2^\ell}$
P_1's input: $\sigma \in S_n$
P_2's input: $x := (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n_{2^\ell}$

- P_1 and P_2 jointly invoke protocol $\Pi^{\text{I}}_{\text{GM}}[S_n, \mathbb{Z}^n_{2^\ell}]$ to get the share (z_1, z_2) of $\sigma(x)$.
- P_1 outputs z_1, and P_2 outputs z_2.

Figure 9: The oblivious permutation protocol $\Pi^{n,\ell}_{\text{O-Perm}}$

Let $\mathbb{G} := \{\pm 1\}, \mathbb{M} := \mathbb{Z}_N$. It easy to see that $\mathbb{M}$ is a $\mathbb{G}$-module. Hence we can use the protocol $\Pi^{\text{II}}_{\text{GM}}[\{\pm 1\}, \mathbb{Z}_N]$ to compute $(-1)^a x$ and hence to compute $x - (-1)^a x \in \mathbb{Z}_N$. In the case that N is an odd number, then 2 is invertible in $\mathbb{Z}_N$, and hence it easy to compute ax from $x - (-1)^a x$. The protocol is depicted in Fig. 11.

Special oblivious selection when N is even. In the case that N is an even number, 2 is not invertible in $\mathbb{Z}_N$, we need to modify the protocol. In fact, if N is an even number, $a = a_1 + a_2 \mod 2$, $x = x_1 + x_2 \mod N$ are the share representations of a and x respectively. One can lift x_1, x_2 to $\tilde{x}_1, \tilde{x}_2 \in \mathbb{Z}_{2N}$ respectively. Let $\tilde{x} := \tilde{x}_0 + \tilde{x}_1 \mod 2N$, then we have $ax \equiv a\tilde{x} \mod N$ for $a = 0, 1$.

Using the same method as in the case N is an odd number, we can get the share representations of $2a\tilde{x} = \tilde{x} - (-1)^a \tilde{x} \in \mathbb{Z}_{2N}$, and $a\tilde{x} \in \mathbb{Z}_N$ which is equal to $ax \in \mathbb{Z}_N$ (cf. Fig. 12).

Oblivious selection. In Fig. 13, we present oblivious selection protocol $\Pi^N_{\text{O-select}}$ using $\Pi^{N,\text{special}}_{\text{O-select}}$ protocol as a building block.

Efficiency. We compare the efficiency of our protocols with SecureNN [WGC19] in Table 5.2. The communication for our special oblivious selection protocols and general oblivious selection protocol are essentially the same. It easy to see that the online communication, round and the total (online+offline)

Protocol $\Pi^{n,\ell}_{\text{O-shuffle}}$

Common input: $S_n, \mathbb{Z}^n_{2^\ell}$
P_1's input: $x^{(1)} := (x^{(1)}_0, \ldots, x^{(1)}_{n-1}) \in \mathbb{Z}^n_{2^\ell}$
P_2's input: $x^{(2)} := (x^{(2)}_0, \ldots, x^{(2)}_{n-1}) \in \mathbb{Z}^n_{2^\ell}$

- P_1 picks random $\sigma_1 \leftarrow S_n$.
- P_1 and P_2 jointly invoke protocol $\Pi^{n,\ell}_{\text{O-Perm}}$, where P_1 inputs σ_1 and P_2 inputs $x^{(2)}$. After the execution, P_1 and P_2 obtain $w^{(1)}$ and $w^{(2)}$, respectively.
- P_1 resets $w^{(1)} := w^{(1)} + \sigma_1(x^{(1)})$.
- P_2 picks random $\sigma_2 \leftarrow S_n$.
- P_2 and P_1 jointly invoke protocol $\Pi^{n,\ell}_{\text{O-Perm}}$, where P_2 inputs σ_2 and P_1 inputs $w^{(1)}$. After the execution, P_1 and P_2 obtain $z^{(1)}$ and $z^{(2)}$, respectively.
- P_2 resets $z^{(2)} := z^{(2)} + \sigma_2(w^{(2)})$.
- P_1 outputs $z^{(1)}$, and P_2 outputs $z^{(2)}$.

Figure 10: The oblivious shuffle protocol $\Pi^{n,\ell}_{\text{O-shuffle}}$

Protocol $\Pi^{N,\text{special}}_{\text{O-select}}$ (for odd N)

Common input: $\{\pm 1\}$, $\mathbb{Z}_N$
P_1's input: $x_1 \in \mathbb{Z}_N$, $a_1 \in \mathbb{Z}_2$
P_2's input: $x_2 \in \mathbb{Z}_N$, $a_2 \in \mathbb{Z}_2$

- P_1 and P_2 jointly invoke protocol $\Pi^{\text{II}}_{\text{GM}}[\{\pm 1\}, \mathbb{Z}_N]$, where P_1 inputs $((-1)^{a_1}, x_1)$ and P_2 inputs $((-1)^{a_2}, x_2)$. After the execution, P_1 and P_2 obtain w_1 and w_2, respectively, as share representation of $(-1)^a x \in \mathbb{Z}_N$.
- P_1 computes $z_1 := \frac{x_1 - w_1}{2}$.
- P_2 computes $z_2 := \frac{x_2 - w_2}{2}$.
- P_1 outputs z_1, and P_2 outputs z_2.

Figure 11: The special oblivious selection protocol $\Pi^{N,\text{special}}_{\text{O-select}}$ (for odd module N)

communication of our protocols less than the one in SecureNN [WGC19] when $\log N \geq 3$.

First Non-zero Bit. Let $p \geq n+2$ be a prime number. P_1, P_2 hold shares of a non-zero binary vector $u := (u_0, \ldots, u_{n-1})$ over $\mathbb{Z}^n_p$. It satisfies

$$u_i = 0 \text{ or } 1, \forall\, i \in \mathbb{Z}_n \qquad \& \qquad \exists\, u_j = 1,\ j \in \mathbb{Z}_n$$

They want to jointly locate the first non-zero bit of u. At the end of the protocol, P_1, P_2 will get the shares of $\min\{i \in \mathbb{Z}_n : u_i \neq 0\}$ over $\mathbb{Z}_n$. Let $\mathbb{G} := \mathbb{Z}_n \ltimes (\mathbb{Z}^*_p)^n$ be the semi-direct product of the groups $\mathbb{Z}_n$ and $(\mathbb{Z}^*_p)^n$. The underlying set of the group $\mathbb{G}$ is the Cartesian product $\mathbb{Z}_n \times (\mathbb{Z}^*_p)^n$ while the group operation is defined by

$$\begin{aligned} \mathbb{G} \times \mathbb{G} &\longrightarrow \mathbb{G} \\ ((i,a),(j,b)) &\longmapsto (i+j, a * \text{L-shift}_i(b)) \end{aligned}$$

Here L-shift_i is the i-th circular left shift operator on $\mathbb{Z}^n_p$, i.e., for $x = (x_0, x_1, \cdots x_{n-1}) \in \mathbb{Z}^n_p$, we have

$$\text{L-shift}_i(x) = (x_i, x_{i+1}, \cdots, x_{n-1}, x_0, \cdots x_{i-1}) \in \mathbb{Z}^n_p$$

Protocol $\Pi^{N,\text{special}}_{\text{O-select}}$ (for even N)

Common input: $\{\pm 1\}$, $\mathbb{Z}_N$
P_1's input: $x_1 \in \mathbb{Z}_N$, $a_1 \in \mathbb{Z}_2$
P_2's input: $x_2 \in \mathbb{Z}_N$, $a_2 \in \mathbb{Z}_2$

- P_1 and P_2 view x_1 and x_2 as elements in $\mathbb{Z}_{2N}$.
- P_1 and P_2 jointly invoke protocol $\Pi^{\text{II}}_{\text{GM}}[\{\pm 1\}, \mathbb{Z}_{2N}]$, where P_1 inputs $((-1)^{a_1}, x_1)$ and P_2 inputs $((-1)^{a_2}, x_2)$. After the execution, P_1 and P_2 obtain w_1 and w_2, respectively, as share representation of $(-1)^a x \in \mathbb{Z}_{2N}$.
- P_1 computes $z_1 := \lfloor \frac{x_1 - w_1}{2} \rfloor \mod N$.
- P_2 computes $z_2 := \lceil \frac{x_2 - w_2}{2} \rceil \mod N$.
- P_1 outputs z_1, and P_2 outputs z_2.

Figure 12: The special oblivious selection protocol $\Pi^{N,\text{special}}_{\text{O-select}}$ (for even module N)

Protocol $\Pi^{N}_{\text{O-select}}$

Common input: $\{\pm 1\}$, $\mathbb{Z}_N$
P_1's input: $x_1 \in \mathbb{Z}_N$, $y_1 \in \mathbb{Z}_N$, $a_1 \in \mathbb{Z}_2$
P_2's input: $x_2 \in \mathbb{Z}_N$, $y_2 \in \mathbb{Z}_N$, $a_2 \in \mathbb{Z}_2$

- P_1 and P_2 jointly invoke protocol $\Pi^{N,\text{special}}_{\text{O-select}}$, where P_1 inputs $((-1)^{a_1}, y_1 - x_1)$ and P_2 inputs $((-1)^{a_2}, y_2 - x_2)$. After the execution, P_1 and P_2 obtain w_1 and w_2, respectively, as share representation of $a(y-x) \in \mathbb{Z}_{2N}$.
- P_1 outputs $w_1 + x_1$, and P_2 outputs $w_2 + x_2$.

Figure 13: The special oblivious selection protocol $\Pi^{N}_{\text{O-select}}$

Table 5.2: Comparison to exists oblivious selection protocol

N	Protocol	offline com.	online com.	round	total com.
odd N	$\Pi^N_{\text{O-select}}$	$\log N$	$2(1+\log N)$	1	$2+3\log N$
even N	$\Pi^N_{\text{O-select}}$	$\log N + 1$	$2(2+\log N)$	1	$5+3\log N$
N	SecureNN	0	$5\log N$	2	$5\log N$
2^{32}	$\Pi^N_{\text{O-select}}$	33	68	1	101
2^{32}	SecureNN	0	160	2	160
2^{64}	$\Pi^N_{\text{O-select}}$	65	132	1	197
2^{64}	SecureNN	0	320	2	320
2^{128}	$\Pi^N_{\text{O-select}}$	129	260	1	389
2^{128}	SecureNN	0	640	2	640

We verify that $\mathbb{G}$ is a non commutative group with the identity element $(0, 1^n)$. One can define the $\mathbb{G}$-module structure on $\mathbb{Z}^n_p$ as follows:

$$\begin{aligned} \mathbb{G} \times \mathbb{Z}^n_p &\longrightarrow \mathbb{Z}^n_p \\ ((i,a),x) &\longmapsto a * \text{L-shift}_i(x) \end{aligned}$$

Then we have the following Lemma.

Lemma 1 *Let $\mathbb{G} := \mathbb{Z}_n \ltimes (\mathbb{Z}^*_p)^n$ be the semi-direct product of the group $\mathbb{Z}_n$ and the group $(\mathbb{Z}^*_p)^n$. There is a $\mathbb{G}$-orbit decomposition*

$$\mathbb{Z}^n_p = \coprod_{d=0}^{n} U_d$$

Protocol $\Pi^{p,n}_{\mathsf{FNZ}}$

Common input: $\mathbb{Z}_n \ltimes (\mathbb{Z}_p^*)^n$, $\mathbb{Z}_p^n$
P_1's input: $(u_0^{(1)}, \dots, u_{n-1}^{(1)}) \in \mathbb{Z}_p^n$
P_2's input: $(u_0^{(2)}, \dots, u_{n-1}^{(2)}) \in \mathbb{Z}_p^n$

- P_1 computes prefix sum $v_i^{(1)} := \sum_{j=0}^{i} u_i^{(1)} \in \mathbb{Z}_p$ for $i \in \mathbb{Z}_n$, and then computes $z_i^{(1)} = f(u_i^{(1)}, v_i^{(1)}) \in \mathbb{Z}_p$, for $i \in \mathbb{Z}_n$.
- P_2 computes prefix sum $v_i^{(2)} := \sum_{j=0}^{i} u_i^{(2)} \in \mathbb{Z}_p$ for $i \in \mathbb{Z}_n$, and then computes $z_i^{(2)} = f(u_i^{(2)}, v_i^{(2)}) \in \mathbb{Z}_p$, for $i \in \mathbb{Z}_n$.
- P_1 and P_2 jointly invoke protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{Z}_n \ltimes (\mathbb{Z}_p^*)^n, \mathbb{Z}_p^n]$. After the execution, P_1 obtains $g = (i, c) \in \mathbb{Z}_n \ltimes (\mathbb{Z}_p^*)^n$, and P_2 obtains $w \in \mathbb{Z}_p^n$ such that $gw = z$.
- P_2 takes j as the only $j \in \mathbb{Z}_n$ such that $w_j = 0$.
- P_1 outputs $-i$, and P_2 outputs j.

Figure 14: The first non-zero bit protocol $\Pi^{p,n}_{\mathsf{FNZ}}$

Protocol Π^{n}_{SC}

Common input: $\mathbb{Z}_{2^n}$
P_1's input: $x := (x_0, \dots, x_{n-1}) \in\in \mathbb{Z}_{2^n}$
P_2's input: $y := (y_0, \dots, y_{n-1}) \in \mathbb{Z}_{2^n}$

- P_1 appends $x_n = 1$ to x, and P_2 appends $y_n = 0$ to y.
- For $i \in \mathbb{Z}_{n+1}$, P_1 and P_2 view $z_i := x_i + y_i \mod 2$ as the jointly shared value $[z_i]$ over $\mathbb{Z}_2$, and invoke $\Pi^{2,p}_{\mathsf{convert}}$ to convert shares of z_i over $\mathbb{Z}_2$ to shares of z_i over $\mathbb{Z}_p$.
- P_1 and P_2 invoke $\Pi^{p,n+1}_{\mathsf{FNZ}}$ to obtain shares of the index of the first non-zero bit of $(z_0, \dots, z_n)$, denoted as $j_1 \in \mathbb{Z}_{n+1}$ and $j_2 \in \mathbb{Z}_{n+1}$.
- P_2 cyclic left-shift $y' \leftarrow \mathsf{L\text{-}shift}_{j_2}(y)$.
- P_1 and P_2 invoke $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$, where P_1 inputs j_1, and P_2 inputs y'. After the execution, P_1 and P_2 obtains $w_1 \in \mathbb{Z}_2$ and $w_2 \in \mathbb{Z}_2$ as the share representation of $y_{j_1+j_2}$.
- P_1 outputs w_1, and P_2 outputs w_2.

Figure 15: The secure comparison protocol Π^{n}_{SC}

of $\mathbb{Z}_p^n$, where U_d is the subset of $\mathbb{Z}_p^n$ consisting of the elements of Hamming weight d. i.e. having d non-zero coordinates.

We can instantiate $\mathbb{G}$-module setting III protocol $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{Z}_n \ltimes (\mathbb{Z}_p^*)^n, \mathbb{Z}_p^n]$ to realizing the first non-zero bit protocol $\Pi^{p,n}_{\mathsf{FNZ}}$. The main idea comes from the following lemma:

Lemma 2 *Let $p \geq n+2$ be a prime number, and let $u = (u_i)_{i=0}^{n-1}$ be a non-zero 0-1 vector in $\mathbb{Z}_p^n$. Let $v \in \mathbb{Z}_p^n$ defined as*

$$v_0 = u_0$$
$$v_i = v_{i-1} + u_i \quad \text{for } i = 1, 2, \cdots, n-1$$

Thus $v_i \in \mathbb{Z}_{n+1}$ for all $i \in \mathbb{Z}_n$. Let f be a map

$$f : \{0,1\} \times \mathbb{Z}_{n+1} \longrightarrow \mathbb{Z}_p$$
$$(a, b) \longmapsto b - 2a + 1 \mod p$$

Then we have $\min\{i | u_i \neq 0, i \in \mathbb{Z}_n\}$ *is the unique $i \in \mathbb{Z}_n$ such that $f(u_i, v_i) = 0 \mod p$.*

Proof: First we claim that $(1,1)$ is the unique $(a,b) \in \{0,1\} \times \mathbb{Z}_{n+1}$ such that $f(a,b) = 0 \mod p$. That is because if $a = 0$, $f(a,b) \in \{1, 2, \cdots n+1\}$ for $b \in \mathbb{Z}_{n+1}$, which implies $f(a,b) \neq 0 \mod p$; while if $a = 1$, then $b = 1$ is the only solution such that of $f(1,b) = 0 \mod p$.

Now it is not difficult to see that $\min\{i = 0, 1, \cdots, n-1 | u_i \neq 0\}$ is the unique $i \in \mathbb{Z}_n$ such that both $u_i = 1$ and $v_i = 1$. Thus $\min\{i | u_i \neq 0,\ i \in \mathbb{Z}_n\}$ is the unique $i \in \mathbb{Z}_n$ such that $f(u_i, v_i) = 0$ which finishes the proof. ■

Following Lemma 2, we design our protocol to compute the first non-zero bit of a non-zero 0-1 vector $u = (u_i)_{i=0}^{n-1}$ in $\mathbb{Z}_p^n$, where the input is its shares over $\mathbb{Z}_p^n$, and the output is its shares over $\mathbb{Z}_n$. The principle is as follows. Let v and f as in Lemma 2, $z = (f(u_0, v_0)), \cdots, f(u_{n-1}, v_{n-1})) \in \mathbb{Z}_p^n$, and the group $\mathbb{G}$ as in Lemma 1, then the orbit $\mathbb{G}z$ will be the unique $\mathbb{G}$-orbit of Hamming weight $n-1$ in the decomposition in Lemma 1, which is public information that both parties know. If there is a $g = (i, c) \in \mathbb{G}$ and $w \in \mathbb{Z}_p^n$ such that $gw = z$, then the first non-zero bit of z is $(-i + j) \mod n$, where j is the first non-zero bit of w. The protocol is depicted in Fig. 14

Efficiency. The round and communication of the protocol $\Pi^{p,n}_{\mathsf{FNZ}}$ are the same as those of $\Pi^{\mathrm{III}}_{\mathsf{GM}}[\mathbb{Z}_n \ltimes (\mathbb{Z}_p^*)^n, \mathbb{Z}_p^n]$. Hence its offline communication is $n \log p$, its online communication is $2n \log p$ in 1 round. Its communication and round is shown in Table. 1.1.

6 Secure Comparison

During the secure comparison protocol Π^n_{SC}, P_1 holds $x \in \mathbb{Z}_{2^n}$ and P_2 holds $y \in \mathbb{Z}_{2^n}$. After the protocol execution, P_1 and P_2 will obtain shares of $(x \overset{?}{<} y)$ over $\mathbb{Z}_2$. We now show how to construct protocol Π^n_{SC} using $\Pi^{2,p}_{\mathsf{convert}}$ (cf. Appendix E), $\Pi^{p,n+1}_{\mathsf{FNZ}}$, and $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$ as building blocks, where $p \geq n+3$ is a prime number.

Intuitively, for $x \neq y \in \mathbb{Z}_{2^n}$, write the binary representation of x, y as $x = \sum_{i=0}^{n-1} x_i * 2^{n-i-1}, y = \sum_{i=0}^{n-1} y_i * 2^{n-i-1}$, where $x_i, y_i \in \{0, 1\}$.

If i is the leftmost bit such that $x_i \neq y_i$, then $(x < y)$ is same as $(y_i = 1)$ and vice versa. The protocol is depicted in Fig. 15.

Efficiency. The Π^n_{SC} protocol invokes n instances of $\Pi^{2,p}_{\mathsf{convert}}$ (x_n and y_n is public, so it does not require share conversion), 1 instance of $\Pi^{p,n+1}_{\mathsf{FNZ}}$ and 1 instance of $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$, where p is a prime number with $p \geq n+3$. Hence its offline communication is $(2n+1)\log p + n + 1$ bits, and its online communication is $(2n+1)\log p + 3n + \log(n+1) + 1$ bits in 4 round.

However, we can reduce the offline communication to $(2n+1)\log p + 1$ by using the protocol $\Pi^{n+1,1}_{\mathsf{S\text{-}OT2}}$ (cf. Appendix. D) instead of the protocol $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$. The online communication of the protocol $\Pi^{n+1,1}_{\mathsf{S\text{-}OT2}}$ is same as $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$, but the

Protocol Π^n_{DReLU}

Common input: $\mathbb{Z}_{2^n}$
P_1's input: $u := (u_0, \dots, u_{n-1}) \in\in \mathbb{Z}_{2^n}$
P_2's input: $v := (v_0, \dots, v_{n-1}) \in \mathbb{Z}_{2^n}$

- P_1 and P_2 invoke Π^{n-1}_{SC}, where P_1 inputs $2^{n-1} - \tilde{u} - 1$, and P_2 inputs $\tilde{v}$. After the execution, P_1 and P_2 obtains $w_1 \in \mathbb{Z}_2$ and $w_2 \in \mathbb{Z}_2$ as the share representation of $((\tilde{u} + \tilde{v}) \geq 2^{n-1})$ over $\mathbb{Z}_2$.
- P_1 computes $z_1 := 1 + w_1 + u_{n-1} \mod 2$.
- P_2 computes $z_2 := w_2 + v_{n-1} \mod 2$.
- P_1 outputs z_1, and P_2 outputs z_2.

Figure 16: The DReLU protocol Π^n_{DReLU}

Protocol Π^n_{ReLU}

Common input: $\mathbb{Z}_{2^n}$
P_1's input: $u := (u_0, \dots, u_{n-1}) \in\in \mathbb{Z}_{2^n}$
P_2's input: $v := (v_0, \dots, v_{n-1}) \in \mathbb{Z}_{2^n}$

- P_1 and P_2 invoke Π^n_{DReLU}, and obtain $w_1 \in \mathbb{Z}_2$ and $w_2 \in \mathbb{Z}_2$ as the share representation of $\mathsf{DReLU}(x)$ over $\mathbb{Z}_2$.
- P_1 and P_2 invoke $\Pi^{2^n,\mathsf{special}}_{\mathsf{O\text{-}select}}$, where P_1 inputs (u, w_1) and P_2 inputs (v, w_2); after the execution, P_1 and P_2 obtains z_1 and z_2 as the share representation of $\mathsf{DReLU}(x) \cdot x$.
- P_1 outputs z_1, and P_2 outputs z_2.

Figure 17: The ReLU protocol Π^n_{ReLU}

offline communication of $\Pi^{n+1,1}_{\mathsf{S\text{-}OT2}}$ is 1 bit; whereas, the offline communication of the protocol $\Pi^{n+1,1}_{\mathsf{S\text{-}OT}}$ is $n+1$ bits.

The round complexity of the protocol $\Pi^{n+1,1}_{\mathsf{S\text{-}OT2}}$ is 2, but the first move of $\Pi^{n+1,1}_{\mathsf{S\text{-}OT2}}$ can overlap with previous round; therefore, the total round complexity does not increase, which is still 4 rounds. Table. 6.1 gives a comparison between our protocol Π^n_{SC} and well-known protocols [BGI16, Cou18, GSV07, KSS09].

7 Secure computation for DReLU and ReLU

DReLU. In fixed point representation of real number, we usually use two's complement to represent a negative number, hence in order to confirm a number $x \in \mathbb{Z}_{2^n}$ is not " negative", we need to check whether $x < 2^{n-1}$ or not. It is easy to see there the DReLU problem is closely related to finding the most significant bit (MSB) problem. In two's complement representation, $\mathsf{DReLU}(x) = -\mathsf{MSB}(x)$, where MSB means Most Significant Bit.

In the share representation of $x = u + v \in \mathbb{Z}_{2^n}$, one can write u and v in the binary form $u = \sum_{i=0}^{n-1} u_i \cdot 2^i$ and $v = \sum_{i=0}^{n-1} v_i \cdot 2^i$, where $u_i, v_i \in \{0,1\}$ for all $i \in \mathbb{Z}_n$. In the following we will use the notation $\tilde{u} = \sum_{i=0}^{n-2} u_i \cdot 2^i$ and $\tilde{v} = \sum_{i=0}^{n-2} v_i \cdot 2^i$ respectively.

Now we define P, Q be two elements in $\mathbb{Z}_2$ as

$$\begin{aligned} P :=& \quad ((\tilde{u} + \tilde{v}) \geq 2^{n-1}) \text{ (boolean expression)} \\ Q :=& \quad (u_{n-1} + v_{n-1}) \mod 2 \end{aligned}$$

Then we get the following lemma.

Lemma 7.1 *The boolean value of $(x < 2^{n-1})$ is equal to $1 + P + Q \mod 2$ under the identities true $= 1$ and false $= 0$.*

Proof: Under the identities true $= 1$ and false $= 0$, we have

$$\begin{aligned} & (x \geq 2^{n-1}) \\ = & \; u_{n-1} + v_{n-1} + \text{ carry of } \tilde{u} + \tilde{v} \\ = & \; Q + P \mod 2 \end{aligned}$$

Hence we have

$$(x < 2^{n-1}) = 1 + P + Q \mod 2.$$

■

Based on the Lemma 7.1, Fig. 16 describes our DReLU protocol Π^n_{DReLU}.

Remark 7.2 *In SecureNN [WGC19] and CrypTFlow [KRC+20], their protocol Π^n_{DeReLU} restricts its input x in the subdomain $[0, 2^k] \cup [2^n - 2^k, 2^n - 1]$ of $[0, 2^n - 1]$, where $k < n - 1$. Whereas, our protocol works for all the x in $[0, 2^n - 1]$. Besides, in SecureNN [WGC19] and CrypTFlow [KRC+20], the assisting server also needs participate the online phase; whereas, the assisting server only needs to participate the offline phase in our protocol.*

Efficiency. The communication of protocol Π^n_{DReLU} is same as the protocol Π^{n-1}_{SC}, and is shown in Table. 1.1. We compare our protocol Π^n_{DReLU} with well-known protocols in Table. 6.2, where p is a prime number greater than or equal to $n+2$.

ReLU. In the fixed point representation of real number, we usually use two's complement to represent a negative number, hence to compute the $\mathsf{ReLU}(x)$ for a number $x \in \mathbb{Z}_{2^n}$, we need to compute

$$\mathsf{ReLU}(x) = \begin{cases} x & \text{if } x < 2^{n-1} \\ 0 & \text{otherwise} \end{cases}$$

i.e., $\mathsf{ReLU}(x) = \mathsf{DReLU}(x) \cdot x$. Fig. 17 depicts the ReLU protocol Π^n_{ReLU}.

Efficiency. The communication and round of our Π^n_{ReLU} protocol is equal to the sum of that of Π^n_{DReLU} and $\Pi^{2^n,\mathsf{special}}_{\mathsf{O\text{-}select}}$. The communication is shown in the Table 1.1.

Now let us compare the communication of our Π^n_{ReLU} protocol to that in SecureNN ([WGC19]) and CrypTFlow ([KRC+20]) in the Table. 7.1 and from now on let p be a prime number with $p \geq n+2$.

8 Related Work

In terms of secure comparison, many solutions have been proposed in the literature [Yao86, Cou18, GMW87, WMK16,

Table 6.1: Compare to exists SC protocols

n	Protocol	offline com.	online comm.	online round	total comm.
n	Ours	$(2n+1)\log p+1$ [*]	$2(n+1)\log p+$ $3n+\log(n+1)+1$	4	$(4n+3)\log p+$ $3n+\log(n+1)+2$
n	FSS [BGI16]	$\approx 2\lambda n$ [**]	2n	1	$\approx 2\lambda n+2n$
n	NPSETC SC1 [Cou18]	$O(kn/\log k)$ if $n=o(k^2)$ $O(n)$ else	$O(n)$	$O(\log\log n)$	$O(n)$
n	NPSETC SC2 [Cou18]	$O(kn/\log k)$ if $n^{1-1/c}=o(k^2)$ $O(n)$ else	$O(n)$	$O(c\log^* n)$	$O(n)$
n	NPSETC SC3 [Cou18]	$O(kn/\log k)$ if $n^{1-1/c}=o(k^2)$ $O(n)$ else	$O(n)$	$O(c\log^* n)$	$O(n)$
32	Ours	340	446	4	786
32	FSS [BGI16]	$\approx 4096\times 2$	64	1	≈ 8256
32	NPSETC SC1 [Cou18]	15120	530	12	15650
32	NPSETC SC2 [Cou18]	12568	3125	7	15693
32	NPSETC SC3 [Cou18]	12394	622	10	13016
32	GSV07 [GSV07]	14062	1068	6	15130
32	KSS09 [KSS09]	12352	12320	2	24672
64	Ours	784	988	4	1772
64	FSS [BGI16]	$\approx 8512\times 2$	128	1	≈ 17152
64	NPSETC SC1 [Cou18]	31388	1120	12	32508
64	NPSETC SC2 [Cou18]	28872	4138	7	33010
64	NPSETC SC3 [Cou18]	28786	1286	10	30072
64	GSV07 [GSV07]	29072	2208	7	31280
64	KSS09 [KSS09]	24804	24640	2	49344
128	Ours	1809	2207	4	4016
128	FSS [BGI16]	$\approx 16384\times 2$	256	1	≈ 33024
128	NPSETC SC1 [Cou18]	52121	2101	12	54222
128	NPSETC SC2 [Cou18]	48031	5801	7	53832
128	NPSETC SC3 [Cou18]	47963	2239	10	50202
128	GSV07 [GSV07]	59250	4500	8	63750
128	KSS09 [KSS09]	49408	49280	2	98688

[*] Here p is a prime number with $p\ge n+3$.

[**] In paper [BGI16], $\lambda=128$.

Table 6.2: Comparison to exists DReLU protocols

n	Protocol	offline com.	online com.	online round	total com.
n	Ours	$(2n-1)\log p+1$ [*]	$2n\log p+$ $3n+\log n-2$	4	$(4n-1)\log p+$ $3n+\log n-1$
n	CrypTFlow [KRC+20]	0	6n log p + 14n	8	6n log p + 14n
n	SecureNN [WGC19]	0	8n log p + 19n	8	8n log p + 19n
32	Ours	329.2	432.4	4	761.6
32	CrypTFlow [KRC+20]	0	1448.2	8	1448.2
32	SecureNN [WGC19]	0	1941.6	8	1941.6
64	Ours	771.4	972.5	4	1743.9
64	CrypTFlow [KRC+20]	0	3225.4	8	3225.4
64	SecureNN [WGC19]	0	4321.8	8	4321.8
128	Ours	1794.5	2189.6	4	3984.1
128	CrypTFlow [KRC+20]	0	7193.7	8	7193.7
128	SecureNN [WGC19]	0	9634.2	8	9634.2

[*] $p\ge n+2$ is a prime number.

Table 7.1: Comparison to exists ReLU protocols

n	Protocol	offline com.	online comm.	online round	total comm.
n	Ours	$(2n-1)\log p$ [*] $+n+2$	$2n\log p+5n+\log n+2$	5	$(4n-1)\log p+6n+\log n+4$
n	CrypTFlow [KRC+20]	0	$6n\log p+19n$	10	$6n\log p+19n$
n	SecureNN [WGC19]	0	$8n\log p+24n$	10	$8n\log p+24n$
32	Ours	362.2	500.4	5	862.6
32	CrypTFlow [KRC+20]	0	1608.3	10	1608.3
32	SecureNN [WGC19]	0	2101.6	10	2101.6
64	Ours	836.4	1104.5	5	1940.9
64	CrypTFlow [KRC+20]	0	3545.4	10	3545.4
64	SecureNN [WGC19]	0	4641.8	10	4641.8
128	Ours	1923.5	2449.6	5	4373.1
128	CrypTFlow [KRC+20]	0	7833.7	10	7833.7
128	SecureNN [WGC19]	0	10274.2	10	10274.2

[*] $p \geq n+2$ is a prime number.

BGI16,GSV07,KSS09]. To the best of our knowledge, function secret sharing (FSS) based secure comparison [BGI16] is the most online efficient constant-round protocol. It only has one online round and very small online communication. However, as shown in the comparison table, their offline communication is very large.

In terms of solutions for ReLU and DReLU. Recently, they have received lots of research focus due to the increasing popularity of privacy-perserving machine learning. We briefly describe the well-known ones in the following. In [MZ17], SecureML evaluates the ReLU function by switching to Yao sharing. The garbled circuit simply adds the two shares and outputs the most significant bit to compute DReLU. They also consider replacing the ReLU activation function with the square function. This approach improves the online efficiency but consumes more multiplication triplets and increases cost of the offline phase. In [MR18], ABY3 uses polynomial piecewise functions to approximately compute ReLUs. To efficient computation of polynomial piecewise functions, they design a mixed-protocol to directly perform the computation on mixed representation. They compute inequality by extracting the MSB. The bit-extraction is performed with binary secret sharing and computed within the garbled circuit or by an additional round of interaction. In [RWT+18], Chameleon evaluates the ReLU activation function as a MUX operation on the sign bit in the Goldreich-Micali-Wigderson (GMW) protocol. Because the most efficient representation of a function in the GMW protocol is the one that has minimum circuit depth, the Boolean circuits describing the ReLU function in Chameleon are depth-optimized. In [WGC19], SecureNN computes the shares of MSB and flips it to compute DReLU. In more detail, since computing LSB of a number is much easier than computing the MSB (as it does not require bit extraction), they flip the problem to computing LSB as follows: MSB(a) = LSB(2a) when it is working over an odd ring. So their protocols for ReLU completely avoid the use of garbled circuits. In [KRC+20], CrypTFlow modifies how two of the protocols in SecureNN are used - ComputeMSB and ShareConvert to compute ReLU function. Both original protocols ComputeMSB and ShareConvert require P2 to send fresh shares of a value to P0 and P1. CrypTFlow makes one of the shares be computed as the output of a shared PRF key between P2 and one of the parties. This cuts the communication of this step to half. In [MLS+20], Delphi uses garbled circuits to evaluate the ReLU activation function and replaces some ReLU activations with polynomial (specifically, quadratic) approximations. They design a planner that automatically discovers which ReLUs to replace with quadratic approximations so as to maximize the number of approximations used while still ensuring that accuracy remains above a specified threshold.

In above works, SecureML [MZ17], ABY3 [MR18], Delphi [MLS+20] use an aproximate ReLU; Chameleon [RWT+18], ABY2.0 [PSSY21] use a real ReLU, but have non-constant round (In fact they have the round complexity $O(\log n)$). On the other hand, SecureNN [WGC19], CrypTFlow [KRC+20] give protocols for real ReLU with constant round complexity. In the protocol of real ReLU with constant round, the minimal communication is in [KRC+20].

In terms of oblivious shuffle, Laur *et al.* [LWZ11] proposed a generic solution for three or more parties. Recently, Chase *et al.* [CGP20] proposed a 3-round two-party oblivious shuffle protocol only using symmetric cryptographic operations. Its communication is $3dn\lambda\log T+(d+1)nl$, where $d=2\lceil\frac{\log n}{\log T}\rceil-1$, $T=2,\cdots n$; λ is the security parameter, and is advised to take 128. The two-party oblivious shuffle protocol proposed in this work is more efficient w.r.t. both communication and computation.

9 Conclusion

In this work, we present secure computation for a mathematical object: $\mathbb{G}$-module. We then instantiate $\mathbb{G}$-module protocols to realize many useful protocols, such as oblivious shuffle, oblivious selection, secure comparison, DReLU, ReLU, etc. In general, our protocols are more efficient than the state of the arts. In the future, we will instantiate $\mathbb{G}$-module to realize more secure computation protocols.

References

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, Heidelberg, October 2008.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CGP20] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 342–372. Springer, Heidelberg, December 2020.

[Cou18] Geoffroy Couteau. New protocols for secure equality test and comparison. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 303–320. Springer, Heidelberg, July 2018.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987.

[GSV07] Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 330–342. Springer, Heidelberg, April 2007.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.

[HS97] Peter John Hilton and Urs Stammbach. A course in homological algebra. In *Graduate Texts in Mathematics*. Springer, 1997.

[KRC+20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In *2020 IEEE Symposium on Security and Privacy*, pages 336–353. IEEE Computer Society Press, May 2020.

[KSS09] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS 09*, volume 5888 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2009.

[LWZ11] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *ISC 2011*, volume 7001 of *LNCS*, pages 262–277. Springer, Heidelberg, October 2011.

[MLS+20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2505–2522. USENIX Association, August 2020.

[MOR16] Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2pc for mobile phones. *Proceedings on Privacy Enhancing Technologies*, 2016(2):82–99, 2016.

[MR18] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 35–52. ACM Press, October 2018.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017.

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In *30. USENIX Security Symposium (USENIX Security'21)*, Virtual Event, August 11-13, 2021. USENIX. To appear. Full version: https://ia.cr/2020/1225.

[RWT+18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 707–721. ACM Press, April 2018.

[SEM03] *An Introduction to Abstract Algebra*. Walter de Gruyter, 2003.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, 2019(3):26–49, July 2019.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit, 2016. https://github.com/emp-toolkit/ Accessed January 5th, 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Security Proof of Theorem. 1

Proof: To prove Thm. 1, we construct a simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$ where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$-hybrid model and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$ where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the ideal world, and corrupted parties are controlled by the simulator $\mathcal{S}$. We consider following cases.

Case 1: P_1 is corrupted; P_2 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\text{INPUTNOTIFY}, \mathsf{sid}, P_2)$ for the honest P_2 from the external $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$, the simulator $\mathcal{S}$ sends $(\text{PREPARE-I}, \mathsf{sid})$ to $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_2. $\mathcal{S}$ then picks random $b \leftarrow \mathbb{M}$ and sends b to P_1 behave of P_2.
- When the dummy party P_1 in the ideal world receives $(\text{INPUT-I}, \mathsf{sid}, g)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\text{INPUT-I}, \mathsf{sid}, g)$ to the external ideal functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.
Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $b \leftarrow \mathbb{M}$ to P_1 instead of $b := h - v$.

Claim 1 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $v \in \mathbb{M}$ is uniformly random, the distribution of b is also uniformly random regardless h. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \quad .$$

Case 2: P_2 is corrupted; P_1 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\text{INPUTNOTIFY}, \mathsf{sid}, P_1)$ for the honest P_1 from the external $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$, the simulator $\mathcal{S}$ sends $(\text{PREPARE-I}, \mathsf{sid})$ to $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_1. $\mathcal{S}$ then picks random $a \leftarrow \mathbb{G}$ and sends a to P_2 behave of P_1.
- When the dummy party P_2 in the ideal world receives $(\text{INPUT-I}, \mathsf{sid}, h)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\text{INPUT-I}, \mathsf{sid}, h)$ to the external ideal functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.
Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $a \leftarrow \mathbb{G}$ to P_1 instead of $a := g \cdot u^{-1}$.

Claim 2 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $u \in \mathbb{G}$ is uniformly random, the distribution of a is also uniformly random regardless g. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{I}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \quad .$$

Case 3: Both P_1 and P_2 are corrupted.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. This is a trivial case. Since both P_1 and P_2 are controlled by the adversary $\mathcal{A}$, no message is simulated by $\mathcal{S}$.

This concludes the proof. ■

B Security Proof of Theorem. 2

Proof: To prove Thm. 2, we construct a simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$ where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$-hybrid model and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$ where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the ideal world, and corrupted parties are controlled by the simulator $\mathcal{S}$. We consider following cases.

Case 1: P_1 is corrupted; P_2 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{InputNotify}, \mathsf{sid}, P_2)$ for the honest P_2 from the external $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$, the simulator $\mathcal{S}$ sends $(\textsc{Prepare-II}, \mathsf{sid})$ to $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_2. $\mathcal{S}$ then picks random $a_2 \leftarrow \mathbb{G}, b_2 \leftarrow \mathbb{M}$ and sends (a_2, b_2) to P_1 behave of P_2.

- When the dummy party P_1 in the ideal world receives $(\textsc{Input-II}, \mathsf{sid}, g_1, h_1)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\textsc{Input-II}, \mathsf{sid}, g_1, h_1)$ to the external ideal functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.
Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $a_2 \leftarrow \mathbb{G}, b_2 \leftarrow \mathbb{M}$ to P_1 instead of $a_2 := g_2 \cdot u_2^{-1}$ and $b_2 := g_2 \cdot (h_2 - v_2)$.

Claim 3 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $u_2 \in \mathbb{G}, v_2 \in \mathbb{M}$ are uniformly random, the distribution of a_2 and b_2 are also uniformly random regardless g_2, h_2. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \ .$$

Case 2: P_2 is corrupted; P_1 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{InputNotify}, \mathsf{sid}, P_1)$ for the honest P_1 from the external $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$, the simulator $\mathcal{S}$ sends $(\textsc{Prepare-II}, \mathsf{sid})$ to $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_1. $\mathcal{S}$ then picks random $a_1 \leftarrow \mathbb{G}, b_1 \leftarrow \mathbb{M}$ and sends (a_1, b_1) to P_2 behave of P_1.

- When the dummy party P_2 in the ideal world receives $(\textsc{Input-II}, \mathsf{sid}, g_2, h_2)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\textsc{Input-II}, \mathsf{sid}, g_2, h_2)$ to the external ideal functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.
Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $a_1 \leftarrow \mathbb{G}, b_1 \leftarrow \mathbb{M}$ to P_1 instead of $a_1 := g_1 \cdot u_1^{-1}$ and $b_1 := g_1 \cdot (h_1 - v_1)$ to P_2.

Claim 4 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $u_1 \in \mathbb{G}, v_1 \in \mathbb{M}$ are uniformly random, the distribution of a_1 and b_1 are also uniformly random regardless g_1 and h_1. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{II}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \ .$$

Case 3: Both P_1 and P_2 are corrupted.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. This is a trivial case. Since both P_1 and P_2 are controlled by the adversary $\mathcal{A}$, no message is simulated by $\mathcal{S}$.

This concludes the proof. ■

C Security Proof of Theorem. 3

Proof: To prove Thm. 3, we construct a simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\text{EXEC}^{\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi^{\text{III}}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$ where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi^{\text{III}}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the $\mathcal{F}_{\text{GM}}^{\text{offline}}[\mathbb{G},\mathbb{M}]$-hybrid model and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\text{EXEC}_{\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$ where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\text{GM}}[\mathbb{G},\mathbb{M}]$ in the ideal world, and corrupted parties are controlled by the simulator $\mathcal{S}$. We consider following cases.

Case 1: P_1 is corrupted; P_2 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\text{InputNotify}, \mathsf{sid}, P_2)$ for the honest P_2 from the external $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$, the simulator $\mathcal{S}$ sends $(\text{Prepare-III}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_2. $\mathcal{S}$ then picks random $a_2 \leftarrow \mathbb{G}$ and sends a_2 to P_1 behave of P_2.

- When the dummy party P_1 in the ideal world receives $(\text{Input-III}, \mathsf{sid}, h_1)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\text{Input-III}, \mathsf{sid}, h_1)$ to the external ideal functionality $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.
Hybrid $\mathcal{H}_0$**:** It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi_{\mathsf{GM}}^{\mathrm{III}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$**:** $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $a_2 \leftarrow \mathbb{G}$ to P_1 instead of $a_2 := h_2 - w_2$.

Claim 5 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $w_2 \in \mathbb{M}$ is uniformly random, the distribution of a_2 is also uniformly random regardless h_2. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi_{\mathsf{GM}}^{\mathrm{III}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \ .$$

Case 2: P_2 is corrupted; P_1 is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ as well as honest P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\text{InputNotify}, \mathsf{sid}, P_1)$ for the honest P_1 from the external $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$ and a_2 from P_2, the simulator $\mathcal{S}$ sends $(\text{Prepare-III}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$ on behave of P_1. $\mathcal{S}$ then picks random $a_1 \leftarrow \mathbb{G}$ and sends a_1 to P_2 behave of P_1.

- When the dummy party P_2 in the ideal world receives $(\text{Input-III}, \mathsf{sid}, h_2)$ from the environment $\mathcal{Z}$, $\mathcal{S}$ directly forwards $(\text{Input-III}, \mathsf{sid}, h_2)$ to the external ideal functionality $\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. The indistinguishability is proven through hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.

Protocol $\Pi_{\mathsf{OT2}}^{n,\ell}$

Common input: $\mathbb{Z}_n, \mathbb{Z}_{2l}$
P_1**'s input:** $i \in \mathbb{Z}_n$
P_2**'s input:** $(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{2l}^n$

- **Offline**: P_0 generate randomly $\tilde{i} \in \mathbb{Z}_n$ and $(\tilde{x}_0, \ldots, \tilde{x}_{n-1}) \in \mathbb{Z}_{2l}^n$, and compute $u := \tilde{x}_{\tilde{i}}$. Then P_0 send $\tilde{i}, u$ to P_1 and send $(\tilde{x}_0, \ldots, \tilde{x}_{n-1})$ to P_2;
- **Online**: P_1 compute $\delta i := i - \tilde{i} \in \mathbb{Z}_n$, and sent δi to P_2;
- P_2 compute $\delta x := \mathsf{L\text{-}Shift}_{\delta i}(x) - \tilde{x} \in \mathbb{Z}_{2l}^n$ and send δx to P_1;
- P_1 compute $v := (\delta x)_{\tilde{i}} + u_1$, and return v.

Figure 18: The 2-round OT Protocol $\Pi_{\mathsf{OT2}}^{n,\ell}$

Protocol $\Pi_{\mathsf{S\text{-}OT2}}^{n,\ell}$

Common input: $\mathbb{Z}_n, \mathbb{Z}_{2l}$
P_1**'s input:** $i \in \mathbb{Z}_n$
P_2**'s input:** $(x_0, \ldots, x_{n-1}) \in \mathbb{Z}_{2l}^n$

- **Offline**: P_0 generate randomly $\tilde{i} \in \mathbb{Z}_n$ and $(\tilde{x}_0, \ldots, \tilde{x}_{n-1}) \in \mathbb{Z}_{2l}^n$, and $u_1 \in \mathbb{Z}_{2l}$, and compute $u_2 := \tilde{x}_{\tilde{i}} - u_1$. Then P_0 send $\tilde{i}, u_1$ to P_1 and send $(\tilde{x}_0, \ldots, \tilde{x}_{n-1})$, u_2 to P_2;
- **Online**: P_1 compute $\delta i := i - \tilde{i} \in \mathbb{Z}_n$, and sent δi to P_2;
- P_2 compute $\delta x := \mathsf{L\text{-}Shift}_{\delta i}(x) - \tilde{x} \in \mathbb{Z}_{2l}^n$ and send δx to P_1.
- P_1 compute $v_1 := (\delta x)_{\tilde{i}} + u_1$, and return v_1;
- P_2 return u_2.

Figure 19: The 2-round Shared OT Protocol $\Pi_{\mathsf{S\text{-}OT2}}^{n,\ell}$

Hybrid $\mathcal{H}_0$**:** It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi_{\mathsf{GM}}^{\mathrm{III}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}}$.
Hybrid $\mathcal{H}_1$**:** $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends a random $a_1 \leftarrow \mathbb{G}$ to P_1 instead of $a_1 := u^{-1} \cdot (h_1 - w_1 + a_2)$ to P_2.

Claim 6 *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof: Since $u \in \mathbb{G}$ is uniformly random, the distribution of a_1 is also uniformly random regardless h_1 and a_2. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. ■

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}}$. Therefore, we have

$$\text{EXEC}^{\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]}_{\Pi_{\mathsf{GM}}^{\mathrm{III}}[\mathbb{G},\mathbb{M}],\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{F}_{\mathsf{GM}}[\mathbb{G},\mathbb{M}],\mathcal{S},\mathcal{Z}} \ .$$

Case 3: Both P_1 and P_2 are corrupted.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\mathsf{GM}}^{\mathsf{offline}}[\mathbb{G},\mathbb{M}]$.

Indistinguishability. This is a trivial case. Since both P_1 and P_2 are controlled by the adversary $\mathcal{A}$, no message is simulated by $\mathcal{S}$.

This concludes the proof. ■

Protocol $\Pi^{2,p}_{\mathsf{convert}}$

Common input: $\mathbb{Z}_p$
P_1's input: $x_1 \in \mathbb{Z}_2$
P_2's input: $x_2 \in \mathbb{Z}_2$

Offline phase:

- S generates random $u \in \mathbb{Z}_2$, then splits u into $(u_1, u_2) \in \mathbb{Z}_2 \times \mathbb{Z}_2$ and $I(u) \in \mathbb{Z}_p$ into $(b_1, b_2) \in \mathbb{Z}_p \times \mathbb{Z}_p$, and finally sends (u_1, b_1) to P_1, send (u_2, b_2) to P_2 respectively;

Online phase:

- P_1 computes $z_1 := a_1 - u_1$ locally.
- P_2 computes $z_2 := a_2 - u_2$ locally.
- P_1 and P_2 reconstruct $z := a - u$ by interchanging z_1 and z_2;
- P_1 computes $y_1 := (-1)^z b_1 \in \mathbb{Z}_p$.
- P_2 computes $y_2 := (-1)^z b_2 + z \in \mathbb{Z}_p$.
- P_1 outputs y_1 and P_2 outputs y_2.

Figure 20: The share conversion protocol $\Pi^{2,p}_{\mathsf{convert}}$

D Two-round (shared) OT

We introduce a protocol of two online round for OT in Figure 18. Its offline communication is ℓ bits, and its online communication is $n\ell + \log n$ bits in 2 round.

It is easy to be modified to a protocol for shared OT of two online round (Figure 19). Its offline communication is ℓ bits, and its online communication is $n\ell + \log n$ bits in 2 round.

E Share Conversion $\mathbb{Z}_2 \to \mathbb{Z}_p$

In this section, we recap the share conversion protocol $\Pi^{2,p}_{\mathsf{convert}}$ proposed in [Cou18] in server-aid model. In the share conversion protocol $\Pi^{2,p}_{\mathsf{convert}}$, P_1 and P_2 hold shares of $x \in \mathbb{Z}_2$ over $\mathbb{Z}_2$, and after the protocol execution, they obtains shares of x over $\mathbb{Z}_p$. Let

$$I : \mathbb{Z}_2 \mapsto \mathbb{Z}_p$$

be the module transform map defined by $I(0) := 0$ and $I(1) = 1$. The protocol is described in Fig. 20.

This protocol needs 2 rounds, and its communication is $2(\log p + 2)$ bits. However if we use the PRF improvement, we need $\log p$ bits communication in offline phase, and 2 bits communication in 1 round in online phase per calling as shown in Table. 1.1.