Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 6039034, 26 pages
https://doi.org/10.1155/2023/6039034

Research Article

Efficient Secure Computation from SM Series Cryptography

Yibiao Lu,[1,2] **Zecheng Wu**,[1,2] **Bingsheng Zhang**,[1,2] **and Kui Ren**[1,2]

[1]*ZJU-Hangzhou Global Scientific and Technological Innovation Center, China*
[2]*Zhejiang University, China*

Correspondence should be addressed to Bingsheng Zhang; bingsheng@zju.edu.cn

Received 26 July 2022; Revised 22 November 2022; Accepted 15 April 2023; Published 17 May 2023

Academic Editor: Nan Li

The wireless network suffers from many security problems, and computation in a wireless network environment may fail to preserve privacy as well as correctness when the adversaries conduct attacks through backdoors, steganography, kleptography, etc. Secure computation ensures the execution security in such an environment, and compared with computation on the plaintext, the performance of secure computation is bounded by the underlying cryptographic algorithms and the network environment between the involved parties. Besides, the Chinese cryptography laws require the cryptographic algorithms that appeared in the commercial market to be authorized. In this work, we show how to implement oblivious transfer (OT), an important primitive in secure multiparty computation (MPC), using the Chinese government-approved SM2 and SM3 algorithms. The SM2 algorithm is based on the elliptic curve cryptography and is much faster than the discrete logarithm-based solutions. Moreover, by adopting the standard OT extension technique, we can extend the number of OTs efficiently with one more round of communication and invocations to the SM3 and SM4 algorithms. The OT primitive can be used in the Beaver multiplication triple generation and other MPC protocols, e.g., private set intersection. Therefore, we can utilize the SM series cryptography, specifically, the SM2, SM3, and SM4 algorithms, to build highly efficient secure computation frameworks which are suitable for the wireless network environment and for commercial applications in China. The experimental evaluation results show that our protocols have comparable performance to existing protocols; specifically, our protocols are quite suitable for bad network environments.

1. Introduction

Wireless network (WLN) enables devices to communicate with each other without cable connections, and it is a major component of the modern Internet. With the advancement of the Internet of things (IoT) technique, a vast amount of wireless networks are being deployed [1]. However, the wireless network can be quite vulnerable [2, 3], an adversary may eavesdrop on or alter the communication in the network. When several parties want to perform a joint computation in a wireless network with potential adversaries, the correctness of the computation and the privacy of inputs can be easily breakdown. Although efforts have been made to avoid or mitigate the security threat in the wireless network [4–6], there are still a lot of security issues. Therefore, we need privacy-enhancing technologies to ensure security in such a network environment.

In this work, we leverage a cryptographic technique called secure multiparty computation (MPC) [7, 8] to construct an efficient and provably secure computation framework that protects parties' privacy in a network potentially controlled by the adversary. The goal of MPC is to design protocols that enable several mutually untrusted parties to jointly compute a function on their private inputs without revealing anything except for the function output. Typically, the computational security of a MPC protocol relies on some computational or setup assumptions. Therefore, if there is an adversary that can break the security of the protocol, either the adversary has unbounded computation power or the computational assumption does not hold.

We focus on the oblivious transfer (OT) primitive, which is complete for the MPC computation [9] and is a fundamental building block for many MPC protocols. Naor and Pinkas [10] provide the first efficient OT protocol, and Chou

and Orlandi [11] propose the simplest OT construction. Specifically, in [12], Masny and Rindal initiate the study of a new OT primitive called endemic OT. The endemic OT is weaker than commonly used OT in the sense that a corrupted party is able to control its output message, and the same definition has also been considered by Garg et al. [13]. As demonstrated by Masny and Rindal [12], endemic OT can be constructed using proper key agreement protocols in the programmable random oracle model. Later, McQuoid et al. [14] improve the efficiency of 1-out-of-N endemic OT protocol using a programmable-once public function (POPF). They also notice some security issues in the batch setting and provide a proper treatment in [15].

In the implementation, the main factor affecting the performance of a MPC protocol is the involved cryptographic primitives, whose performance depends on the underlying cryptographic assumptions. A wide range of computational MPC protocols are built on asymmetric-key cryptosystems, and as for asymmetric-key cryptography, the Rivest-Shamir-Adleman (RSA) based on integer factorization and elliptic curve cryptography (ECC) based on discrete logarithm are two of the most important algorithms being used. Evaluation results show that ECC has great advantages over RSA in both computation time [16] and resource consumption [17]. In 2010, the Chinese State Cryptography Administration announced the public key cryptographic algorithm SM2 [18] and several other SM series algorithms, which are based on the elliptic curve cryptography. According to the Chinese cryptography laws [19], it is mandatory to adopt cryptography algorithms which have been authorized for commercial use in China, e.g., the SM algorithms.

1.1. Our Contribution. The contributions of this work can be summarized as follows. First, we construct an endemic OT protocol based on the SM2 key agreement protocol. Moreover, we build several MPC protocols on the top of the endemic OT protocol, including a two-party secure computation protocol on the Boolean circuits, a multiparty party secure computation protocol on arithmetic circuits, and a two-party private set intersection (PSI) protocol. Our constructions consider both efficiency and availability and only use the SM series cryptography. The security of our protocols can be proved in the random oracle model and in the public key infrastructure (PKI) setting. Since a PKI is used, the parties can communicate without the secure channel functionality, and in implementation, the parties can transmit messages without the TLS protocol. To the best of our knowledge, we are the first to propose a secure computation framework that complies with the Chinese national standards and regulations.

2. Related Work

There have been some secure computation solutions for the wireless network. We can categorize them into low-communication MPC and hardware-based computation.

Garay et al. [20] investigated the feasibility of designing protocols with sublinear communication complexity. Their work enables large-scale secure computation in a communication-restricted environment. Moreover, Gentry et al. proposed a communication model called YOSO [21], in which each party only sends one message to others. In YOSO MPC, only a small fraction of parties execute and communicate in each round; therefore, its communication complexity is also sublinear to the total amount of involved parties. Fully homomorphic encryption is a widely used primitive in low-communication MPC protocol design. Asharov et al. [22] constructed a MPC protocol using threshold FHE. The proposed protocol only needs two or three communication rounds depending on the underlying assumption, and its communication cost is independent of the function to be computed. López-Alt et al. [23] and Mukherjee and Wichs [24] considered to use multikey FHE and a third-party untrusted server in their construction, the proposed protocol achieves a minimal communication complexity which is independent of the function and the number of parties. All these FHE-based protocols have the property that the communication size only depends on the input/output size; however, using FHE dramatically increases the computation burden. Another way to reduce communication in the protocol execution is to offload the major communication workload to a preprocessing phase. Damgård et al. proposed the celebrated SPDZ computation framework [25], in which a bunch of authenticated triples are generated in the preprocessing phase and consumed in the online phase. The authenticated garbling protocols proposed by Wang et al. [26, 27] also use authenticated triples to speed up the online computation. Specifically, these garbled circuit-based protocols have constant round complexity, which makes them more suitable for the wireless network. Carter et al. [28] noticed that a remote server can be used to instantiate the preprocessing phase. However, in such a server-aided setting, the adversaries are allowed to corrupt the cloud server.

Hardware-based computation has many things in common with the server-aided computation, and the efficiency of the protocols varies with the parties' trust to the hardware. Since the hardware can have a fast connection with the computing parties, hardware-based computation is typically much faster than other solutions [29–31]. Generally, hardware-based computation uses a hardware token or trusted hardware issued by the parties or a hardware manufacturer, and the parties can use the hardware to generate preprocessing information or even directly compute the function. These hardware-based protocols assume that the hardware is tamper-resistant or tamper-proof, while in practice, there have been works that successfully break the security of some commonly used hardware [32, 33].

3. Preliminaries

3.1. Notations. Throughout this paper, we use the following notations and terminologies. Let λ be the computational security parameter, and μ be the statistical security parameter. Denote a binary matrix with a rows and b columns as $\{0,1\}^{a\times b}$. When A and B are two-bit strings, $A\|B$ is the concatenation of them. When A is a bit string, a vector, or an array, $A[i]$ is its i-th element. Denote the set $\{a, a+1, \cdots, b\}$ by $[a, b]$, let $[b]$ denote $[1, b]$, and let $\varnothing$ denote the empty set. When a set $A = \{a_i\}_{i\in[n]}$ is used, we assume the elements are

arranged by the indexes as $a_1, \cdots, a_n$. When A is a set, $a \longleftarrow A$ stands for sampling a uniformly at random from A, and $|A|$ stands for the size of A in terms of the number of elements. When A is a matrix, A^i denotes its i-th column, and A_j denotes its j-th row. When A is a randomized algorithm, $y \longleftarrow A(x)$ 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. Let poly$(\cdot)$ and negl$(\cdot)$ be a polynomially bounded function and negligible function, respectively. We assume each party has a unique PID. For readability, we refer P_i as the PID for the party P_i. We abbreviate "probabilistic polynomial time" as PPT and "interactive Turing machine" as ITM.

3.1.1. Elliptic Curve Cryptography Notation. In this work, we work on the finite field $\mathbb{F}_p$, and the elliptic curve E is defined by two elements $a, b \in \mathbb{F}_p$. The set of all the points on E is denoted as $E_p(a, b)$. $g := (x_g, y_g)$ is the base point of E with order n, and $h := |E_p(a, b)|/n$ is the cofactor.

3.2. Security Definition. Our security model is based on the universal composability (UC) framework [34], which lays down a solid foundation for designing and analyzing protocol secure against attacks in an arbitrary network execution environment (therefore, it is also known as a network-aware security model). We refer to the original work [34] for details.

Roughly speaking, in the UC framework, protocols are carried out over multiple interconnected machines; to capture attacks, a network adversary $\mathscr{A}$ is introduced, which is allowed to partially control the communication network and corrupt some machines (i.e., have full control of all physical parts of some machines). Then, a protocol Π is a UC-secure implementation of a functionality $\mathscr{F}$, if it satisfies that for every network adversary $\mathscr{A}$ attacking an execution of Π, there is another adversary $\mathscr{S}$—known as the simulator—-attacking the ideal process that uses $\mathscr{F}$ (by corrupting the same set of machines) such that the executions of Π with $\mathscr{A}$ and that of $\mathscr{F}$ with $\mathscr{S}$ make no difference to any network execution environment $\mathscr{Z}$.

3.2.1. The Ideal World. In the ideal world, $P_1, \cdots, P_N$ only communicate with an ideal functionality $\mathscr{F}^f_{\text{mpc}}$ during the execution. As depicted in Figure 1, $\mathscr{F}^f_{\text{mpc}}$ waits for each party to provide input, and when all parties' inputs have been received, it computes the function $(y_1, \cdots, y_N) \longleftarrow f(x_1, \cdots, x_N)$ and sends the output y_i to the party P_i, for $i \in [N]$. Besides, the functionality $\mathscr{F}^f_{\text{mpc}}$ interacts with the simulator $\mathscr{S}$. When a party P_i sends its input x_i to $\mathscr{F}^f_{\text{mpc}}$, $\mathscr{S}$ receives a notification (ComputeNotify, sid, P_i). Before $\mathscr{F}^f_{\text{mpc}}$ outputs, it sends (Output, sid) to ask for permission of $\mathscr{S}$, and it only sends y_i to P_i if a (Deliver, sid, P_i) is received.

3.2.2. Adversary Models. There are two main adversary models. A semihonest adversary follows the protocol description and a malicious adversary can deviate from the protocol description arbitrarily. Both adversaries try to help the environment distinguish between the ideal world and the real world by learning more information from the protocol execution.

3.2.3. Model of Protocol Execution. In the protocol execution, an environment $\mathscr{Z}$ provides inputs to the parties and receives outputs from them. Moreover, it can interact with the adversary $\mathscr{A}$ freely. At the end of the protocol, $\mathscr{Z}$ outputs a binary variable. Let $\text{exec}_{\Pi,\mathscr{A},\mathscr{Z}}(i)$ denote the output variable of $\mathscr{Z}$ in an execution of protocol Π with environment $\mathscr{Z}$ and adversary $\mathscr{A}$ on input i, and $\text{exec}_{\Pi,\mathscr{A},\mathscr{Z}}$ denote the ensemble $\{\text{exec}_{\Pi,\mathscr{A},\mathscr{Z}}(i)\}_{i\in\{0,1\}^*}$. We use $\text{exec}^{\mathscr{F}}_{\Pi,\mathscr{A},\mathscr{Z}}$ when protocol Π is in the $\mathscr{F}$-hybrid model, i.e., $\mathscr{F}$ can be invoked in Π. We slightly abuse notation and use $\text{exec}_{\mathscr{F},\mathscr{S},\mathscr{Z}}$ for the ideal execution.

3.2.4. Random Oracle. A random oracle [35] is an idealized hash function that can be publicly accessed. In the random oracle model, the random oracle maintains a table of the previous queries. For a query with input x, the random oracle first checks if x is recorded. For an unrecorded x, the random oracle chooses an element from its output domain uniformly at random and responds with this element, and it then records x and the corresponding response; for a recorded x, the random oracle simply responds with the recorded response.

3.2.5. Public Key Infrastructure. A public key infrastructure (PKI) links a party's public identity with its public key. In this work, we use a PKI to guarantee the authenticity and validity of a party's public key and further ensure the security of the communication. In such a PKI setting, the parties can communicate with each other without the requirement of an underlying secure channel functionality [36]. In implementation, we allow the parties to send messages in an insecure network environment that may be eavesdropped on or tampered with without the TLS protocol.

3.3. One-Round Key Agreement Protocol. Key agreement (KA) protocols allow two parties A and B to jointly establish a key known to no one else. We use a similar notation as in [14], which considers two-round key agreement protocols, while we focus on one-round key agreement protocols. We first provide an illustrative example that is provided in Figure 2, and the protocol involves the following parameters:

(i) KA.$\mathscr{R}$ is the set of randomness used by the parties

(ii) KA.Msg_1 and KA.Msg_2 are A's message generation function and B's message generation function, respectively

(iii) KA.$\mathscr{M}_1$ and KA.$\mathscr{M}_2$ are the set of A's protocol message and the set of B's protocol message, respectively

(iv) KA.Key_1 and KA.Key_2 are A's key generation function and B's key generation function, respectively

(v) KA.$\mathscr{K}$ is the set of output keys

Functionality $\mathcal{F}^{f}_{\mathsf{mpc}}$

It interacts with players $\mathcal{P} := \{P_1, \ldots, P_N\}$ and the adversary $\mathcal{S}$. Let $\mathcal{P}_c$ be the set of corrupted parties. Initially, set $\mathcal{P}_c = \emptyset$.

Compute:

(i) Upon receiving $(\textsc{Compute}, \mathsf{sid}, x_i)$ from party $P_i \in \mathcal{P}$:
  (i) Send a notification $(\textsc{ComputeNotify}, \mathsf{sid}, P_i)$ to $\mathcal{S}$;
(ii) If it has received x_i from P_i for $i \in [N]$:
  (i) Compute $(y_1, \ldots, y_N) \leftarrow f(x_1, \ldots, x_N)$;
  (ii) Send $(\textsc{Output}, \mathsf{sid})$ to $\mathcal{S}$;
  (iii) For $i \in [N]$, upon receiving $(\textsc{Deliver}, \mathsf{sid}, P_i)$ from $\mathcal{S}$ and send $(\textsc{Compute}, \mathsf{sid}, y_i)$ to P_i;

Corruption handling:

(i) Upon receiving $(\textsc{Corrupt}, \mathsf{sid}, P_i)$ from the adversary $\mathcal{S}$, if $P_i \in \mathcal{P}$:
  (i) Set $\mathcal{P}_c := \mathcal{P}_c \cup \{P_i\}$;

Figure 1: Secure multiparty computation functionality $\mathcal{F}^{f}_{\mathrm{mpc}}$.

A		B
$t_A \leftarrow \mathsf{KA}.\mathcal{R}$		$t_B \leftarrow \mathsf{KA}.\mathcal{R}$
$m_A \leftarrow \mathsf{KA.Msg}_1(t_A)$		$m_B \leftarrow \mathsf{KA.Msg}_2(t_B)$
	$\xrightarrow{m_A}$	
	$\xleftarrow{m_B}$	
$k_A \leftarrow \mathsf{KA.Key}_1(t_A, m_B)$		$k_B \leftarrow \mathsf{KA.Key}_2(t_B, m_A)$

Figure 2: An illustrative example of one-round key agreement protocol.

In a one-round key agreement protocol Π, party A picks random $t_A \longleftarrow \mathrm{KA}.\mathscr{R}$ and computes $m_A \longleftarrow \mathrm{KA.Msg}_1(t_A)$, and it sends m_A to B; in the meanwhile, party B also picks random $t_B \longleftarrow \mathrm{KA}.\mathscr{R}$ and computes $m_B \longleftarrow \mathrm{KA.Msg}_2(t_B)$, and it sends m_B to A. At the end, A and B establish the same key by computing $k_A \longleftarrow \mathrm{KA.Key}_1(t_A, m_B)$ and $k_B \longleftarrow \mathrm{KA.Key}_2(t_B, m_A)$, respectively. Throughout the protocol, the computational security parameter λ is used implicitly as a parameter of the algorithms.

We require the protocol to have the following properties:

Definition 1 (Correctness). A one-round KA protocol Π is correct if for any $t_A, t_B \longleftarrow \mathrm{KA}.\mathscr{R}$, and $m_A \longleftarrow \mathrm{KA.Msg}_1(t_A), m_B \longleftarrow \mathrm{KA.Msg}_2(t_B)$,

$$\Pr[k_A = \mathrm{KA.Key}_1(t_A, m_B) = \mathrm{KA.Key}_2(t_B, m_A) = k_B] = 1 - \mathrm{negl}(\lambda). \tag{1}$$

Definition 2 (Security). A one-round KA protocol Π is secure if for any PPT distinguisher $\mathscr{D}$,

$$\left| \Pr\left[\begin{array}{c} t_A, t_B \longleftarrow \mathrm{KA}.\mathscr{R}\,; \\ m_A \longleftarrow \mathrm{KA.Msg}_1(t_A)\,; m_B \longleftarrow \mathrm{KA.Msg}_2(t_B)\,; \\ k_1 \longleftarrow \mathrm{KA.Key}_1(t_A, m_B)\,; k_2 \longleftarrow \mathrm{KA}.\mathscr{K}\,; \\ b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathscr{D}(k_b, m_A, m_B)\colon b = b^* \end{array}\right] - \frac{1}{2} \right| = \mathrm{negl}(\lambda). \tag{2}$$

Definition 3 (Uniformity). A one-round KA protocol Π is Msg_1-uniform if for any PPT distinguisher $\mathscr{D}$,

$$\left| \Pr\left[\begin{array}{c} t_A \longleftarrow \mathrm{KA}.\mathscr{R}\,; m_1 \longleftarrow \mathrm{KA.Msg}_1(t_A)\,; m_2 \longleftarrow \mathrm{KA}.\mathscr{M}_1\,; \\ b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathscr{D}(m_b)\colon b = b^* \end{array}\right] - \frac{1}{2} \right| = \mathrm{negl}(\lambda). \tag{3}$$

Likewise, we can define Msg_2 uniformity.

Definition 4 (Robustness). A one-round KA protocol Π is robust if for any PPT distinguisher $\mathscr{D}$,

$$\left| \Pr\left[\begin{array}{c} t_A, t_B \longleftarrow \mathrm{KA}.\mathscr{R}\,; m_A \longleftarrow \mathrm{KA.Msg}_1(t_A)\,; m_B \longleftarrow \mathrm{KA.Msg}_2(t_B)\,; \\ (m_B^*, \mathrm{state}) \longleftarrow \mathscr{D}(m_A)\,; k^* \longleftarrow \mathrm{KA.Key}_1(t_A, m_B^*)\,; \\ k_1 \longleftarrow \mathrm{KA.Key}_2(t_A, m_B)\,; k_2 \longleftarrow \mathrm{KA}.\mathscr{K}\,; \\ b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathscr{D}(\mathrm{state}, k_b, k^*, m_B)\colon b = b^* \end{array}\right] - \frac{1}{2} \right| = \mathrm{negl}(\lambda). \tag{4}$$

3.4. Programmable-Once Public Function. The primitive programmable-once public function (POPF) is proposed by McQuoid et al. [14]. Later in [15], they fixed some issues in the definition and formally defined a batch 2-POPF. In our endemic OT protocol, we use a N-POPF, and in its

multi-instance variant, we use a batch N-POPF; therefore, we provide a formal definition of a batch N-POPF here.

A batch N-POPF consists of two algorithms: Program : $[N] \times \mathcal{N} \longrightarrow \mathcal{M}$ and Eval : $\mathcal{M} \times [N] \longrightarrow \mathcal{N}$. Programmable-once means that one can compute $\phi = \text{Program}(x, y)$ for an $x \in [N]$ and $y \in \mathcal{N}$, but for any other $x' \neq x$, the value of $y' \in \mathcal{N}$ such that $\text{Program}(x', y') = \phi$ should be unpredictable, i.e., y' looks like random. This unpredictability is defined with respect to a 1-weak random oracle $F : \mathcal{N} \longrightarrow \mathcal{O}$ which produces a pseudorandom $y := F(x)$ when F is only allowed to be accessed once.

Definition 5 (1-weak random oracle). A function $F : \mathcal{N} \longrightarrow \mathcal{O}$ is a 1-weak random oracle if for any PPT distinguisher $\mathcal{D}$,

$$\left|\Pr\left[x \longleftarrow \mathcal{N}\,; y_0 := F(x)\,; y_1 \longleftarrow \mathcal{O}\,; b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathcal{D}(x, y_b) : b = b^*\right] - \frac{1}{2}\right| = \text{negl}(\lambda). \tag{5}$$

$\mathcal{D}$ can only access F through this experiment.

Now we can formally define a batch N-POPF. Generally, a batch N-POPF makes use of some local setups $\mathcal{H}$, which can consist of random oracles, common reference strings, etc. We use $\text{Program}^{\mathcal{H}}$ and $\text{Eval}^{\mathcal{H}}$ to denote the algorithms when they access $\mathcal{H}$. Besides, a batch N-POPF should include two alternative local setups:

$\mathcal{H}_{\text{Sim}}$: this setup provides the same interface as $\mathcal{H}$ and an additional method Sim : $\mathcal{N}^N \longrightarrow \mathcal{M}$.

$\mathcal{H}_{\text{Extract}}$: this setup provides the same interface as $\mathcal{H}$ and an additional method Extract : $\mathcal{M} \longrightarrow [N]$.

We require the batch N-POPF to have the following properties:

Definition 6 (Correctness). A batch N-POPF is correct if for any $x \in [N], y \in \mathcal{N}$,

$$\Pr[\text{Eval}(\text{Program}(x, y), x) = y] = 1 - \text{negl}(\lambda). \tag{6}$$

Definition 7 (Honest simulation). A batch N-POPF has honest simulation if for any PPT distinguisher $\mathcal{D}$ and PPT adversary $\mathcal{A}$,

$$\left|\Pr\left[\begin{array}{c} x \longleftarrow [N]\,; (s, y) \longleftarrow \mathcal{A}(\quad)\,; \phi^0 \longleftarrow \text{Program}(x, y), \text{for} i \in [N], r_i^0 := \text{Eval}(\phi^0, i)\,; \\ r_x^1 := y, \text{for} i \neq x, r_i^1 \longleftarrow \mathcal{N}, \phi^1 \longleftarrow \text{Sim}(r_1, \cdots, r_N)\,; \\ b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathcal{D}^{\mathcal{H}_{\text{Sim}}}\left(s, \phi^b, \left\{r_i^b\right\}_{i \in [N]}\right) : b = b^* \end{array}\right] - \frac{1}{2}\right| = \text{negl}(\lambda). \tag{7}$$

The honest simulation property captures the batch N-POPF's ability of hiding x: when $b = 0$, ϕ^0 is generated using x and when $b = 1$, ϕ^1 is generated from random $\{r_i^1\}_{i \in [N]}$. If ϕ^0 and ϕ^1 are indistinguishable even when $\{r_i^1\}_{i \in [N]}$ is given, we can say that ϕ does not leak the information of x.

Definition 8 (Uncontrollable outputs). A batch N-POPF has uncontrollable outputs if for any 1-weak random oracle F any PPT distinguisher $\mathcal{D}$ and PPT adversary $\mathcal{A}$,

$$\left|\Pr\left[\begin{array}{c} (\phi, \text{state}) \longleftarrow \mathcal{A}^{\mathcal{H}_{\text{Extract}}}\,; \\ x := \text{Extract}(\phi), \text{for} i \neq x, r_i^0 := F(\text{Eval}(\phi, i))\,; \\ \text{for} i \neq x, r_i^1 \longleftarrow \mathcal{N}\,; \\ b \longleftarrow \{0,1\}\,; b^* \longleftarrow \mathcal{D}\left(\text{state}, \left\{r_i^b\right\}_{i \neq x}\right) : b = b^* \end{array}\right] - \frac{1}{2}\right| = \text{negl}(\lambda). \tag{8}$$

The uncontrollable output property restricts the adversary to only be able to program once. Given any ϕ produced by the adversary $\mathcal{A}$, the Extract method finds an x such that for $i \neq x$, the value of $\text{Eval}(\phi, i)$ is unpredictable.

Moreover, when the batch N-POPF has an honest simulation and uncontrollable outputs, and the interface of $\mathcal{H}_{\text{Sim}}$ and $\mathcal{H}_{\text{Extract}}$ looks indistinguishable from the adversary, we can say that the batch N-POPF is secure.

In this work, we use a correct and secure batch N-POPF which is constructed by McQuoid et al. in [15]. For simplicity, we denote $E_p(a, b)$ as $\mathbb{G}$. The hash functions $\{\text{hash}_i^{\mathbb{G}}\}_{i \in [N]}$ are defined as $\text{hash}_i^{\mathbb{G}} : \mathbb{G}^{N-1} \longrightarrow \mathbb{G}$ and are modeled as random oracles. We provide the details of the batch N-POPF in Figure 3, and we have the following theorem from [15]:

Theorem 9 (See [15]). *Figure 3 defines a correct and secure batch N-POPF.*

3.5. Oblivious Transfer. Oblivious transfer (OT) is a cryptographic primitive that allows a receiver Rec to choose and to obtain several messages from a bunch of messages held by a sender Sen, while Sen is not aware of Rec's choice, and Rec will not learn anything about the unchosen messages. The messages held by Sen can be contributed by itself or generated by the OT functionality. We denote an OT functionality where Sen decides the messages as sender OT $\mathcal{F}_{S\text{–OT}}$ and an OT functionality which generates messages for Sen as random OT $\mathcal{F}_{U\text{–OT}}$. In [10], Naor and Pinkas first provide an efficient implementation of $\mathcal{F}_{S\text{–OT}}$, and in [11], Chou and Orlandi propose the simplest OT protocol for

$\mathcal{N} := \mathbb{G}, \mathcal{M} := \mathbb{G}^N$

$\mathcal{H}$
  Record the history calls in the transcript $\mathcal{T}$
$\mathsf{hash}_i^{\mathbb{G}}(u)$:
  If there exists a v *s.t.* $(v = \mathsf{hash}_i^{\mathbb{G}}(u)) \in \mathcal{T}$:
    Output v
  Else:
    Output random $v \leftarrow \mathbb{G}$

$\mathsf{Program}(x, y)$:
  For $i \neq x$, pick random $r_i \leftarrow \mathbb{G}$
  Compute $r_x := y - \mathsf{hash}_x^{\mathbb{G}}(\{r_i\}_{i \neq x})$
  Output $\phi := \{r_i\}_{i \in [N]}$

$\mathsf{Eval}(\phi = \{r_i\}_{i \in [N]}, x)$:
  Output $r_x + \mathsf{hash}_x^{\mathbb{G}}(\{r_i\}_{i \neq x})$

$\mathcal{H}_{\text{SIM}}$
  Record the history calls in the transcript $\mathcal{T}$
  Allocate an empty array U
$\mathsf{hash}_i^{\mathbb{G}}(u)$:
  If there exists a v *s.t.* $(v = \mathsf{hash}_i^{\mathbb{G}}(u)) \in \mathcal{T}$:
    Output v
  Else if $U[i, u]$ has been defined:
    Output $U[i, u]$
  Else:
    Output random $v \leftarrow \mathbb{G}$
$\text{SIM}(r_1, \ldots, r_N)$:
  Pick random $\phi = (s_1, \ldots, s_N) \leftarrow \mathcal{M}$
  For $i \in [N]$, set $U[i, \{s_j\}_{j \neq i}] := r_i + s_i$
  Output ϕ

$\mathcal{H}_{\text{EXTRACT}}$
  Record the history calls in the transcript $\mathcal{T}$
  The interfaces of $\{\mathsf{hash}_i^{\mathbb{G}}\}_{i \in [N]}$ are the same as $\mathcal{H}$
$\text{EXTRACT}(\phi = \{r_i\}_{i \in [N]})$
  Find the first query in $\mathcal{T}$ *s.t.* $\mathsf{hash}_x^{\mathbb{G}}(\{r_i\}_{i \neq x})$ is in $\mathcal{T}$
    Output x
  If there are no such queries:
    Output 1

Figure 3: Batch N-way programmable-once public function from [15].

1-out-of-N Endemic Oblivious Transfer Functionality $\mathcal{F}_{\text{E-OT}}^{1,N}$

It interacts with players $\mathcal{P} := \{\mathsf{Sen}, \mathsf{Rec}\}$ and the adversary $\mathcal{S}$. It is parameterized by the length of the messages length. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Transfer:

(i) Upon receiving $(\text{SEND}, \mathsf{sid}, \mathsf{ssid})$ from Sen:
  (i) Send a notification $(\text{SENDNOTIFY}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\text{SEND}, \mathsf{sid}, \mathsf{ssid})$;
  (iii) Ignore future $(\text{SEND}, \mathsf{sid}, \mathsf{ssid})$ messages with the same $\mathsf{sid}, \mathsf{ssid}$;
(ii) Upon receiving $(\text{RECEIVE}, \mathsf{sid}, \mathsf{ssid}, c)$ from Rec, where c in $[N]$:
  (i) Send a notification $(\text{RECEIVENOTIFY}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\text{RECEIVE}, \mathsf{sid}, \mathsf{ssid}, c)$;
  (iii) Ignore future $(\text{RECEIVE}, \mathsf{sid}, \mathsf{ssid}, \ldots)$ messages with the same $\mathsf{sid}, \mathsf{ssid}$;
(iii) If both $(\text{SEND}, \mathsf{sid}, \mathsf{ssid})$ and $(\text{RECEIVE}, \mathsf{sid}, \mathsf{ssid}, c)$ are stored:
  (i) For $i \in [N]$, pick random $m_i \leftarrow \{0, 1\}^{\mathsf{length}}$;
  (ii) If $\mathsf{Sen} \in \tilde{\mathcal{P}}$, wait for $(\text{FIXMESSAGE}, \mathsf{sid}, \mathsf{ssid}, \{\tilde{m}_i\}_{i \in [N]})$ from $\mathcal{S}$, set $m_i := \tilde{m}_i$, for $i \in [N]$;
  (iii) If $\mathsf{Rec} \in \tilde{\mathcal{P}}$, wait for $(\text{FIXMESSAGE}, \mathsf{sid}, \mathsf{ssid}, \tilde{m}_c)$ from $\mathcal{S}$, set $m_c := \tilde{m}_c$;
  (iv) Send $(\text{SEND}, \mathsf{sid}, \mathsf{ssid}, \{m_i\}_{i \in [N]})$ to Sen and $(\text{RECEIVE}, \mathsf{sid}, \mathsf{ssid}, m_c)$ to Rec;

Corruption handling:

(i) Upon receiving $(\text{CORRUPT}, \mathsf{sid}, \mathsf{ssid}, P)$ from the adversary $\mathcal{S}$, if $P \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P\}$;
  (ii) Send $(\text{INPUT}, \mathsf{sid}, \mathsf{ssid}, P, x)$ to $\mathcal{S}$ if party P's input x is already defined;

Figure 4: 1-out-of-N endemic oblivious transfer functionality $\mathcal{F}_{E\text{-OT}}^{1,N}$.

$\mathcal{F}_{U\text{-OT}}$, and they show how to transform it into a $\mathcal{F}_{S\text{-OT}}$ protocol. However, both [10] and the random OT protocol in [11] do not provide full simulation-based security.

In this work, we make extensive use of a special notion of OT called 1-out-of-N endemic OT, which is proposed in [12]. Endemic OT is essentially the same as random OT, except that the endemic OT functionality allows the adversary $\mathcal{S}$ to determine the corrupted party's messages. As depicted in Figure 4, the 1-out-of-N endemic OT functionality $\mathcal{F}_{E\text{-OT}}^{1,N}$ waits for $(\mathsf{Send}, \mathsf{sid}, \mathsf{ssid})$ from Sen

Private Set Intersection Functionality $\mathcal{F}_{\text{PSI}}$

It interacts with players $\mathcal{P} := \{\text{Sen}, \text{Rec}\}$ and the adversary $\mathcal{S}$. It is parameterized by the set size of Sen and Rec, n_1 and n_2. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Compute:

(i) Upon receiving $(\textsc{Compute}, \text{sid}, \text{Sen}, X = (x_1, \ldots, x_{n_1}))$ from Sen:
  (i) Send a notification $(\textsc{ComputeNotify}, \text{sid}, \text{Sen})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Compute}, \text{sid}, \text{Sen}, X)$;
  (iii) Ignore future $(\textsc{Compute}, \text{sid}, \text{Sen}, \ldots)$ messages with the same sid;
(ii) Upon receiving $(\textsc{Compute}, \text{sid}, \text{Rec}, Y = (y_1, \ldots, y_{n_2}))$ from Rec:
  (i) Send a notification $(\textsc{ComputeNotify}, \text{sid}, \text{Rec})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Compute}, \text{sid}, \text{Rec}, Y)$;
  (iii) Ignore future $(\textsc{Compute}, \text{sid}, \text{Rec}, \ldots)$ messages with the same sid;
(iii) If both $(\textsc{Compute}, \text{sid}, \text{Sen}, X)$ and $(\textsc{Compute}, \text{sid}, \text{Rec}, Y)$ are stored:
  (i) Compute $\text{Res} := X \cap Y$;
  (ii) Send $(\textsc{Compute}, \text{sid})$ to Sen and $(\textsc{Compute}, \text{sid}, \text{Res})$ to Rec;

Corruption handling:

(i) Upon receiving $(\textsc{Corrupt}, \text{sid}, P)$ from the adversary $\mathcal{S}$, if $P \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P\}$;
  (ii) Send $(\textsc{Input}, \text{sid}, P, x)$ to $\mathcal{S}$ if party P's input x is already defined;

Figure 5: Private set intersection functionality $\mathcal{F}_{\text{PSI}}$.

and $(\text{Receive}, \text{sid}, \text{ssid}, c)$ from Rec, where $c \in [N]$ denotes Rec's choices. After both messages are obtained, $\mathcal{F}_{E\text{-OT}}^{1,N}$ picks n uniformly random messages $\{m_i\}_{i\in[N]}$. However, when the adversary $\mathcal{S}$ corrupts Sen, it is allowed to determine all the messages by sending $(\text{FixMessage}, \text{sid}, \text{ssid}, \{\tilde{m}_i\}_{i\in[N]})$, and when the adversary $\mathcal{S}$ corrupts Rec, it is allowed to determine the message m_c by sending $(\text{FixMessage}, \text{sid}, \text{ssid}, m_c)$. At the end, $\mathcal{F}_{E\text{-OT}}^{1,N}$ sends $\{m_i\}_{i\in[N]}$ to Sen and m_c to Rec. We also provide the standard random OT functionality $\mathcal{F}_{U\text{-OT}}^{1,N}$ in Appendix A.1.

3.6. Private Set Intersection. Private set intersection (PSI) is a specialized MPC problem. In PSI, two parties want to compute the intersection of their input sets, without revealing the content of their inputs. As described in Figure 5, the party Sen and Rec send their input sets X and Y to the functionality $\mathcal{F}_{\text{PSI}}$, and $\mathcal{F}_{\text{PSI}}$ computes the intersection $\text{Res} := X \cap Y$ and sends Res to Rec. PSI can be solved using generic MPC techniques, like GMW protocol [37] and Yao's garbled circuit protocol [38], while there are also custom protocols for this problem that are more efficient.

4. SM Series Cryptography

4.1. SM3 Hash Function and Key Derivation Function. Let $\text{SM3}(\cdot)$ denote the SM3 hash function that can map an arbitrary length string to a ℓ-bit hash digest, i.e., $\text{SM3} : \{0,1\}^* \longrightarrow \{0,1\}^\ell$. Let $\text{KDF}(m, \text{length})$ denote the key derivation function that takes input as the string $m \in \{0,1\}^*$ and the key length $\text{length} \in \mathbb{N}$, and it outputs length-bit key string k. In this work, we implement KDF with SM3 hash function SM3, and the details are shown in Algorithm 1.

The security of KDF directly follows the security of SM3.

4.2. SM4 Block Cipher Algorithm. Let $F : \{0,1\}^\lambda \times \{0,1\}^\ell \longrightarrow \{0,1\}^\ell$ denote the block cipher that takes a λ-bit seed and ℓ-bit plaintext as input and output an ℓ-bit ciphertext. The SM4 block cipher algorithm has an electronic codebook (ECB) mode and a cipher block chaining (CBC) mode. In ECB mode, SM4 is instantiated by repeatedly invoking F, in other words, $\text{SM4}_k(m_1, \cdots, m_n) = (F_k(m_1), \cdots, F_k(m_n))$; in CBC mode, each block of ciphertext depends on the previous ciphertext block and an initialization vector iv is used, more specifically, $\text{SM4}_k(m_1, \cdots, m_n) = (c_1, \cdots, c_n)$, where $c_1 := F_k(m_1 \oplus \text{iv})$ and $c_i := F_k(m_i \oplus c_{i-1})$ for $i \neq 1$.

4.3. SM2 Key Agreement Protocol. The SM2 key agreement protocol is defined in the PKI setting, and it works on elliptic curve E defined over the field $\mathbb{F}_p$. The parties A and B first agree on the output key length and an elliptic curve system where the elliptic curve discrete logarithm problem (ECDLP) is hard. The system parameters include $p, a, b, g = (x_g, y_g), n, h$, and $w := \lceil (\lceil \log_2(n) \rceil / 2) \rceil - 1$ is used in the computation.

We assume each party knows the other party's distinguishing identifier ID and a PKI distributes the public key $\text{pk} = (x_{\text{pk}}, y_{\text{pk}})$ computed from $\text{pk} = [\text{sk}]g$. Therefore, the parties can compute the identifier hash value $Z := \text{SM3}(\text{ENTL} \| \text{ID} \| a \| b \| x_g \| y_g \| x_{\text{pk}} \| y_{\text{pk}})$, where ENTL is the length of ID and SM3 outputs a 256-bit string. Therefore, A has s $\text{k}_A, \text{pk}_A, \text{pk}_B, Z_A, Z_B$, and B has $\text{sk}_B, \text{pk}_A, \text{pk}_B, Z_A, Z_B$. The public key/secret key pairs are used to prevent the man-in-the-middle attack, and the identifier hash values are used to identify the parties executing the protocol. In a nutshell, the SM2 key agreement protocol consists of the following PPT algorithms which use elliptic curve system parameters implicitly:

1. Set ctr = 1;
2. For $i \in [\lceil \text{length}/\ell \rceil]$:
   (a) Compute $h_{\text{ctr}} \longleftarrow \text{SM3}(m \| ctr)$;
   (b) Set ctr = ctr + 1;
3. If $\lceil \text{length}/\ell \rceil \neq \text{length}/\ell$, set $\bar{h}$ as the top length $- (\ell * \lfloor \text{length}/\ell \rfloor)$ bits of $h_{\lceil \text{length}/\ell \rceil}$; otherwise, set $\bar{h} := h_{\lceil \text{length}/\ell \rceil}$;
4. Set $k := h_1 \| \cdots \| h_{\lceil \text{length}/\ell \rceil - 1} \| \bar{h}$;
5. Output k.

Algorithm 1

(i) $(m, t) \longleftarrow \text{MsgGen}()$ is the message generation algorithm that outputs a fresh random private message t and the corresponding public message m

(ii) $p \longleftarrow \text{PointGen}(t, m, m', \text{sk}, \text{pk}')$ is the point generation algorithm that takes input as the party's private message t, both parties' public messages m, m', the party's secret key sk, and the other party's public key pk', and it outputs a point that can be used to derive the shared key

(iii) $k \longleftarrow \text{KeyGen}(p, Z_A, Z_B)$ is the key generation algorithm that takes input as the point p, A and B's identification message Z_A, Z_B, and it outputs a shared key k

We slightly modify the order of message delivery in the original SM2 key agreement protocol to achieve the one-round property. In the original protocol, party B sends the messages m_B to A after it obtains the key k_B, which indicates B sends m_B only after m_A is received, while we notice that this step can be done right after m_B is generated, and it does not raise any security issues. In our modified protocol, A first invokes MsgGen to generate $(m_A, t_A) \longleftarrow \text{MsgGen}()$, and it sends m_A to B. After receiving m_B from B, A computes the point $p_A := \text{PointGen}(t_A, m_A, m_B, \text{sk}_A, \text{pk}_B)$ and derives $k_A := \text{KeyGen}(p_A, Z_A, Z_B)$. The execution of B is exactly symmetric. The process of the protocol is illustrated in Figure 6, and the details of the algorithms can be found in Figure 7.

When key confirmation is needed, an augmented SM2 key agreement protocol can be used which contains one more PPT algorithm Verify and several more steps.

$h \longleftarrow \text{Verify}(s, p, m_A, m_B, Z_A, Z_B)$ is the verification algorithm that takes input as a string s, a point p, both parties' public messages m_A, m_B, and both parties' identifier hash values Z_A, Z_B, and it outputs a hash value h.

As in Figure 6, after generating p_A and k_A, A invokes $h_A := \text{Verify}(\text{"}A\text{"}, p_A, m_A, m_B, Z_A, Z_B)$ as a proof that it obtains a correct point p_A and is able to derive a correct key, and it sends h_A to B. When it receives h_B from B, it checks if B obtains the correct point p_B. The process of B is likewise.

Claim 10. If the key derivation function KDF is modeled as a random oracle, and the ECDLP is hard in $E_p(a, b)$, then the SM2 key agreement protocol is a one-round KA protocol with perfect correctness, security, perfect Msg_1 uniformity, perfect Msg_2 uniformity, and robustness.

Proof. It has been shown in [39] that the SM2 key agreement protocol is secure in the well-known Bellare-Rogaway model [40, 41] when the key derivation function is modeled as a random oracle and the ECDLP is hard in $E_p(a, b)$. Security in the Bellare-Rogaway model means perfect correctness and security of the KA protocol. Moreover, as illustrated in Figure 7, $m_A = [t_A]g$, where t_A is a uniformly random element from $[n-1]$; since m_A is generated by a bijective function, it should be perfectly indistinguishable from a random element in $\mathbb{G}$, which indicates the perfect Msg_1 uniformity of the protocol, and the perfect Msg_2 uniformity can be obtained in the same way. At the end, the parties A and B already hold $p_A = p_B$ after PointGen is invoked, which can be used as a shared key, and the KeyGen function only converts the point to a bit string. As proved in [14], this gives the robustness of the SM2 key agreement protocol when KDF is modeled as a random oracle, since a random oracle outputs uniformly random strings. □

5. Construct Oblivious Transfer Using SM

In this section, we show how to construct an oblivious transfer protocol using the SM series cryptography and the batch N-POPF. Moreover, we illustrate how to extend the number of OT instances using OT extension protocols. Before presenting the constructions, we first provide the descriptions of the symbols used in Table 1.

5.1. Oblivious Transfer from SM2 Key Agreement. Our one-round 1-out-of-N endemic oblivious transfer protocol $\Pi^{1,N}_{E-\text{OT}}$ is constructed from the SM2 key agreement protocol and the batch N-POPF defined in Figure 3. As depicted in Figure 8, the sender Sen and the receiver Rec first run a setup phase to determine the protocol parameters and to exchange the public keys along with the distinguishable identifiers. This setup phase only needs to be run once between these two parties Sen and Rec, and the parameters can be used in multiple instances.

When Sen receives the instruction (Send, sid, ssid) from the environment $\mathcal{Z}$, it invokes MsgGen to generate m_A, t_A, and it sends m_A to Rec. Meanwhile, Rec receives the instruction (Receive, sid, ssid, c) from $\mathcal{Z}$, and it invokes $(m_B, t_B) \longleftarrow \text{MsgGen}()$. After that, Rec generates $\{r_i\}_{i \in [N]}$

$A : \mathsf{sk}_A$		$B : \mathsf{sk}_B$
$(m_A, t_A) \leftarrow \mathsf{MsgGen}()$		$(m_B, t_B) \leftarrow \mathsf{MsgGen}()$
	$\xrightarrow{m_A}$	
	$\xleftarrow{m_B}$	
$p_A := \mathsf{PointGen}(t_A, m_A, m_B, \mathsf{sk}_A, \mathsf{pk}_B)$		$p_B := \mathsf{PointGen}(t_B, m_B, m_A, \mathsf{sk}_B, \mathsf{pk}_A)$
$k_A := \mathsf{KeyGen}(p_A, Z_A, Z_B)$		$k_B := \mathsf{KeyGen}(p_B, Z_A, Z_B)$
	$*optional*$	
$h_A := \mathsf{Verify}(\text{“}A\text{”}, p_A, m_A, m_B, Z_A, Z_B)$		$h_B := \mathsf{Verify}(\text{“}B\text{”}, p_B, m_A, m_B, Z_A, Z_B)$
	$\xrightarrow{h_A}$	
	$\xleftarrow{h_B}$	
$h'_B := \mathsf{Verify}(\text{“}B\text{”}, p_A, m_A, m_B, Z_A, Z_B)$		$h'_A := \mathsf{Verify}(\text{“}A\text{”}, p_B, m_A, m_B, Z_A, Z_B)$
Assert $h_B = h'_B$		Assert $h_A = h'_A$

Figure 6: SM2 key agreement protocol with optional key confirmation. The parties A and B share the elliptic curve system parameters $p, a, b, g, n, h, w := \lceil (\lceil \log_2(n) \rceil / 2) \rceil - 1$; the public keys $\mathrm{pk}_A, \mathrm{pk}_B$; the identifier hash values Z_A, Z_B; and the output key length.

$\mathsf{MsgGen}() :$
- Pick random $t \leftarrow [n-1]$
- Set $m := [t]g$
- Output (m, t)

$\mathsf{KeyGen}(p, Z_A, Z_B) :$
- $p = (x^*, y^*)$
- Compute $k := \mathsf{KDF}(x^* || y^* || Z_A || Z_B, \mathsf{length})$
- Output k

$\mathsf{PointGen}(t, m, m', \mathsf{sk}, \mathsf{pk}') :$
- $m = (x, y)$ and $m' = (x', y')$
- Set $\bar{x} := 2^w + (x \& (2^w - 1))$
- Set $\bar{x}' := 2^w + (x' \& (2^w - 1))$
- Set $r := (\mathsf{sk} + \bar{x} \cdot t) \mod n$
- Set $p := [h \cdot r](\mathsf{pk}' + [\bar{x}']m')$
- Output p

$\mathsf{Verify}(s, p, m_A, m_B, Z_A, Z_B)$
- $p = (x^*, y^*)$
- $m_A = (x_A, y_A)$ and $m_B = (x_B, y_B)$
- Compute $r := \mathsf{SM3}(x^* || Z_A || Z_B || x_A || y_A || x_B || y_B)$
- Compute $h := \mathsf{SM3}(s || y^* || r)$
- Output h

Figure 7: Algorithms used in SM2 key agreement protocol.

Table 1: Symbols used in the OT protocol and the OT extension protocol.

Symbol	Description
sk_A, sk_B	The private key of the users
pk_A, pk_B	The public key of the users
Z_A, Z_B	The hash value of the user's ID, the system's parameter, and the user's public key
MsgGen()	The message generation algorithm in the SM2 key agreement protocol
PointGen()	The point generation algorithm in the SM2 key agreement protocol
KeyGen()	The key generation algorithm in the SM2 key agreement protocol
Program()	The program algorithm in the batch N-POPF
Eval()	The evaluation algorithm in the batch N-POPF

by $\{r_i\}_{i \in [N]} := \mathrm{Program}(c, m_B)$ and sends $\{r_i\}_{i \in [N]}$ to Sen. Upon receiving $\{r_i\}_{i \in [N]}$ from Rec, for $i \in [N]$, Sen sets $m_{B,i} := \mathrm{Eval}(\{r_j\}_{j \in [N]}, i)$, computes $p_i := \mathrm{PointGen}(t_A, m_A, m_{B,i}, \mathrm{sk}_A, \mathrm{pk}_B)$, and sets $k_i := \mathrm{KeyGen}(p_i, Z_A, Z_B)$. Sen then returns (Send, sid, ssid, $\{k_i\}_{i \in [N]}$) to $\mathcal{Z}$. Upon receiving m_A, Rec computes $p_c := \mathrm{PointGen}(t_B, m_B, m_A, \mathrm{sk}_B, \mathrm{pk}_A)$ and sets $k_c := \mathrm{KeyGen}(p_c, Z_A, Z_B)$. At the end, Rec returns (Receive, sid, ssid, k_c) to $\mathcal{Z}$.

The correctness of the protocol directly follows the correctness of the SM2 key agreement protocol and the batch N-POPF. For the security proof, intuitively, since the SM2 key agreement has perfect Msg_2 uniformity, the message m_B should be indistinguishable from a random element from $\mathbb{G}$; by the honest simulation property of the batch N-POPF, r_c should be indistinguishable from other $\{r_i\}_{i \neq c}$. Therefore, Rec's choice c remains private to Sen. Besides, because of the robustness of the SM2 key agreement

The One-Round 1-out-of-N Endemic Oblivious Transfer Protocol $\Pi^{1,N}_{\text{E-OT}}$

Setup: The sender Sen and the receiver Rec first agree on the SM2 key agreement protocol parameters with the assistance of the PKI. The parameters include the elliptic curve system parameters p, a, b, g, n, h, public keys $\mathsf{pk}_A, \mathsf{pk}_B$, and identifier hash values Z_A, Z_B, the key length is set as the OT message length.

(i) Upon receiving $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ from the environment $\mathcal{Z}$, Sen:
  (i) Compute $(m_A, t_A) \leftarrow \mathsf{MsgGen}()$;
  (ii) Send m_A to Rec;
(ii) Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, c)$ from the environment $\mathcal{Z}$, Rec:
  (i) Compute $(m_B, t_B) \leftarrow \mathsf{MsgGen}()$;
  (ii) Compute $\{r_i\}_{i\in[N]} := \mathsf{Program}(c, m_B)$;
  (iii) Send $\{r_i\}_{i\in[N]}$ to Sen;
(iii) Upon receiving $\{r_i\}_{i\in[N]}$ from Rec, Sen:
  (i) For $i \in [N]$:
    (i) Compute $m_{B,i} := \mathsf{Eval}(\{r_j\}_{j\in[N]}, i)$;
    (ii) Compute $p_i := \mathsf{PointGen}(t_A, m_A, m_{B,i}, \mathsf{sk}_A, \mathsf{pk}_B)$;
    (iii) Compute $k_i := \mathsf{KeyGen}(p_i, Z_A, Z_B)$;
  (ii) Return $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid}, \{k_i\}_{i\in[N]})$ to the environment $\mathcal{Z}$;
(iv) Upon receiving m_A from Sen, Rec:
  (i) Compute $p_c := \mathsf{PointGen}(t_B, m_B, m_A, \mathsf{sk}_B, \mathsf{pk}_A)$;
  (ii) Compute $k_c := \mathsf{KeyGen}(p_c, Z_A, Z_B)$;
  (iii) Return $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, k_c)$ to the environment $\mathcal{Z}$;

Figure 8: One-round 1-out-of-N endemic oblivious transfer protocol $\Pi^{1,N}_{E-\text{OT}}$.

protocol and the uncontrollable output property of the batch N-POPF, the output messages $\{k_i\}_{i\neq c}$ should be unpredictable to Rec.

More formally, we use a theorem from [15]:

Theorem 11 (See [15]). *If the KA protocol has security, Msg_2 uniformity and robustness and the batch N-POPF are secure, and then, the protocol $\Pi^{1,N}_{E-OT}$ described in Figure 8 securely realizes the endemic 1-OPRF functionality in the random oracle model.*

Specifically, 1-OPRF is essentially 1-out-of-N OT. Therefore, we have the following result:

Theorem 12. *If ECDLP is hard in $\mathbb{G}$, the hash functions $\{hash_i^{\mathbb{G}}\}_{i\in[N]}$ and the key derivation function KDF are modeled as random oracles, and then, the protocol $\Pi^{1,N}_{E-OT}$ described in Figure 8 securely realizes $\mathcal{F}^{1,N}_{E-OT}$ described in Figure 4 against any PPT malicious adversary corrupting Sen or/and Rec.*

The proof is automatically done given Theorems 9 and 11.

The main focus of [12, 14, 15] is the malicious setting. When it comes to the semihonest setting, we notice that $\Pi^{1,N}_{E-\text{OT}}$ securely realizes the standard random OT functionality $\mathcal{F}^{1,N}_{U-\text{OT}}$ as well as the endemic OT functionality $\mathcal{F}^{1,N}_{E-\text{OT}}$. This is because the power of the adversary is limited to observing the protocol messages. We provide the results below.

Theorem 13. *If ECDLP is hard in $\mathbb{G}$, the hash functions $\{hash_i^{\mathbb{G}}\}_{i\in[N]}$ and the key derivation function KDF are modeled as random oracles, and then, the protocol $\Pi^{1,N}_{E-OT}$ described in Figure 8 securely realizes $\mathcal{F}^{1,N}_{U-OT}$ described in Figure 9 against any PPT semihonest adversary corrupting Sen or/and Rec.*

The proof can be found in Appendix B.1.

Corollary 14. *If ECDLP is hard in $\mathbb{G}$, the hash functions $\{hash_i^{\mathbb{G}}\}_{i\in[N]}$ and the key derivation function KDF are modeled as random oracles, and then, the protocol $\Pi^{1,N}_{E-OT}$ described in Figure 8 securely realizes $\mathcal{F}^{1,N}_{E-OT}$ described in Figure 4 against any PPT semihonest adversary corrupting Sen or/and Rec.*

The proof can be found in Appendix B.2.

5.2. Oblivious Transfer Extension. Although the endemic OT protocol $\Pi^{1,N}_{E-\text{OT}}$ is quite efficient, the exponentiation operations in $\Pi^{1,N}_{E-\text{OT}}$ can still be too expensive when we need millions of OTs in applications. In such cases, a technique called oblivious transfer extension can be adopted to generate OTs much faster. An OT extension protocol takes a bunch of "base" OTs to initiate the protocol, and then, it extends them to polynomially many OTs using only symmetric primitives, instead of asymmetric primitives. The Beaver [42] first introduced the idea of OT extension, and the following works [43, 44] proposed several highly efficient OT extension protocols. In this work, we use the well-optimized 1-out-of-2 OT extension protocol from [45] in the semihonest setting; in the malicious setting, we consider the protocol from [46]

1-out-of-N Random Oblivious Transfer Functionality $\mathcal{F}_{\text{U-OT}}^{1,N}$

It interacts with players $\mathcal{P} := \{\text{Sen}, \text{Rec}\}$ and the adversary $\mathcal{S}$. It is parameterized by the length of the messages length. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Transfer:

(i) Upon receiving $(\textsc{Send}, \text{sid}, \text{ssid})$ from Sen:
  (i) Send a notification $(\textsc{SendNotify}, \text{sid}, \text{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Send}, \text{sid}, \text{ssid})$;
  (iii) Ignore future $(\textsc{Send}, \text{sid}, \text{ssid})$ messages with the same sid, ssid;
(ii) Upon receiving $(\textsc{Receive}, \text{sid}, \text{ssid}, c)$ from Rec, where c in $[N]$:
  (i) Send a notification $(\textsc{ReceiveNotify}, \text{sid}, \text{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Receive}, \text{sid}, \text{ssid}, c)$;
  (iii) Ignore future $(\textsc{Receive}, \text{sid}, \text{ssid}, \ldots)$ messages with the same sid, ssid;
(iii) If both $(\textsc{Send}, \text{sid}, \text{ssid})$ and $(\textsc{Receive}, \text{sid}, \text{ssid}, c)$ are stored:
  (i) For $i \in [N]$, pick random $m_i \leftarrow \{0,1\}^{\text{length}}$;
  (ii) Send $(\textsc{Send}, \text{sid}, \text{ssid}, \{m_i\}_{i\in[N]})$ to Sen and $(\textsc{Receive}, \text{sid}, \text{ssid}, m_c)$ to Rec;

Corruption handling:

(i) Upon receiving $(\textsc{Corrupt}, \text{sid}, \text{ssid}, P)$ from the adversary $\mathcal{S}$, if $P \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P\}$;
  (ii) Send $(\textsc{Input}, \text{sid}, \text{ssid}, P, x)$ to $\mathcal{S}$ if party P's input x is already defined;

Figure 9: 1-out-of-N random oblivious transfer functionality $\mathcal{F}_{U\text{–OT}}^{1,N}$.

Multi-instance Endemic Oblivious Transfer Functionality $\mathcal{F}_{\text{E-OT}}^{\text{num}}$

It interacts with players $\mathcal{P} := \{\text{Sen}, \text{Rec}\}$ and the adversary $\mathcal{S}$. It is parameterized by the length of the messages length. num is the number of OT's to be generated. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Transfer:

(i) Upon receiving $(\textsc{Send}, \text{sid}, \text{ssid})$ from Sen:
  (i) Send a notification $(\textsc{SendNotify}, \text{sid}, \text{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Send}, \text{sid}, \text{ssid})$;
  (iii) Ignore future $(\textsc{Send}, \text{sid}, \text{ssid})$ messages with the same sid, ssid;
(ii) Upon receiving $(\textsc{Receive}, \text{sid}, \text{ssid}, \{c_i\}_{i\in[\text{num}]})$ from Rec:
  (i) Send a notification $(\textsc{ReceiveNotify}, \text{sid}, \text{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Receive}, \text{sid}, \text{ssid}, \{c_i\}_{i\in[\text{num}]})$;
  (iii) Ignore future $(\textsc{Receive}, \text{sid}, \text{ssid}, \ldots)$ messages with the same sid, ssid;
(iii) If both $(\textsc{Send}, \text{sid}, \text{ssid})$ and $(\textsc{Receive}, \text{sid}, \text{ssid}, \{c_i\}_{i\in[\text{num}]})$ are stored:
  (i) For $i \in [\text{num}], j \in \{0,1\}$, pick random $m_i^j \leftarrow \{0,1\}^{\text{length}}$;
  (ii) If Sen $\in \tilde{\mathcal{P}}$, wait for $(\textsc{FixMessage}, \text{sid}, \text{ssid}, \{\tilde{m}_i^j\}_{i\in[\text{num}], j\in\{0,1\}})$ from $\mathcal{S}$, set $m_i^j := \tilde{m}_i^j$, for $i \in [\text{num}], j \in \{0,1\}$;
  (iii) If Rec $\in \tilde{\mathcal{P}}$, wait for $(\textsc{FixMessage}, \text{sid}, \text{ssid}, \{\tilde{m}_i^{c_i}\}_{i\in[\text{num}]})$ from $\mathcal{S}$, set $m_i^{c_i} := \tilde{m}_i^{c_i}$, for $i \in [\text{num}]$;
  (iv) Send $(\textsc{Send}, \text{sid}, \text{ssid}, \{m_i^j\}_{i\in[\text{num}], j\in\{0,1\}})$ to Sen and $(\textsc{Receive}, \text{sid}, \text{ssid}, \{m_i^{c_i}\}_{i\in[\text{num}]})$ to Rec;

Corruption handling:

(i) Upon receiving $(\textsc{Corrupt}, \text{sid}, \text{ssid}, P)$ from the adversary $\mathcal{S}$, if $P \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P\}$;
  (ii) Send $(\textsc{Input}, \text{sid}, \text{ssid}, P, x)$ to $\mathcal{S}$ if party P's input x is already defined;

Figure 10: Multi-instance endemic oblivious transfer functionality $\mathcal{F}_{E\text{–OT}}^{\text{num}}$.

with endemic OT as base OT and apply the result of [47] to reduce the communication round. When it comes to the 1-out-of-N OT extension, results of [48, 49] can be adopted. The OT extension protocols securely realize the multi-instance version of the OT functionalities, and we provide the multi-instance 1-out-of-2 endemic OT functionality in Figure 10, which is similar to $\mathcal{F}_{E\text{–OT}}$. The multi-instance 1-out-of-2 uniform OT functionality can be found in Appendix A.1.

The Multi-instance Endemic Oblivious Transfer Protocol $\Pi^{\text{num}}_{\text{E-OT}}$

Setup: The sender Sen and the receiver Rec first agree on the SM2 key agreement protocol parameters with the assistance of the PKI. The parameters include the elliptic curve system parameters p, a, b, g, n, h, public keys $\mathsf{pk}_A, \mathsf{pk}_B$, and identifier hash values Z_A, Z_B, the key length is set as the OT message length.

(i) Upon receiving $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ from the environment $\mathcal{Z}$, Sen:
  (i) Compute $(m_A, t_A) \leftarrow \mathsf{MsgGen}()$;
  (ii) Send m_A to Rec;
(ii) Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{c_i\}_{i\in[\mathsf{num}]})$ from the environment $\mathcal{Z}$, Rec:
  (i) For $i \in [\mathsf{num}]$:
    (i) Compute $(m_{B,i}, t_{B,i}) \leftarrow \mathsf{MsgGen}()$;
    (ii) Compute $\{r_i^j\}_{j\in[2]} := \mathsf{Program}(c, m_{B,i})$;
  (ii) Send $\{r_i^j\}_{i\in[\mathsf{num}],j\in[2]}$ to Sen;
(iii) Upon receiving $\{r_i^j\}_{i\in[\mathsf{num}],j\in[2]}$ from Rec, Sen:
  (i) For $i \in [\mathsf{num}], j \in [2]$:
    (i) Compute $m_{B,i}^j := \mathsf{Eval}(\{r_i^k\}_{k\in[2]}, j)$;
    (ii) Compute $p_i^j := \mathsf{PointGen}(t_A, m_A, m_{B,i}^j, \mathsf{sk}_A, \mathsf{pk}_B)$;
    (iii) Compute $k_i^j := \mathsf{KeyGen}(p_i^j, Z_A, Z_B)$;
  (ii) Return $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid}, \{k_i^j\}_{i\in[\mathsf{num}],j\in\{0,1\}})$ to the environment $\mathcal{Z}$;
(iv) Upon receiving m_A from Sen, Rec:
  (i) For $i \in [\mathsf{num}]$:
    (i) Compute $p_i^{c_i} := \mathsf{PointGen}(t_{B,i}, m_{B,i}, m_A, \mathsf{sk}_B, \mathsf{pk}_A)$;
    (ii) Compute $k_i^{c_i} := \mathsf{KeyGen}(p_i^{c_i}, Z_A, Z_B)$;
  (ii) Return $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{k_i^{c_i}\}_{i\in[\mathsf{num}]})$ to the environment $\mathcal{Z}$;

Figure 11: Multi-instance endemic oblivious transfer protocol $\Pi^{\text{num}}_{E-\text{OT}}$.

5.2.1. Batching Base OT. The base OTs can be obtained by repeatedly invoking the single-instance functionality $\mathcal{F}^{1,2}_{E-\text{OT}}$ or $\mathcal{F}^{1,2}_{U-\text{OT}}$; however, the more efficient way is to design a protocol that directly realizes the multi-instance functionality. Our construction $\Pi^{\text{num}}_{E-\text{OT}}$ is essentially the same as $\Pi^{1,N}_{E-\text{OT}}$, where $N = 2$, except that Rec generates multiple message pairs $(m_{B,i}, t_{B,i})$ at once, and each message pair is used to generate one OT instance. The details can be found in Figure 11. This batching method saves Sen from repeatedly generating message pairs $m_{A,i}, t_{A,i}$, thus reducing the computation and communication costs. In [15], McQuoid et al. showed that this batching preserves the security of the original protocol when a tag is used in the generation of the KA protocol output to produce different OT results for each OT instance. In our protocol, we preserve the structure of the SM2 key agreement protocol, and we add the tag in the key derivation function KDF: we set the tag as $i\|1$ for the i-th KDF invocation. Therefore, the protocol $\Pi^{\text{num}}_{E-\text{OT}}$ in Figure 11 is secure, and we have the following theorem:

Theorem 15. *If ECDLP is hard in $\mathbb{G}$, the hash functions $\{hash_i^{\mathbb{G}}\}_{i\in[N]}$ and the key derivation function KDF are modeled as random oracles, and then, the protocol Π^{num}_{E-OT} described in Figure 11 securely realizes $\mathcal{F}^{num}_{E-OT}$ described in Figure 10 against any PPT malicious adversary corrupting Sen or/and Rec.*

5.2.2. OT Extension in Semi-Honest Setting. In the semihonest setting, the protocol $\Pi^{1,N}_{E-\text{OT}}$ securely realizes the random OT functionality $\mathcal{F}^{1,2}_{U-\text{OT}}$ when $N = 2$, so the protocol $\Pi^{\text{num}}_{E-\text{OT}}$ securely realizes the multi-instance random OT functionality $\mathcal{F}^{\text{num}}_{U-\text{OT}}$. Therefore, we take 1-out-of-2 random OT as the base OT of the OT extension protocol. As depicted in Figure 12, the OT extension protocol $\Pi^{\text{semi}}_{\text{OTE}}$ needs λ base OTs to start the extension; typically, $\lambda = 128$ is used considering both security and performance. To generate the base OTs, the sender Sen of the outer protocol acts as the receiver, and it picks random select bits $\{r_i\}_{i\in[\lambda]}$ and sends $\{r_i\}_{i\in[\lambda]}$ to $\mathcal{F}^{\text{num}}_{U-\text{OT}}$; the receiver Rec of the outer protocol acts as the sender and sends (Send, sid, ssid) to $\mathcal{F}^{\text{num}}_{U-\text{OT}}$. The random OT functionality picks random $\{k_i^0, k_i^1\}_{i\in[\lambda]}$, and it sends $\{k_i^0, k_i^1\}_{i\in[\lambda]}$ to Rec and sends $\{k_i^{r_i}\}_{i\in[\lambda]}$ to Sen. After obtaining the base OTs, Rec forms the choice bits $\{c_i\}_{i\in[m]}$ as a column vector. For $i \in [\lambda]$, Rec computes the PRG to generate $t^i \longleftarrow \text{PRG}(k_i^0)$ and parses t^i as a column vector, and it sets $u^i := t^i \oplus \text{PRG}(k_i^1) \oplus C$. After that, Rec forms a $m \times \lambda$ matrix $T := (t^1, \cdots, t^\lambda)$. For $i \in [m]$, Rec computes $h_i^{c_i} \longleftarrow$ has $\text{h}_{\text{length}}(i, T_i)$ as its random OT message where T_i is the i-th row of the matrix T. Subsequently, Rec sends $\{u^i\}_{i\in[\lambda]}$ and outputs. After obtaining the base OT results and $\{u^i\}_{i\in[\lambda]}$, Sen sets $q^i := (r_i \cdot u^i) \oplus \text{PRG}(k_{r_i}^i)$ for $i \in [\lambda]$. It then forms a $m \times \lambda$ matrix $Q := (q^1, \cdots, q^\lambda)$ and a row vector $R := (r_1, \cdots, r_\lambda)$. For $i \in [m]$, Sen computes $h_i^0 \longleftarrow \text{hash}_{\text{length}}(i, Q_i)$ and $h_i^1 \longleftarrow$ has $\text{h}_{\text{length}}(i, Q_i \oplus R)$ as its OT output.

Now we examine the correctness of $\Pi^{\text{semi}}_{\text{OTE}}$. For each column of the matrix Q, $Q^i = (r_i \cdot u^i) \oplus \text{PRG}(k_i^{r_i}) = (r_i \cdot (\text{PRG}(k_i^0) \oplus \text{PRG}(k_i^1) \oplus C)) \oplus \text{PRG}(k_i^{r_i})$, we can write

The Semi-honest Setting 1-out-of-2 Oblivious Transfer Extension Protocol $\Pi_{\text{OTE}}^{\text{semi}}$

We use $\mathsf{num} = \lambda$ base OT's. The base OT's length is λ and the extended OT's length is length. $\mathsf{PRG} : \{0,1\}^{\lambda} \to \{0,1\}^{m}$ is a pseudorandom number generator function and $\mathsf{hash}_{\mathsf{length}} : \{0,1\}^{*} \to \{0,1\}^{\mathsf{length}}$ is a correlation robust hash function.

(i) Upon receiving $(\textsc{Extend}, \mathsf{sid}, \{c_i\}_{i\in[m]})$ from the environment $\mathcal{Z}$, Rec:
  (i) Send $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\text{U-OT}}^{\mathsf{num}}$;
(ii) Upon receiving $(\textsc{Extend}, \mathsf{sid})$ from the environment $\mathcal{Z}$, Sen:
  (i) For $i \in [\lambda]$, pick random $r_i \leftarrow \{0,1\}$;
  (ii) Send $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{r_i\}_{i\in[\lambda]})$ to $\mathcal{F}_{\text{U-OT}}^{\mathsf{num}}$;
(iii) Upon receiving $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid}, \{k_i^0, k_i^0\}_{i\in[\lambda]})$ from $\mathcal{F}_{\text{U-OT}}^{\mathsf{num}}$, Rec:
  (i) Set $C := (c_1, \ldots, c_m)^{\mathsf{T}}$ as a column vector;
  (ii) For $i \in [\lambda]$:
    (i) Set $t^i \leftarrow \mathsf{PRG}(k_i^0)$ as a column vector;
    (ii) Set $u^i := t^i \oplus \mathsf{PRG}(k_i^1) \oplus C$ as a column vector;
  (iii) Set $T := (t^1, \ldots, t^{\lambda})$ as a $m \times \lambda$ matrix;
  (iv) For $i \in [m]$, compute $h_i^{c_i} := \mathsf{hash}_{\mathsf{length}}(i, T_i)$;
  (v) Send $\{u^i\}_{i\in[\lambda]}$ to Sen;
  (vi) Return $(\textsc{Extend}, \mathsf{sid}, \{h_i^{c_i}\}_{i\in[m]})$ to the environment $\mathcal{Z}$;
(iv) Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{k_i^{r_i}\}_{i\in[\lambda]})$ from $\mathcal{F}_{\text{U-OT}}^{\mathsf{num}}$ and receiving $\{u^i\}_{i\in[\lambda]}$ from Rec, Sen:
  (i) For $i \in [\lambda]$, set $q^i := (r_i \cdot u^i) \oplus \mathsf{PRG}(k_i^{r_i})$ as a column vector;
  (ii) Set $Q := (q^1, \ldots, q^{\lambda})$ as a $m \times \lambda$ matrix;
  (iii) Set $R := (r_1, \ldots, r_{\lambda})$ as a row vector;
  (iv) For $i \in [m]$, compute $h_i^0 := \mathsf{hash}_{\mathsf{length}}(i, Q_i)$ and $h_i^1 := \mathsf{hash}_{\mathsf{length}}(i, Q_i \oplus R)$;
  (v) Return $(\textsc{Extend}, \mathsf{sid}, \{h_i^0, h_i^1\}_{i\in[m]})$ to the environment $\mathcal{Z}$;

Figure 12: Semihonest setting 1-out-of-2 oblivious transfer extension protocol $\Pi_{\text{OTE}}^{\text{semi}}$.

Multi-instance Random Oblivious Transfer Functionality $\mathcal{F}_{\text{U-OT}}^{\mathsf{num}}$

It interacts with players $\mathcal{P} := \{\mathsf{Sen}, \mathsf{Rec}\}$ and the adversary $\mathcal{S}$. It is parameterized by the length of the messages length. num is the number of OT's to be generated. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Transfer:

(i) Upon receiving $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ from Sen:
  (i) Send a notification $(\textsc{SendNotify}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$;
  (iii) Ignore future $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ messages with the same sid, ssid;
(ii) Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{c_i\}_{i\in[\mathsf{num}]})$ from Rec:
  (i) Send a notification $(\textsc{ReceiveNotify}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$;
  (ii) Store $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{c_i\}_{i\in[\mathsf{num}]})$;
  (iii) Ignore future $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \ldots)$ messages with the same sid, ssid;
(iii) If both $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid})$ and $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{c_i\}_{i\in[\mathsf{num}]})$ are stored:
  (i) For $i \in [\mathsf{num}], j \in \{0,1\}$, pick random $m_i^j \leftarrow \{0,1\}^{\mathsf{length}}$;
  (ii) Send $(\textsc{Send}, \mathsf{sid}, \mathsf{ssid}, \{m_i^j\}_{i\in[\mathsf{num}], j\in[2]})$ to Sen and $(\textsc{Receive}, \mathsf{sid}, \mathsf{ssid}, \{m_i^{c_i}\}_{i\in[\mathsf{num}]})$ to Rec;

Corruption handling:

(i) Upon receiving $(\textsc{Corrupt}, \mathsf{sid}, \mathsf{ssid}, P)$ from the adversary $\mathcal{S}$, if $P \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P\}$;
  (ii) Send $(\textsc{Input}, \mathsf{sid}, \mathsf{ssid}, P, x)$ to $\mathcal{S}$ if party P's input x is already defined;

Figure 13: Multi-instance random oblivious transfer functionality $\mathcal{F}_{U-\text{OT}}^{\text{num}}$.

$\mathrm{PRG}(k_i^{r_i}) = (1 \oplus r_i) \cdot \mathrm{PRG}(k_i^0) \oplus r_i \cdot \mathrm{PRG}(k_i^1)$ as before, and this gives us $Q^i = \mathrm{PRG}(k_i^0) \oplus r_i \cdot C = T^i \oplus r_i \cdot C$. Therefore, for each row of the matrix, we have $Q_i = T_i \oplus c_i \cdot R$, and thus, the protocol is correct. As for the security of the protocol, in [45], Asharov et al. prove that $\Pi_{\text{OTE}}^{\text{semi}}$ securely realizes $\mathcal{F}_{U-\text{OT}}^{\text{num}}$. Moreover, We can obtain a result similar to Corollary 14 that $\Pi_{\text{OTE}}^{\text{semi}}$ securely realizes the multi-instance endemic OT functionality $\mathcal{F}_{E-\text{OT}}^{\text{num}}$.

Theorem 16. *The protocol Π_{OTE}^{semi} described in Figure 12 securely realizes $\mathcal{F}_{U-OT}^{num}$ described in Figure 13 against any PPT semihonest adversary corrupting Sen or/and Rec.*

The proof is done by Asharov et al. [45].

Corollary 17. *The protocol Π_{OTE}^{semi} described in Figure 12 securely realizes $\mathcal{F}_{E-OT}^{num}$ described in Figure 10 against any PPT semihonest adversary corrupting Sen or/and Rec.*

The Malicious Setting 1-out-of-2 Oblivious Transfer Extension Protocol $\Pi_{\text{OTE}}^{\text{mal}}$

We use $\text{num} = \lambda$ base OT's. The base OT's length is λ and the extended OT's length is length. We set $m' = m + \mu$ and $\mathbb{F} = \{0,1\}^{\lambda}$. $\text{PRG} : \{0,1\}^{\lambda} \to \{0,1\}^{m'}$ is a pseudorandom function, $\text{hash}_{\mathbb{F}}^{m'} : \{0,1\}^{m' \times \lambda} \to \mathbb{F}^{m'}$ is a hash function and $\text{hash}_{\text{length}} : \{0,1\}^{*} \to \{0,1\}^{\text{length}}$ is a correlation robust hash function.

(i) Upon receiving $(\textsc{Extend}, \text{sid}, \{c_i\}_{i\in[m]})$ from the environment $\mathcal{Z}$, Rec:
  (i) Send $(\textsc{Send}, \text{sid}, \text{ssid})$ to $\mathcal{F}_{\text{E-OT}}^{\text{num}}$;
(ii) Upon receiving $(\textsc{Extend}, \text{sid})$ from the environment $\mathcal{Z}$, Sen:
  (i) For $i \in [\lambda]$, pick random $r_i \leftarrow \{0,1\}$;
  (ii) Send $(\textsc{Receive}, \text{sid}, \text{ssid}, \{r_i\}_{i\in[\lambda]})$ to $\mathcal{F}_{\text{E-OT}}^{\text{num}}$;
(iii) Upon receiving $(\textsc{Send}, \text{sid}, \text{ssid}, \{k_i^0, k_i^0\}_{i\in[\lambda]})$ from $\mathcal{F}_{\text{E-OT}}^{\text{num}}$, Rec:
  (i) For $i \in [\lambda]$, pick random $c_i' \leftarrow \{0,1\}$;
  (ii) Set $C := (c_1, \ldots, c_m, c_1', \ldots, c_\lambda')^{\mathsf{T}}$ as a column vector and $C' := (C, \ldots, C)$ as a $m' \times \lambda$ matrix;
  (iii) For $i \in [\lambda]$:
    (i) Set $t^i \leftarrow \text{PRG}(k_i^0)$ as a column vector;
    (ii) Set $u^i := t^i \oplus \text{PRG}(k_i^1) \oplus C$ as a column vector;
  (iv) Set $T := (t^1, \ldots, t^\lambda)$ and $U := (u^1, \ldots, u^\lambda)$ as $m' \times \lambda$ matrices;
  (v) Compute $\{\chi_1, \ldots, \chi_{m'}\} := \text{hash}_{\mathbb{F}}^{m'}(U)$;
  (vi) Set $v := \bigoplus_{i\in[m']}(\chi_i \cdot T_i)$ and $w := \bigoplus_{i\in[m']}(\chi_i \cdot C_i')$;
  (vii) For $i \in [m]$, compute $h_i^{c_i} := \text{hash}_{\text{length}}(i, T_i)$;
  (viii) Send U, v, w to Sen;
  (ix) Return $(\textsc{Extend}, \text{sid}, \{h_i^{c_i}\}_{i\in[m]})$ to the environment $\mathcal{Z}$;
(iv) Upon receiving $(\textsc{Receive}, \text{sid}, \text{ssid}, \{k_i^{r_i}\}_{i\in[\lambda]})$ from $\mathcal{F}_{\text{U-OT}}^{\text{num}}$ and receiving U, v, w from Rec, Sen:
  (i) Compute $\{\chi_1, \ldots, \chi_{m'}\} := \text{hash}_{\mathbb{F}}^{m'}(U)$;
  (ii) For $i \in [\lambda]$, set $q^i := (r_i \cdot U^i) \oplus \text{PRG}(k_i^{r_i})$ as a column vector;
  (iii) Set $Q := (q^1, \ldots, q^\lambda)$ as a $m' \times \lambda$ matrix;
  (iv) Set $R := (r_1, \ldots, r_\lambda)$ as a row vector;
  (v) Assert $\bigoplus_{i\in[m']}(\chi_i \cdot Q_i) = v \oplus R \cdot w$;
  (vi) For $i \in [m]$, compute $h_i^0 := \text{hash}_{\text{length}}(i, Q_i)$ and $h_i^1 := \text{hash}_{\text{length}}(i, Q_i \oplus R)$;
  (vii) Return $(\textsc{Extend}, \text{sid}, \{h_i^0, h_i^1\}_{i\in[m]})$ to the environment $\mathcal{Z}$;

Figure 14: Malicious setting 1-out-of-2 oblivious transfer extension protocol $\Pi_{\text{OTE}}^{\text{mal}}$.

The proof is similar to the proof of Corollary 14.

5.2.3. OT Extension in Malicious Setting. The malicious setting is more difficult to handle. When the base OTs are endemic OTs, we can only extend them to more endemic OTs, and the sender Sen needs to check the consistency of the messages sent by the receiver Rec. We provide the malicious setting OT extension protocol $\Pi_{\text{OTE}}^{\text{mal}}$ in Figure 14. The main process of extending oblivious transfer remains the same as the semihonest setting protocol $\Pi_{\text{OTE}}^{\text{semi}}$, so is the correctness of the protocol. However, a malicious Rec can violate the requirement that the same choice bits should be used when computing the vectors $\{u^i\}_{i\in[\lambda]}$. In [46], Rec needs to prove its honesty in zero knowledge, and generally speaking, the consistency check uses a random linear combination of the row vectors of the matrices, and the coefficients of the linear combination should be unpredictable to Rec, e.g., they are randomly picked by Sen. In [47], Doerner et al. use Fiat-Shamir heuristic [50] to make the zero-knowledge proof process noninteractive: the coefficients are generated by a hash function taking the matrix U as input. The formal security proof of the protocol $\Pi_{\text{OTE}}^{\text{mal}}$ can be found in [12].

Theorem 18. *The protocol Π_{OTE}^{mal} described in Figure 14 securely realizes $\mathcal{F}_{E-OT}^{num}$ described in Figure 10 against any PPT malicious adversary corrupting Sen or/and Rec.*

The proof is done by Masny and Rindal [12].

6. Generate the Beaver Triple

A wide range of MPC protocols working on circuits requires heavy computation and huge communication to compute AND gates and multiplication gates. To speed up the MPC protocols, a research trend is to split the protocol into a preprocessing phase independent of parties' input and an online phase where the computation proceeds using actual input and data from the preprocessing phase. Therefore, parties can run the preprocessing phase whenever they are available and respond instantly when the computation needs to proceed.

As for secret-sharing-based MPC protocols, Beaver introduces a notion of the Beaver triple (or multiplication triple) in [51]. Basically, a Beaver triple consists of shared triples $\{a_i, b_i, c_i\}_{i\in[N]}$, where a_i, b_i, c_i are chosen randomly with $\oplus c_i = \oplus a_i \wedge \oplus b_i$ that always holds. The length of the Beaver triple can vary with applications, and in this work, we refer the Beaver triple with a length larger than 1 as a

The Beaver/Multiplication Triple Generation Functionality $\mathcal{F}_{\text{TRIPLE}}$

It interacts with players $\mathcal{P} := \{P_1, \ldots, P_N\}$ and the adversary $\mathcal{S}$. It is parameterized by the length of the triples length. Let $\tilde{\mathcal{P}}$ be the set of corrupted parties. Initially, set $\tilde{\mathcal{P}} = \emptyset$.

Generate:

(i) Upon receiving $(\text{Generate}, \text{sid}, \text{ssid}, P_i)$ from P_i:
  (i) Send a notification $(\text{GenerateNotify}, \text{sid}, \text{ssid}, P_i)$ to $\mathcal{S}$;
  (ii) Store $(\text{Generate}, \text{sid}, \text{ssid}, P_i)$;
  (iii) Ignore future $(\text{Generate}, \text{sid}, \text{ssid}, P_i)$ messages with the same $\text{sid}, \text{ssid}, P_i$;

(ii) If all of $(\text{Generate}, \text{sid}, \text{ssid}, P_i)$ are stored for $i \in [N]$:
  (i) Wait for $(\text{FixTriple}, \text{sid}, \text{ssid}, P_i, (a_i, b_i, c_i))$ from $\mathcal{S}$ for $P_i \in \tilde{\mathcal{P}}$:
  (ii) For $P_i \notin \tilde{\mathcal{P}}$, pick random $a_i, b_i, c_i \leftarrow \{0,1\}^{\text{length}}$;
  (iii) Reset $c_i := (\sum a_j) \cdot (\sum b_j) - (\sum_{j \neq i} c_j)$ for the smallest i such that $P_i \notin \tilde{\mathcal{P}}$;
  (iv) For $i \in [N]$, send $(\text{Generate}, \text{sid}, \text{ssid}, P_i, (a_i, b_i, c_i))$ to P_i;

Corruption handling:

(i) Upon receiving $(\text{Corrupt}, \text{sid}, \text{ssid}, P_i)$ from the adversary $\mathcal{S}$, if $P_i \in \mathcal{P}$:
  (i) Set $\tilde{\mathcal{P}} := \tilde{\mathcal{P}} \cup \{P_i\}$;

Figure 15: Beaver/multiplication triple generation functionality $\mathcal{F}_{\text{triple}}$.

multiplication triple, and a Beaver triple with a length of 1 is simply called the Beaver triple.

As depicted in Figure 15, the triple-generation functionality $\mathcal{F}_{\text{triple}}$ first waits for all parties to send a (Generate, sid , P_i) instruction. After that, it allows the adversary $\mathcal{S}$ to determine the content of the triple received by the corrupted P_i by sending (FixTriple, sid, P_i, (a_i, b_i, c_i)). Subsequently, as for the parties not corrupted, $\mathcal{F}_{\text{triple}}$ picks uniformly random a_i, b_i, c_i. However, to ensure that $\oplus c_i = \oplus a_i \wedge \oplus b_i$, $\mathcal{F}_{\text{triple}}$ chooses the party with the smallest index i among the uncorrupted parties and sets $c_i := (\sum a_j) \cdot (\sum b_j) - (\sum_{j \neq i} c_j)$. At the end, $\mathcal{F}_{\text{triple}}$ sends (Generate, sid, P_i, (a_i, b_i, c_i)) back to P_i.

6.1. Two-Party Beaver Triple Generation. We first introduce how to generate the Beaver triple among two parties using endemic OT. In an OT execution, the receiver sends b to obtain m_b, and the sender obtains two messages m_0, m_1. Notice that m_b can be represented as $b \wedge m_1 \oplus (1 \oplus b) \wedge m_0$, and if we set $a = m_0 \oplus m_1$, then $m_0 \oplus m_b = b \wedge m_0 \oplus b \wedge m_1 = a \wedge b$. Therefore, invoking the endemic OT twice with P_1 and P_2 playing the sender in turn is directly a Beaver triple-generation protocol Π_{triple}, which can be found in Figure 16. In Π_{triple}, party P_i first picks random b_i, and then, it invokes the endemic OT functionality $\mathcal{F}^{1,2}_{E\text{-OT}}$ as sender and receiver, respectively. When $\mathcal{F}^{1,2}_{E\text{-OT}}$ sends m^i_0, m^i_1, and $m^{3-i}_{b_i}$ back to P_i, P_i sets $a_i := m^i_0 \oplus m^i_1$ and $c_i := a_i \wedge b_i \oplus m^i_0 \oplus m^{3-i}_{b_i}$.

In the implementation, the main cost of the protocol Π_{triple} is to invoke $\mathcal{F}^{1,2}_{E\text{-OT}}$ twice. When $\mathcal{F}^{1,2}_{E\text{-OT}}$ is instantiated with $\Pi^{1,N}_{E\text{-OT}}$, which is a one-round protocol, the protocol Π_{triple} also has round complexity 1. Besides, $\Pi^{1,N}_{E\text{-OT}}$ only needs to transfer 3 group elements in total, which means Π_{triple} only needs 6. Moreover, we can use the multi-instance OT functionality to generate a bunch of OT instances in advance, which further reduces the computation and communication costs. Therefore, Π_{triple} can be extremely suitable for lightweight devices in extreme network environments with high delay and low bandwidth.

Although endemic OT is a weak version of general random OT, the protocol Π_{triple} is still secure in the malicious setting, and we provide the theorem together with the proof below.

Theorem 19. *The protocol Π_{triple} described in Figure 16 securely realizes $\mathcal{F}_{triple}$ described in Figure 15 with length = 1 in the $\mathcal{F}^{1,2}_{E\text{-OT}}$-hybrid model against any PPT malicious adversary corrupting P_1 or P_2.*

The proof can be found in Appendix B.3.

Given this two-party Beaver triple-generation protocol Π_{triple}, we can use it to generate sufficiently many Beaver triples in the preprocessing phase and carry out a GMW-style two-party computation protocol [37] in the online phase, where the XOR gates can be computed locally and the AND gates consume one Beaver triple each and need communication. The details of the protocol $\Pi_{2\text{PC}}$ can be found in Figure 17. This provides an efficient solution to a generic two-party computation over the Boolean circuits.

Theorem 20. *The protocol Π_{2PC} described in Figure 17 securely realizes $\mathcal{F}^f_{mpc}$ described in Figure 1 in the $\mathcal{F}_{triple}$-hybrid model against any PPT semihonest adversary corrupting P_1 or P_2.*

The proof is done by combining the result of [37, 51].

6.2. Multiparty Multiplication Triple Generation. We can extend the Beaver triple-generation protocol Π_{triple} to the multiparty setting and generate multiplication triple of length ≥ 1 with one more round communication. Therefore, we obtain the protocol $\Pi^{N,\text{length}}_{\text{triple}}$ described in Figure 18 which is secure in the semihonest setting, even when the endemic

Two Party Beaver Triple Generation Protocol Π_{TRIPLE} in the $\mathcal{F}^{1,2}_{\text{E-OT}}$-hybrid model

The OT message length is set to 1.

(i) For $i \in [2]$, upon receiving $(\text{GENERATE}, \text{sid}, \text{ssid}, P_i)$ from the environment $\mathcal{Z}$, the party P_i:
  (i) Pick random $b_i \leftarrow \{0, 1\}$;
  (ii) Send $(\text{SEND}, \text{sid}, \text{ssid}_i)$ and $(\text{RECEIVE}, \text{sid}, \text{ssid}_{3-i}, b_i)$ to $\mathcal{F}^{1,2}_{\text{E-OT}}$;
(ii) For $i \in [2]$, upon receiving $(\text{SEND}, \text{sid}, \text{ssid}_i, (m^i_0, m^i_1))$ and $(\text{RECEIVE}, \text{sid}, \text{ssid}_{3-i}, m^{3-i}_{b_i})$ from $\mathcal{F}^{1,2}_{\text{E-OT}}$, the party P_i:
  (i) Set $a_i := m^i_0 \oplus m^i_1$;
  (ii) Set $c_i := a_i \wedge b_i \oplus m^i_0 \oplus m^{3-i}_{b_i}$;
  (iii) Return $(\text{GENERATE}, \text{sid}, \text{ssid}, P_i, (a_i, b_i, c_i))$ to the environment $\mathcal{Z}$;

FIGURE 16: Two-party Beaver triple generation protocol Π_{triple} in the $\mathcal{F}^{1,2}_{\text{E-OT}}$-hybrid model.

GMW-style Two Party Computation Protocol Π_{2PC} in the $\mathcal{F}_{\text{TRIPLE}}$-hybrid model

Let f be the circuit to be computed, WLOG, we assume f only consists of AND gates and XOR gates. The triple length is set to 1.

Initialization Phase

(i) For $j \in [2]$, upon receiving $(\text{COMPUTE}, \text{sid}, x_j := (x_{j,1}, \ldots, x_{j,n_j}))$ from the environment $\mathcal{Z}$, the party P_j:
  (i) For $k \in [n_j]$, pick random $x^1_{j,k} \leftarrow \{0, 1\}$ and set $x^2_{j,k} := x_{j,k} \oplus x^1_{j,k}$;
  (ii) Send $\{x^{3-j}_{j,k}\}_{k \in [n_j]}$ to P_{3-j};

Evaluation Phase

After the initialization phase, P_1 and P_2 obtain their shares of the input wires. They then evaluate the circuit gate by gate.

(i) For an XOR gate $(\text{id}, \alpha, \beta, \gamma)$:
  (i) For $j \in [2]$, the party P_j computes $x^j_\gamma = x^j_\alpha \oplus x^j_\beta$;
(ii) For an AND gate $(\text{id}, \alpha, \beta, \gamma)$:
  (i) For $j \in [2]$, P_j send $(\text{GENERATE}, \text{sid}, \text{ssid}, P_j)$ to $\mathcal{F}_{\text{TRIPLE}}$;
  (ii) For $j \in [2]$, upon receiving $(\text{GENERATE}, \text{sid}, \text{ssid}, P_j, (a_j, b_j, c_j))$ from $\mathcal{F}_{\text{TRIPLE}}$, the party P_j
    (i) Set $d_j := x^j_\alpha \oplus a_j$ and $e_j := x^j_\beta \oplus b_j$;
    (ii) Send d_j, e_j to P_{3-j};
  (iii) For $j \in [2]$, upon receiving d_{3-j}, e_{3-j} from P_{3-j}, the party P_j:
    (i) Set $d := d_1 \oplus d_2$ and $e := e_1 \oplus e_2$;
    (ii) Set $x^j_\gamma := d \wedge e \oplus d \wedge b_j \oplus e \wedge a_j \oplus c_j$;

Output Phase

(i) For $j \in [2]$, the party P_j broadcast $\{x^j_i\}_{i \in \text{out}}$;
(ii) For $j \in [2]$, upon receiving $\{x^{3-j}_i\}_{i \in \text{out}}$ from P_{3-j}, the party P_j:
  (i) For $i \in \text{out}$, compute $y_i := x^1_i \oplus x^2_i$;
  (ii) Return $(\text{COMPUTE}, \text{sid}, P_j, y := \{y_i\}_{i \in \text{out}})$ to the environment $\mathcal{Z}$;

FIGURE 17: GMW-style two-party computation protocol Π_{2PC} in the $\mathcal{F}_{\text{triple}}$-hybrid model.

OT functionality is used which gives more power to the adversary $\mathcal{A}$.

The core idea is still using the OT functionality to generate correlated messages, and again, we can simply invoke the multi-instance OT functionality and use the OT extension technique to generate polynomially many OT instances in advance. In Π_{triple}, all computations are on the ring $\mathbb{Z}_2$, while now, we consider $\mathbb{Z}_{2_{\text{length}}}$. Assume an endemic OT outputs m_0, m_1 to sender and m_b to receiver for choice bit b, and it holds that $m_b = b \cdot m_1 + (1 - b) \cdot m_0 \bmod 2^{\text{length}}$.

Consider the simple case where length = 1, we have

$$\begin{aligned} s_{j,i} - m^0_{i,j} &\equiv m^{b_i}_{j,i} + b_i \cdot r_{j,i} - m^0_{i,j} \\ &\equiv b_i \cdot m^1_{j,i} + (1 - b_i) \cdot m^0_{j,i} + b_i \\ &\quad \cdot \left(a_j + m^0_{j,i} - m^1_{j,i}\right) - m^0_{i,j} \\ &\equiv m^0_{j,i} + b_i \cdot a_j - m^0_{i,j}, \end{aligned} \tag{9}$$

Multi-Party Triple Generation Protocol $\Pi_{\text{TRIPLE}}^{N,\text{length}}$ in the $\mathcal{F}_{\text{E-OT}}^{1,2}$-hybrid model

We works on the ring $\mathbb{Z}_{2^{\text{length}}}$, where length is the triple length. The OT message length is set to length.

(i) For $i \in [N]$, upon receiving $(\textsc{Generate}, \text{sid}, \text{ssid}, P_i)$ from the environment $\mathcal{Z}$, the party P_i:
  (i) Pick random $a_i, b_i \leftarrow \{0,1\}^{\text{length}}$;
  (ii) For $j \neq i, k \in [\text{length}]$, send $(\textsc{Send}, \text{sid}, \text{ssid}_{i,j,k})$ and $(\textsc{Receive}, \text{sid}, \text{ssid}_{j,i,k}, b_i[k])$ to $\mathcal{F}_{\text{E-OT}}^{1,2}$;
(ii) For $i \in [N]$, upon receiving $\{(\textsc{Send}, \text{sid}, \text{ssid}_{i,j,k}, m_{i,j,k}^0, m_{i,j,k}^1)\}_{j\neq i, k\in[\text{length}]}$ and $\{(\textsc{Receive}, \text{sid}, \text{ssid}_{j,i}, m^{b_i[k]})\}_{j\neq i, k\in[\text{length}]}$ from $\mathcal{F}_{\text{E-OT}}^{1,2}$, the party P_i:
  (i) For $j \neq i, k \in [\text{length}]$, set $r_{i,j,k} := a_i + m_{i,j,k}^0 - m_{i,j,k}^1 \mod 2^{\text{length}}$;
  (ii) For $j \neq i$, send $\{r_{i,j,k}\}_{k\in[\text{length}]}$ to P_j;
(iii) For $i \in [N]$, upon receiving $\{r_{j,i,k}\}_{k\in[\text{length}]}$ from $\{P_j\}_{j\neq i}$, the party P_i:
  (i) For $j \neq i, k \in [\text{length}]$, set $s_{j,i,k} := m_{j,i,k}^{b_i[k]} + b_i[k] \cdot r_{j,i,k} \mod 2^{\text{length}}$;
  (ii) Set $c_i := a_i \cdot b_i + (\sum_{j\neq i} \sum_k (s^{j,i,k} - m_0^{i,j,k}) \cdot 2^{k-1}) \mod 2^{\text{length}}$;
  (iii) Return $(\textsc{Generate}, \text{sid}, \text{ssid}, P_i, (a_i, b_i, c_i))$ to the environment $\mathcal{Z}$;

Figure 18: Multiparty multiplication triple generation protocol $\Pi_{\text{triple}}^{N,\text{length}}$ in the $\mathcal{F}_{E\text{–OT}}^{1,2}$-hybrid model.

SPDZ-style Multi-Party Computation Protocol Π^{MPC} in the $\mathcal{F}_{\text{TRIPLE}}$-hybrid model

Let f be the circuit to be computed, WLOG, we assume f only consists of addition gates and multiplication gates over $\mathbb{Z}_{2^{\text{length}}}$. The triple length is set to length and the computations are over $\mathbb{Z}_{2^{\text{length}}}$.

Initialization Phase

(i) For $j \in [N]$, upon receiving $(\textsc{Compute}, \text{sid}, x_j := (x_{j,1}, \ldots, x_{j,n_j}))$ from the environment $\mathcal{Z}$, the party P_j:
  (i) For $k \in [n_j]$, for $i \neq j$, pick random $x_{j,k}^i \leftarrow \{0,1\}^{\text{length}}$;
  (ii) For $k \in [n_j]$, set $x_{j,k}^j := x_{j,k} - \sum_{i\neq j} x_{j,k}^i$;
  (iii) For $i \neq j$, send $\{x_{j,k}^i\}_{k\in[n_j]}$ to P_i;

Evaluation Phase

After the initialization phase, the parties $P_1, \ldots, P_N$ obtain their shares of the input wires. They then evaluate the circuit gate by gate.

(i) For an addition gate $(\text{id}, \alpha, \beta, \gamma)$:
  (i) For $j \in [N]$, the party P_j computes $x_\gamma^j = x_\alpha^j \oplus x_\beta^j$;
(ii) For a multiplication gate $(\text{id}, \alpha, \beta, \gamma)$:
  (i) For $j \in [N]$, P_j send $(\textsc{Generate}, \text{sid}, \text{ssid}, P_j)$ to $\mathcal{F}_{\text{TRIPLE}}$;
  (ii) For $j \in [N]$, upon receiving $(\textsc{Generate}, \text{sid}, \text{ssid}, P_j, (a_j, b_j, c_j))$ from $\mathcal{F}_{\text{TRIPLE}}$, the party P_j:
    (i) Set $d_j := x_\alpha^j - a_j$ and $e_j := x_\beta^j - b_j$;
    (ii) Broadcast d_j, e_j;
  (iii) For $j \in [N]$, upon receiving $\{d_i, e_i\}_{i\neq j}$ from $\{P_i\}_{i\neq j}$, the party P_j:
    (i) Set $d := \sum d_i$ and $e := \sum e_i$;
    (ii) Set $x_\gamma^j := d \cdot e + d \cdot b_j + e \cdot a_j + c_j$;

Output Phase

(i) For $j \in [N]$, the party P_j sends $\{x_k^j\}_{k\in\text{out}}$ to $\{P_i\}_{i\neq j}$;
(ii) For $j \in [N]$, upon receiving $\{x_k^i\}_{k\in\text{out}}$ from $\{P_i\}_{i\neq j}$, the party P_j:
  (i) For $k \in \text{out}$, compute $y_i := \sum x_k^i$;
  (ii) Return $(\textsc{Compute}, \text{sid}, P_j, y := \{y_k\}_{k\in\text{out}})$ to the environment $\mathcal{Z}$;

Figure 19: SPDZ-style multiparty computation protocol Π^{MPC} in the $\mathcal{F}_{\text{triple}}$-hybrid model.

The Private Set Intersection Protocol Π_{PSI} in the $\mathcal{F}^{1,2}_{\text{U-OT}}$-hybrid model

Sen and Rec first agree on the protocol parameters m, w, ℓ_1, ℓ_2, hash functions $\text{hash}_{\ell_1} : \{0,1\}^* \to \{0,1\}^{\ell_1}$ and $\text{hash}_{\ell_2} : \{0,1\}^w \to \{0,1\}^{\ell_2}$, and the pseudorandom function $\text{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\ell_1} \to [m]^w$.

(i) Upon receiving $(\textsc{Compute}, \text{sid}, \text{Rec}, Y = (y_1, \ldots, y_{n_2}))$ from the environment $\mathcal{Z}$, Rec:
  (i) Initialize an $m \times w$ binary matrix D to all 1's;
  (ii) Pick random $k \leftarrow \{0,1\}^\lambda$;
  (iii) For $i \in [n_2]$:
    (i) Set $v_i := \text{PRF}_k(\text{hash}_{\ell_1}(y_i))$;
    (ii) For $j \in [w]$, set $D^j[v[j]] = 0$;
  (iv) For $i \in [w]$, send $(\textsc{Send}, \text{sid})$ to $\mathcal{F}^{1,2}_{\text{U-OT}}$ as Sen;
(ii) Upon receiving $(\textsc{Compute}, \text{sid}, \text{Sen}, X = (x_1, \ldots, x_{n_1}))$ from the environment $\mathcal{Z}$, Sen:
  (i) Pick random $s \leftarrow \{0,1\}^w$;
  (ii) For $i \in [w]$, send $(\textsc{Receive}, \text{sid}, s[i])$ to $\mathcal{F}^{1,2}_{\text{U-OT}}$ as Rec;
(iii) Upon receiving $(\textsc{Send}, \text{sid}, \{r_i^0, r_i^1\})$ from $\mathcal{F}^{1,2}_{\text{U-OT}}$ for $i \in [w]$, Rec:
  (i) Form a $m \times w$ binary matrix A where $A^i = r_i^0$, for $i \in [w]$;
  (ii) Set $B = A \oplus D$;
  (iii) For $i \in [w]$, set $\Delta_i = B^i \oplus r_i^1$;
  (iv) Send $\{\Delta_i\}_{i \in [w]}, k$ to Sen;
(iv) Upon receiving $(\textsc{Receive}, \text{sid}, r_i^{s[i]})$ from $\mathcal{F}^{1,2}_{\text{U-OT}}$ for $i \in [w]$ and receiving $\{\Delta_i\}_{i \in [w]}, k$ from Rec, Sen:
  (i) Form a $m \times w$ binary matrix C where $C^i = r_i^{s[i]} \oplus s[i] \cdot \Delta_i$;
  (ii) For $i \in [n_1]$:
    (i) Set $u_i := \text{PRF}_k(\text{hash}_{\ell_1}(x_i))$;
    (ii) Set $\phi_i := \text{hash}_{\ell_2}(C^1[u_i[1]] || \ldots || C^w[u_i[w]])$;
  (iii) Send $\Phi := \{\phi_i\}_{i \in [n_1]}$ to Rec;
  (iv) Return $(\textsc{Compute}, \text{sid})$ to the environment $\mathcal{Z}$;
(v) Upon receiving Φ from Sen, Rec:
  (i) For $i \in [n_2]$, Set $\psi_i := \text{hash}_{\ell_2}(A^1[v_i[1]] || \ldots || A^w[v_i[w]])$;
  (ii) Set $\Psi := \{\psi_i\}_{i \in [n_2]}$;
  (iii) Set $\Omega := \Phi \cap \Psi$;
  (iv) Set $\text{Res} := \{y_i | \psi_i \in \Omega\}$;
  (v) Return $(\textsc{Compute}, \text{sid}, \text{Res})$ to the environment $\mathcal{Z}$;

Figure 20: Private set intersection protocol Π_{PSI} in the $\mathcal{F}^{1,2}_{\text{U-OT}}$-hybrid model.

and we can extend Equation (9) to Equation (10), which proves the correctness of the protocol $\Pi^{N,\text{length}}_{\text{triple}}$.

$$\begin{aligned}\sum_i c_i &\equiv \sum_i \left(a_i \cdot b_i + \sum_{j \neq i} \left(s_{j,i} - m^0_{i,j} \right) \right) \\ &\equiv \sum_i \left(a_i \cdot b_i + \sum_{j \neq i} \left(m^0_{j,i} + b_i \cdot a_j - m^0_{i,j} \right) \right) \\ &\equiv \left(\sum_i a_i \right) \cdot \left(\sum_i b_i \right).\end{aligned} \tag{10}$$

Now, we proceed to provide the security of the protocol $\Pi^{N,\text{length}}_{\text{triple}}$.

Theorem 21. *The protocol $\Pi^{N,length}_{triple}$ described in Figure 18 securely realizes $\mathcal{F}_{triple}$ described in Figure 15 with length $\geq$ 1 in the $\mathcal{F}^{1,2}_{E-OT}$-hybrid model against any PPT semihonest adversary corrupting no more than $N-1$ parties.*

The proof can be found in Appendix B.4.

Given this multiparty multiplication triple-generation protocol $\Pi^{N,\text{length}}_{\text{triple}}$, we can use it in the semihonest SPDZ-style multiparty computation protocol [25], where each multiplication gate consumes one multiplication triple. Note that the original SPDZ protocol is designed for the malicious setting, and it includes information-theoretic MAC to ensure correctness, while in the semihonest setting, these checks can be removed. The details of the resulting MPC protocol Π^{MPC} can be found in Figure 19. Π^{MPC} provides an efficient solution to generic multiparty computation over the arithmetic circuits, which are more powerful than the Boolean circuits.

Theorem 22. *The protocol Π^{MPC} described in Figure 19 securely realizes $\mathcal{F}^f_{mpc}$ described in Figure 1 in the $\mathcal{F}_{triple}$-hybrid model against any PPT semihonest adversary corrupting up to $N-1$ parties.*

The proof is done by combining the result of [37, 51].

7. Private Set Intersection from OT

Apart from generic MPC protocols, there are also protocols dedicated for special usage, e.g., PSI. The PSI protocol Π_{PSI}

Sender Chosen Oblivious Transfer Protocol $\Pi^{1,N}_{\text{S-OT}}$ in the $\mathcal{F}^{1,N}_{\text{E-OT}}$-hybrid model

(i) Upon receiving $(\textsc{Send}, \text{sid}, \text{ssid}, \{m_i\}_{i\in[N]})$ from the environment $\mathcal{Z}$, Sen:
  (i) Send $(\textsc{Send}, \text{sid}, \text{ssid})$ to $\mathcal{F}^{1,N}_{\text{E-OT}}$;
(ii) Upon receiving $(\textsc{Receive}, \text{sid}, \text{ssid}, c)$ from the environment $\mathcal{Z}$, Rec:
  (i) Send $(\textsc{Receive}, \text{sid}, \text{ssid}, c)$ to $\mathcal{F}^{1,N}_{\text{E-OT}}$;
(iii) Upon receiving $(\textsc{Send}, \text{sid}, \text{ssid}, \{k_i\}_{i\in[N]})$ from $\mathcal{F}^{1,N}_{\text{E-OT}}$, Sen:
  (i) For $i \in [N]$, sets $m'_i := m_i \oplus k_i$;
  (ii) Send $\{m'_i\}_{i\in[N]}$ to Rec;
  (iii) Return $(\textsc{Send}, \text{sid}, \text{ssid}, \{m_i\}_{i\in[N]})$ to the environment $\mathcal{Z}$;
(iv) Upon receiving $(\textsc{Receive}, \text{sid}, \text{ssid}, k_c)$ from $\mathcal{F}^{1,N}_{\text{E-OT}}$ and receiving $\{m'_i\}_{i\in[N]}$ from Sen, Rec:
  (i) Set $m_c := m'_c \oplus k_c$;
  (ii) Return $(\textsc{Receive}, \text{sid}, \text{ssid}, m_c)$ to the environment $\mathcal{Z}$;

Figure 21: Sender chosen oblivious transfer protocol $\Pi^{1,N}_{\text{S-OT}}$ in the $\mathcal{F}^{1,N}_{\text{E-OT}}$-hybrid model.

(Figure 5) is taken from the work of Chase and Miao [52], and we instantiate the OT functionality with $\Pi^{1,N}_{\text{E-OT}}$ as another application of this endemic OT protocol. When the required OT number is large, we can use the OT extension technique to extend the number of OT instances.

The security of Π_{PSI} in the semihonest setting directly follows the result of [52] since according to Theorem 13, the protocol $\Pi^{1,N}_{\text{E-OT}}$ (as well as the protocol $\Pi^{\text{semi}}_{\text{OTE}}$) securely realizes the random OT functionality in this setting. Moreover, Chase and Miao consider one-sided malicious security, where the Sen can be maliciously corrupted by the adversary. However, we notice that $\mathcal{F}_{\text{E-OT}}$ does not meet the security requirement of Π_{PSI}, and in such a case, the protocol can be insecure. As stated above, in the malicious setting, $\Pi^{1,N}_{\text{E-OT}}$ only realizes $\mathcal{F}^{1,2}_{\text{E-OT}}$, which allows the adversary $\mathcal{A}$ to control the OT messages. Specifically, $\mathcal{A}$ can influence the protocol output by changing the OT messages, enabling the environment to distinguish between the real world and the ideal world. We provide the result along with its proof below.

Theorem 23. *The protocol Π_{PSI} described in Figure 20 is not secure against a PPT malicious adversary corrupting Rec when the endemic OT functionality $\mathcal{F}^{1,2}_{E\text{-OT}}$ is used even if F is a secure PRF, hash_{ℓ_1} and hash_{ℓ_2} are modeled as random oracles, and parameters m, w, ℓ_1, ℓ_2 are chosen properly.*

Proof. To prove Theorem 23, we construct an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ such that for any PPT simulator $\mathcal{S}$, $\mathcal{Z}$ can distinguish between (i) the real execution exe $c^{\mathcal{F}^{1,2}_{E\text{-OT}}}_{\Pi_{\text{PSI}},\mathcal{A},\mathcal{Z}}$, where the parties $\mathcal{P} := \{\text{Sen}, \text{Rec}\}$ run protocol Π_{PSI} in the $\mathcal{F}^{1,2}_{E\text{-OT}}$-hybrid model and the corrupted Sen is controlled by $\mathcal{A}$, and (ii) the ideal execution $\text{exec}_{\mathcal{F}_{\text{PSI}},\mathcal{S},\mathcal{Z}}$, where the parties Sen and Rec interact with the functionality $\mathcal{F}_{\text{PSI}}$ in the ideal world, and corrupted Sen is controlled by the simulator $\mathcal{S}$. □

7.1. Adversary. The adversary $\mathcal{A}$ instructs Sen to run the protocol faithfully except for the following steps.

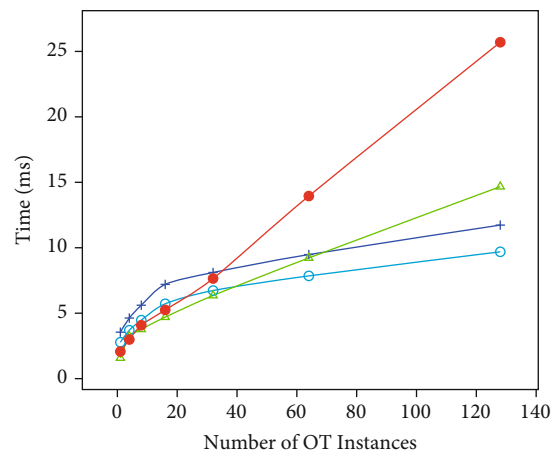

Figure 22: Running OT protocols on LAN.

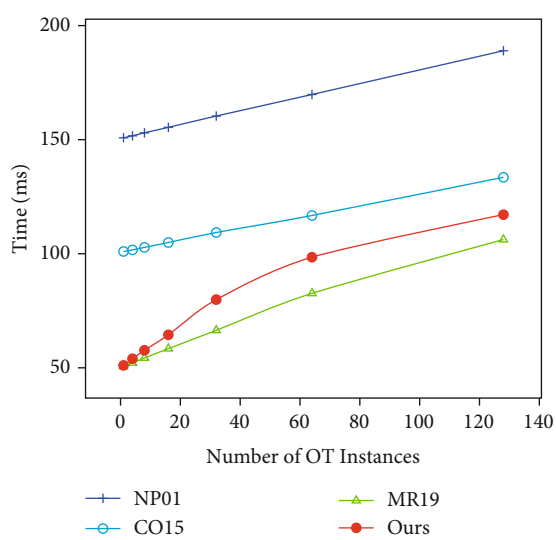

Figure 23: Running OT protocols on WAN.

For $i \in [w]$, upon receiving (SendNotify, sid) from $\mathcal{F}^{1,2}_{E\text{-OT}}$, the adversary $\mathcal{A}$ sends (Receive, sid, 0) to $\mathcal{F}^{1,2}_{E\text{-OT}}$ on behalf of Sen.

Table 2: Running time of OT protocols in milliseconds.

Protocol	Rounds	Num									
		1	8	32	64	128	1	8	32	64	128
		LAN					WAN				
[10]	3	3.56	5.61	8.11	9.48	11.73	150.8	153.0	160.4	169.8	189.0
[11]	2	2.78	4.47	6.73	7.85	9.69	101.0	102.8	109.3	116.8	133.5
[12]	1	1.58	3.77	6.35	9.22	14.67	50.6	54.3	66.4	82.6	106.2
Ours	1	2.07	4.08	7.65	13.95	25.70	51.1	57.7	79.9	98.5	117.1

Table 3: Running times in milliseconds of the semihonest OT extension protocol with different base OT protocols.

Protocol	Num							
	10^4	10^5	10^6	10^7	10^4	10^5	10^6	10^7
	LAN				WAN			
[10]	19	48	220	1550	249	421	1847	15543
[11]	17	45	217	1546	242	414	1842	15439
[12]	23	52	228	1553	214	389	1813	15332
Ours	34	63	238	156	227	404	1826	15410

For $i \in [w]$, upon receiving (ReceiveNotify, sid) from $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{A}$ sends (FixMessage, sid, 0) to $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$.

$\mathcal{A}$ makes poly(λ) random queries to hash[1] after receiving $\{\Delta_i\}_{i\in[w]}, k$ from Rec on behalf of Sen.

The environment $\mathcal{Z}$ outputs 1 if Rec sends (Compute, sid, Res) back where Res $= X \cap Y$, and it outputs 0 otherwise.

This can be seen as a drawback of the endemic OT functionality in that its application scenarios are limited, and sometimes, we need other types of OT functionalities. As depicted in Figure 21, we can adopt the transformation of [53] and obtain $\mathcal{F}_{S-\mathrm{OT}}^{1,N}$ at the cost of one more round communication. After transformation, our endemic OT protocol can be used to construct this highly efficient PSI protocol Π_{PSI} even in the malicious setting. We can also obtain $\mathcal{F}_{U-\mathrm{OT}}$ and $\mathcal{F}_{R-\mathrm{OT}}$ following the protocols in [12].

8. Implementation and Benchmarks

In our protocols, we instantiate all the hash functions involved with SM3 hash function $\mathrm{SM3} : \{0,1\}^* \longrightarrow \{0,1\}^\ell$. When the required output length is not to ℓ, we adopt a similar technique as used in the construction of the key derivation function (cf. Section 4.1). The pseudorandom function PRF and pseudorandom number generator function PRG can be instantiated with the SM4 block cipher algorithm $\mathrm{SM4} : \{0,1\}^\lambda \times \{0,1\}^\ell \longrightarrow \{0,1\}^\ell$. Roughly speaking, to implement $\mathrm{PRF} : \{0,1\}^\lambda \times \{0,1\}^\ell \longrightarrow \{0,1\}^\ell$, we use $\mathrm{PRF}_k(m) = \mathrm{SM4}_k(m)$. To implement $\mathrm{PRG} : \{0,1\}^\lambda \longrightarrow \{0,1\}^m$, where $m = n \cdot \ell$, we use $\mathrm{PRG}(k) = \mathrm{SM4}_k(0) \| \cdots \| \mathrm{SM4}_k(n-1)$. When $\ell \nmid m$, we truncate the extra bits as in Section 4.1. As for other mentioned protocols, we instantiate the hash function, PRF and PRG functions as described in their works, e.g., SHA256, for the hash function, and AES for PRF.

8.1. Experimental Setup. We perform the experiments on Dell OptiPlex 7080 equipped with an Intel Core 8700 CPU @ 3.20 GHz with 32.0 GB RAM, running Ubuntu 18.04 LTS. We evaluate all protocols in two simulated network settings: (i) a LAN setting with 1 Gbps bandwidth and 1 ms delay and (ii) a WAN setting with 100 Mbps bandwidth and 50 ms delay. All test results are the average of 10 tests.

8.2. Oblivious Transfer Evaluation. We first compare the performance of our multi-instance OT protocol $\Pi_{E-\mathrm{OT}}^{\mathrm{num}}$ with several state-of-the-art OT protocols [10, 11] and [12]. Note that the OT protocol in [10] is a sender OT protocol, and it needs an additional round to transfer messages. While [11, 12] and our protocol only generate random correlated messages. Besides, our protocol is based on the SM series cryptography, especially the SM2 key agreement protocol, while the other three protocols are inspired by the Diffie-Hellman key agreement protocol [54].

In Figure 22, we show the running time of the protocols in the LAN setting. Since the SM2 key agreement protocol needs more exponentiation operations than the Diffie-Hellman key agreement protocol, our protocol will be slower than the other protocols when the number of OT instances is large. However, as is shown in Figure 23, in the WAN setting, our protocol is faster than the protocol of Naor and Pinkas [10] as well as the protocol of Chou and Orlandi [11] because our protocol only needs one round as the protocol of Masny and Rindal [12]. Therefore, our protocol is specifically suitable for bad network environments, e.g., the wireless network. We also provide the detailed running time in Table 2. In the LAN setting, [11] is the fastest protocol for a large number of OT instances since it requires the least number of exponentiation operations. And in the WAN setting, [12] is the fastest protocol because it only needs one round of communication, and our protocol is slightly slower

Table 4: Running times in milliseconds of the triple generation protocol with different OT protocols.

Protocol	Num							
	10^4	10^5	10^6	10^7	10^4	10^5	10^6	10^7
	LAN				WAN			
[10]	39	97	452	3203	515	903	4106	32072
[11]	35	91	446	3193	496	884	4098	31723
[12]	46	106	465	3210	436	843	4048	31658
Ours	68	128	490	3238	464	871	4086	31814

than [12]. We note that our protocol is the only one that is based on the SM series cryptography and can be legally used for commercial purposes in China.

In real-life applications, OT extension techniques are often used to generate hundreds of thousands of OT instances with high speed. In such cases, the performance of the base OT protocols only has a minor impact on the performance of the overall protocol. To illustrate this, we provide the test result in Table 3. We use the OT protocols of [10–12] and our multi-instance OT protocol as the base OT for the semihonest setting OT extension. As the number of OT instances increases, the running time of different protocols is relatively closer. In the LAN setting, our protocol only takes 1.563 seconds to generate 10 million OT instances. And in the WAN setting, our protocol can generate the same number of OT instances in 15.41 seconds. Therefore, our protocol is comparable with other OT protocols in many application scenarios.

8.3. Triple-Generation Evaluation. One of the applications of our OT protocol is to generate the Beaver triples, which can be used in many MPC applications. Our Beaver triple-generation protocol Π_{triple} invokes the endemic OT functionality. We use different OT protocols as the base OT protocols for the OT extension protocol, and we use the OT extension protocols for the triple generation protocols. The test results can be found in Table 4, and as one can see, the performance of the triple-generation protocol mainly depends on its underlying OT extension protocols. In the LAN setting, our protocol needs 3.238 seconds to generate 10 million Beaver triples. And in the WAN setting, our protocol can generate the same number of Beaver triples in 31.814 seconds.

9. Conclusion

In this work, we investigate the problem of secure computation from the SM series cryptography, which complies with the Chinese cryptographic laws and is authorized for commercial usages in China. We show how to generate OT using the SM2 and SM3 algorithms. Moreover, we instantiate the OT extension protocols in the semihonest setting and malicious setting with the SM3 and SM4 algorithms, which can efficiently extend some base OTs to a polynomial number of OTs. With the generated OT, we can securely realize the Beaver multiplication triple-generation functionality and further construct generic MPC protocols. Besides, we show that the specific MPC, PSI, can also be implemented using the SM2, SM3, and SM4 algorithms. The proposed protocols are secure in the random oracle model and the public key infrastructure setting. The evaluation results indicate that our constructions are comparable to existing protocols and especially suitable for the wireless network environment. Therefore, we provide an efficient secure computation solution from SM series cryptography, and it is the first solution that can be used for commercial purposes in China.

Appendix

A. Functionalites

A.1. Random Oblivious Transfer. As depicted in Figure 9, the 1-out-of-N endemic OT functionality $\mathcal{F}_{U-\text{OT}}^{1,N}$ waits for (Send, sid, ssid) from Sen and (Receive, sid, ssid, c) from Rec, where $c \in [N]$ denotes Rec's choices. After both messages are obtained, $\mathcal{F}_{U-\text{OT}}^{1,N}$ picks n uniformly random messages $\{m_i\}_{i\in[N]}$. At the end, $\mathcal{F}_{U-\text{OT}}^{1,N}$ sends $\{m_i\}_{i\in[N]}$ to Sen and m_c to Rec.

Figure 13 depicts the multi-instance version of 1-out-of-2 $\mathcal{F}_{U-\text{OT}}$.

B. Proof of Theorems

B.1. Proof of Theorem 13

Proof. To prove Theorem 13, we construct a simulator $\mathcal{S}$ such that for any nonuniform PPT environment $\mathcal{Z}$, the following ensembles are indistinguishable: (i) the real execution $\text{exec}_{\Pi_{E-\text{OT}}^{1,N},\mathcal{A},\mathcal{Z}}$, where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi_{E-\text{OT}}^{1,N}$ and the corrupted party is controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\text{exec}_{\mathcal{F}_{U-\text{OT}}^{1,N},\mathcal{S},\mathcal{Z}}$, where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{U-\text{OT}}^{1,N}$ in the ideal world and the corrupted party is controlled by the simulator $\mathcal{S}$. We consider following cases. □

Case 1. Sen is corrupted; Rec 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 $\mathcal{F}_{U-\text{OT}}^{1,N}$ as well as honest Rec. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$:

(i) Upon receiving (ReceiveNotify, sid, ssid) from the external functionality $\mathcal{F}_{U-\text{OT}}^{1,N}$ and receiving (Send, sid, ssid, c) from the environment $\mathcal{Z}$ for Sen, the simulator $\mathcal{S}$ sends (Send, sid, ssid) to $\mathcal{F}_{U-\text{OT}}^{1,N}$, it then

receives (SendNotify, sid, ssid) and (Send, sid, ssid, $\{k_i\}_{i\in[N]}$) from $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$. For $i \in [N]$, $\mathcal{S}$ picks random $r_i \longleftarrow \mathbb{G}$, and it sends $\{r_i\}_{i\in[N]}$ to Sen

(ii) For $i \in [N]$, when Sen queries the KDF for the i-th time, $\mathcal{S}$ returns k_i

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \cdots, \mathcal{H}_3$.

Hybrid $\mathcal{H}_0$: it is the real protocol execution.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, the simulator $\mathcal{S}$ receives $\{k_i\}_{i\in[N]}$ for corrupted Sen from $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$. The view of Sen is not changed since Rec behaves exactly the same.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that in $\mathcal{H}_2$, Rec picks random $r_i \longleftarrow \mathbb{G}$, for $i \in [N]$. In $\mathcal{H}_1$, $r_c = m_A - \mathrm{hash}^{\mathbb{G}}_c(\{r_i\}_{i\neq c})$, where $\mathrm{hash}^{\mathbb{G}}_c(\{r_i\}_{i\neq c}$ should be indistinguishable from a random element and serve as a one-time pad (OTP); therefore, r_c in $\mathcal{H}_1$ and $\mathcal{H}_2$ should be indistinguishable.

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except that in $\mathcal{H}_3$, the random oracle KDF returns k_i for the i-th query. Because of the key indistinguishability of the SM2 key agreement protocol, $\mathcal{H}_2$ and $\mathcal{H}_3$ should be indistinguishable.

The adversary's view of $\mathcal{H}_3$ is identical to the simulated view. Therefore, $\mathrm{exec}_{\Pi^{1,N}_{E-\mathrm{OT}},\mathcal{A},\mathcal{Z}}$ and $\mathrm{exec}_{\mathcal{F}^{1,N}_{U-\mathrm{OT}},\mathcal{S},\mathcal{Z}}$ are indistinguishable.

Case 2. Rec is corrupted; Sen 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 $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$ as well as honest Sen. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$:

(i) Upon receiving (SendNotify, sid, ssid) from the external functionality $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$ and receiving (Receive, sid, ssid, c) from the environment $\mathcal{Z}$ for Rec, the simulator $\mathcal{S}$ sends (Receive, sid, ssid, c) to $\mathcal{F}^{1,N}_{\mathrm{U-OT}}$, and it then receives (ReceiveNotify, sid, ssid) and (Receive, sid, ssid, k_c) from $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$. $\mathcal{S}$ invokes $(m_B, t_B) \longleftarrow$ MsgGen(), and it sends m_B to Rec

(ii) When Rec queries the KDF, $\mathcal{S}$ returns k_c

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \cdots, \mathcal{H}_2$.

Hybrid $\mathcal{H}_0$: it is the real protocol execution.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, the simulator $\mathcal{S}$ receives m_c for corrupted Rec from $\mathcal{F}^{1,N}_{U-\mathrm{OT}}$. The view of Rec is not changed since Sen behaves exactly the same.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that in $\mathcal{H}_2$, the random oracle KDF returns k_c when Rec queries it. Because of the key indistinguishability of the SM2 key agreement protocol, $\mathcal{H}_1$ and $\mathcal{H}_2$ should be indistinguishable.

The adversary's view of $\mathcal{H}_2$ is identical to the simulated view. Therefore, $\mathrm{exec}_{\Pi^{1,N}_{E-\mathrm{OT}},\mathcal{A},\mathcal{Z}}$ and $\mathrm{exec}_{\mathcal{F}^{1,N}_{U-\mathrm{OT}},\mathcal{S},\mathcal{Z}}$ are indistinguishable.

Case 3. Both Sen and Rec are corrupted.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$.

Indistinguishability. This is a trivial case, since both Sen and Rec are controlled by the adversary $\mathcal{A}$.

B.2. Proof of Corollary 14

Proof. The simulator used to proof Corollary 14 is exactly the same as the simulator used to proof Theorem 13. Although the functionality $\mathcal{F}^{1,N}_{E-\mathrm{OT}}$ allows the simulator to fix the corrupted party's message, we never invoke the (FixMessage, $\cdots$) instruction. □

B.3. Proof of Theorem 19

Proof. To prove Theorem 19, we construct a simulator $\mathcal{S}$ such that for any nonuniform PPT environment $\mathcal{Z}$, the following ensembles are indistinguishable: (i) the real execution $\mathrm{exec}^{\mathcal{F}^{1,2}_{E-\mathrm{OT}}}_{\Pi_{\mathrm{triple}},\mathcal{A},\mathcal{Z}}$, where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol Π_{triple} in the $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$-hybrid model and the corrupted party is controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathrm{exec}_{\mathcal{F}_{\mathrm{triple}},\mathcal{S},\mathcal{Z}}$, where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\mathrm{triple}}$ in the ideal world and the corrupted party is controlled by the simulator $\mathcal{S}$. Since the protocol is symmetric, we only consider the case where P_1 is corrupted. □

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$:

(i) Upon receiving (GenerateNotify, sid, ssid, P_2) from the external $\mathcal{F}_{\mathrm{triple}}$, the simulator $\mathcal{S}$ sends (SendNotify, sid, ssid_2) and (ReceiveNotify, sid, ssid_1) to the adversary $\mathcal{A}$ on behalf of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$. $\mathcal{S}$ also sends (Generate, sid, P_1) to $\mathcal{F}_{\mathrm{triple}}$ and receives (GenerateNotify, sid, P_1)

(ii) Upon receiving (Send, sid, ssid_1) from P_1 via the interface of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$, $\mathcal{S}$ sends (SendNotify, sid, ssid_1) to $\mathcal{A}$ on behalf of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$. Upon receiving (FixMessage, sid, ssid_1, (m^1_0, m^1_1)) from $\mathcal{A}$ via the interface of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$, $\mathcal{S}$ sets $a_1 = m^1_0 \oplus m^1_1$, and it sends (Send, sid, (m^1_0, m^1_1)) to P_1 on behalf of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$

(iii) Upon receiving (Receive, sid, ssid_2, b_1) from P_1 via the interface of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$, $\mathcal{S}$ sends (ReceiveNotify, sid, ssid_2) to $\mathcal{A}$ on behalf of $\mathcal{F}^{1,2}_{E-\mathrm{OT}}$. Upon receiving (FixMessage, sid, ssid_2, $m^2_{b_1}$) from $\mathcal{A}$ via the interface

of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (Receive, sid, $m_{b_1}^2$) to P_2 on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$

(iv) $\mathcal{S}$ sets $c_1 := a_1 \wedge b_1 \oplus m_0^1 \oplus m_{b_1}^2$, and it sends (FixTriple, sid, P_1, (a_1, b_1, c_1)) to $\mathcal{F}_{\mathrm{triple}}$. It outputs whatever $\mathcal{F}_{\mathrm{triple}}$ outputs

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \cdots, \mathcal{H}_3$.

Hybrid $\mathcal{H}_0$: it is the real protocol execution.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, the simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, and it receives b_1 from P_1 and $m_{b_1}^2, m_0^1, m_1^1$ from the adversary $\mathcal{A}$. The view of P_1 is not changed since $\mathcal{S}$ behaves exactly the same as $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that in $\mathcal{H}_2$, the simulator $\mathcal{S}$ sets $a_1 = m_0^1 \oplus m_1^1$ and $c_1 := a_1 \wedge b_1 \oplus m_0^1 \oplus m_{b_1}^2$, and it sends (FixTriple, sid, P_1, (a_1, b_1, c_1)) to the external $\mathcal{F}_{\mathrm{triple}}$ to modify the triple. The view of P_1 is not changed since no message sent to P_1 is changed.

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except that in $\mathcal{H}_3$, the output of P_2 is directly from $\mathcal{F}_{\mathrm{triple}}$. The output distribution remains the same since (1) the simulator modifies (a_1, b_1, c_1) in $\mathcal{H}_2$ to the values obtained by P_1; (2) in both $\mathcal{H}_2$ and $\mathcal{H}_3$, b_2 is randomly picked; (3) in $\mathcal{H}_2$, $a_2 = m_0^2 \oplus m_1^2$, where one of the values is randomly picked, and in $\mathcal{H}_3$, a_2 is randomly picked; and (4) in both $\mathcal{H}_2$ and $\mathcal{H}_3$, it holds that $(a_1 \oplus a_2) \wedge (b_1 \oplus b_2) = c_1 \oplus c_2$.

The adversary's view of $\mathcal{H}_3$ is identical to the simulated view $\mathrm{exec}_{\mathcal{F}_{\mathrm{triple}},\mathcal{S},\mathcal{Z}}$. Therefore, it is perfectly indistinguishable.

B.4. Proof of Theorem 21

Proof. To prove Theorem 21, we construct a simulator $\mathcal{S}$ such that for any nonuniform PPT environment $\mathcal{Z}$, the following ensembles are indistinguishable: (i) the real execution $\mathrm{exec}_{\Pi_{\mathrm{triple}}^{N,\mathrm{length}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{E-\mathrm{OT}}^{1,2}}$, where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi_{\mathrm{triple}}^{N,\mathrm{length}}$ in the $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$-hybrid model and the corrupted party is controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathrm{exec}_{\mathcal{F}_{\mathrm{triple}},\mathcal{S},\mathcal{Z}}$, where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\mathrm{triple}}$ in the ideal world and the corrupted party is controlled by the simulator $\mathcal{S}$. We consider the extreme case where only P_1 is not corrupted. □

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$ as well as honest P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$:

(i) Upon receiving (GenerateNotify, sid, ssid, P_1) from the external $\mathcal{F}_{\mathrm{triple}}$, the simulator $\mathcal{S}$ sends (SendNotify, sid, $\mathrm{ssid}_{1,j,k}$) and (ReceiveNotify, sid, $\mathrm{ssid}_{j,1,k}$) to the adversary $\mathcal{A}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, for $j \neq 1$ and $k \in [\mathrm{length}]$. $\mathcal{S}$ also sends (Generate, sid, P_j) to $\mathcal{F}_{\mathrm{triple}}$ and receives (GenerateNotify, sid, P_j), for $j \neq 1$

(ii) For $j \neq 1, k \in [\mathrm{length}], j' \in [N] \setminus \{1, j\}$:

(a) Upon receiving (Send, sid, $\mathrm{ssid}_{j,1,k}$) from P_j via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (SendNotify, sid, $\mathrm{ssid}_{j,1,k}$) to $\mathcal{A}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$. Upon receiving (FixMessage, sid, $\mathrm{ssid}_{j,1,k}$, $(m_0^{j,1,k}, m_1^{j,1,k})$) from $\mathcal{A}$ via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (Send, sid, $\mathrm{ssid}_{j,1,k}$, $(m_0^{j,1,k}, m_1^{j,1,k})$) to P_j on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$

(b) Upon receiving (Receive, sid, $\mathrm{ssid}_{1,j,k}$, $b_j[k]$) from P_j via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (ReceiveNotify, sid, $\mathrm{ssid}_{1,j,k}$) to $\mathcal{A}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$. Upon receiving (FixMessage, sid, $\mathrm{ssid}_{1,j,k}$, $m_{b_j[k]}^{1,j,k}$) from $\mathcal{A}$ via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (Receive, sid, $m_{b_j[k]}^{1,j,k}$) to P_2 on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$

(c) Upon receiving (Send, sid, $\mathrm{ssid}_{j,j',k}$) from P_j via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (SendNotify, sid, $\mathrm{ssid}_{j,j',k}$) to $\mathcal{A}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$. Upon receiving (Receive, sid, $\mathrm{ssid}_{j,j',k}$, $b_{j'}[k]$) from $P_{j'}$ via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (ReceiveNotify, sid, $\mathrm{ssid}_{j,j',k}$) to $\mathcal{A}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$. Upon receiving (FixMessage, sid, $\mathrm{ssid}_{j,j',k}$, $(m_0^{j,j',k}, m_1^{j,j',k})$) from $\mathcal{A}$ via the interface of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$, $\mathcal{S}$ sends (Send, sid, $\mathrm{ssid}_{j,j',k}$, $(m_0^{j,j',k}, m_1^{j,j',k})$) to P_j and (Receive, sid, $\mathrm{ssid}_{j,j',k}$, $m_{b_{j'}[k]}^{j,j',k}$) to $P_{j'}$ on behalf of $\mathcal{F}_{E-\mathrm{OT}}^{1,2}$

(iii) $\mathcal{S}$ picks random $r_{1,j,k} \longleftarrow \{0,1\}^{\mathrm{length}}$, for $j \neq 1, k \in [\mathrm{length}]$. It then sends $\{r_{1,j,k}\}_{k \in [\mathrm{length}]}$ to P_j, for $j \neq 1$

(iv) For $j \neq 1$, upon receiving $\{r_{j,1,k}\}_{k \in [\mathrm{length}]}$ from P_j for P_1, $\mathcal{S}$ computes $a_j = r_{j,1,1} - m_0^{j,1,1} + m_1^{j,1,1} \bmod 2^{\mathrm{length}}$. After that, $\mathcal{S}$ computes $r_{j,j',k} := a_j + m_0^{j,j',k} - m_1^{j,j',k} \bmod 2^{\mathrm{length}}$, for $j \neq 1, j' \in [N] \setminus \{1, j\}, k \in [\mathrm{length}]$. Subsequently, $\mathcal{S}$ computes $s_{j',j,k} := m_{b_j[k]}^{j',j,k} + b_j[k] \cdot r_{j',j,k} \bmod 2^{\mathrm{length}}$, for $j \neq 1, j' \neq j, k \in [\mathrm{length}]$. At the end, $\mathcal{S}$ computes $c_j = a_j \cdot b_j + (\sum_{j' \neq j} \sum_k (s^{j',j,k} - m_0^{j,j',k}) \cdot 2^{k-1}) \bmod 2^{\mathrm{length}}$, and it sends (*FixTriple*, sid, P_j, (a_j, b_j, c_j)) to $\mathcal{F}_{\mathrm{triple}}$, for $j \neq 1$. It outputs whatever $\mathcal{F}_{\mathrm{triple}}$ outputs

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \cdots, \mathcal{H}_4$.

Hybrid $\mathscr{H}_0$: it is the real protocol execution.

Hybrid $\mathscr{H}_1$: $\mathscr{H}_1$ is the same as $\mathscr{H}_0$ except that in $\mathscr{H}_1$, the simulator $\mathscr{S}$ simulates the functionality $\mathscr{F}_{E-\mathrm{OT}}^{1,2}$ to extract $\{b_j\}_{j\neq 1}$ and obtains $\{m_0^{j,j',k}, m_1 j, j', k\}_{j\neq 1, j'\neq j, k\in[\mathrm{length}]}$ and $\{m_{b_j[k]}^{1,j,k}\}_{j\neq 1, k\in[\mathrm{length}]}$. The view of P_1 is not changed since $\mathscr{S}$ behaves exactly the same as $\mathscr{F}_{E-\mathrm{OT}}^{1,2}$.

Hybrid $\mathscr{H}_2$: $\mathscr{H}_2$ is the same as $\mathscr{H}_1$ except that in $\mathscr{H}_2$, the simulator $\mathscr{S}$ computes $\{a_j, c_j\}_{j\neq 1}$ using the knowledge of $\{b_j\}_{j\neq 1}$, $\{m_0^{j,j',k}, m_1^{j,j',k}\}_{j\neq 1, j'\neq j, k\in[\mathrm{length}]}$, $\{m_{b_j[k]}^{1,j,k}\}_{j\neq 1, k\in[\mathrm{length}]}$, and $\{r_{1,j,k}, r_{j,1,k}\}_{j\neq 1, k\in[\mathrm{length}]}$. It then sends $(\mathrm{FixTriple}, \mathrm{sid}, P_j, (a_j, b_j, c_j))$ to $\mathscr{F}_{\mathrm{triple}}$ to the external $\mathscr{F}_{\mathrm{triple}}$ to modify the triple, for $j \neq 1$. The view of P_1 is not changed since no message sent to P_1 is changed.

Hybrid $\mathscr{H}_3$: $\mathscr{H}_3$ is the same as $\mathscr{H}_2$ except that in $\mathscr{H}_3$, the simulator $\mathscr{S}$ picks random $r_{1,j,k} \longleftarrow \{0,1\}^{\mathrm{length}}$, for $j \neq 1$, $k \in [\mathrm{length}]$, instead of computing $r_{1,j,k} := a_1 + m_0^{1,j,k} - m_1^{1,j,k} \bmod 2^{\mathrm{length}}$. The views of the other parties in $\mathscr{H}_2$ and $\mathscr{H}_3$ have the same distribution since one of $m_0^{1,j,k}, m_1^{1,j,k}$ is uniformly random.

Hybrid $\mathscr{H}_4$: $\mathscr{H}_4$ is the same as $\mathscr{H}_3$ except that in $\mathscr{H}_4$, the output of P_1 is directly from $\mathscr{F}_{\mathrm{triple}}$. The output distribution remains the same since (1) the simulator modifies (a_j, b_j, c_j) in $\mathscr{H}_3$ to the values obtained by P_j; (2) in both $\mathscr{H}_3$ and $\mathscr{H}_4$, a_1 is randomly picked; (3) in both $\mathscr{H}_3$ and $\mathscr{H}_4$, b_1 is randomly picked; and (4) in both $\mathscr{H}_3$ and $\mathscr{H}_4$, it holds that $(\sum_i a_i) \cdot (\sum_i b_i) \equiv \sum_i c_i$. The adversary's view of $\mathscr{H}_4$ is identical to the simulated view. Therefore, it is perfectly indistinguishable.

Data Availability

The data used in the submitted manuscript are available by email contacting the corresponding author.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key R&D Program of China (No. 2021YFB3101601) and the National Natural Science Foundation of China (Grant No. 62072401 and No. 62232002). It is also supported by the "Open Project Program of Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province". This project is supported by Input Output (iohk.io).

References

[1] K. Gulati, R. S. K. Boddu, D. Kapila, S. L. Bangare, N. Chandnani, and G. Saravanan, *A Review Paper on Wireless Sensor Network Techniques in Internet of Things (IoT)*, vol. 51, Materials Today: Proceedings, 2022.

[2] M.-k. Choi, R. J. Robles, C.-h. Hong, and T.-h. Kim, "Wireless network security: vulnerabilities, threats and countermeasures," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 3, no. 3, pp. 77–86, 2008.

[3] A. Kavianpour and M. C. Anderson, "An overview of wireless network security," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pp. 306–309, New York, NY, USA, June 2017.

[4] Z. Hu, L. Wang, L. Qi, Y. Li, and W. Yang, "A novel wireless network intrusion detection method based on adaptive synthetic sampling and an improved convolutional neural network," *IEEE Access*, vol. 8, pp. 195741–195751, 2020.

[5] P. Manickam, K. Shankar, E. Perumal, M. Ilayaraja, and K. S. Kumar, "Secure data transmission through reliable vehicles in vanet using optimal lightweight cryptography," in *Cybersecurity and secure information systems*, pp. 193–204, Springer, 2019.

[6] M. Šarac, N. Pavlović, N. Bacanin, F. al-Turjman, and S. Adamović, "Increasing privacy and security by integrating a blockchain secure interface into an iot device security gateway architecture," *Energy Reports*, vol. 7, pp. 8075–8082, 2021.

[7] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*, Cambridge University Press, 2015.

[8] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164, Chicago, IL, USA, November 1982.

[9] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 20–31, Chicago, Illinois, USA, 1988.

[10] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pp. 448–457, Washington, DC, USA, 2001.

[11] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *Progress in Cryptology – LATINCRYPT 2015*, pp. 40–58, Springer, 2015.

[12] D. Masny and P. Rindal, "Endemic oblivious transfer," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 309–326, London, UK, November 2019.

[13] S. Garg, Y. Ishai, and A. Srinivasan, "Two-round mpc: information-theoretic and black-box," in *Theory of Cryptography. TCC 2018*, pp. 123–151, Springer, 2018.

[14] I. McQuoid, M. Rosulek, and L. Roy, "Minimal symmetric pake and 1-out-of-n ot from programmable-once public functions," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 425–442, October 2020.

[15] I. McQuoid, M. Rosulek, and L. Roy, "Batching base oblivious transfers," in *Advances in Cryptology – ASIACRYPT 2021. ASIACRYPT 2021*, pp. 281–310, Springer, 2021.

[16] D. Mahto and D. K. Yadav, "RSA and ECC: a comparative analysis," *International Journal of Applied Engineering Research*, vol. 12, no. 19, pp. 9053–9061, 2017.

[17] M. Bafandehkar, S. M. Yasin, R. Mahmod, and Z. M. Hanapi, "Comparison of ECC and RSA algorithm in resource constrained devices," in *2013 international conference on IT convergence and security (ICITCS)*, pp. 1–3, Macao, China, December 2013.

[18] Standing Committee of the National People's Congress, "Public key cryptographic algorithm SM2 based on elliptic curves,"

April 2022, https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002386/files/b791a9f908bb4803875ab6aeeb7b4e03.pdf.

[19] State Cryptography Administration of China, "Cryptography law of the People's Republic of China," April 2022, http://www.npc.gov.cn/npc/c30834/201910/6f7be7dd5ae5459a8de8baf36296bc74.shtml.

[20] J. Garay, Y. Ishai, R. Ostrovsky, and V. Zikas, "The price of low communication in secure multi-party computation," in *Advances in Cryptology – CRYPTO 2017. CRYPTO 2017*, pp. 420–446, Springer, 2017.

[21] C. Gentry, S. Halevi, H. Krawczyk et al., "YOSO: you only speak once," in *Advances in Cryptology – CRYPTO 2021*, pp. 64–93, Springer, 2021.

[22] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Advances in Cryptology – EUROCRYPT 2012*, pp. 483–501, Springer, 2012.

[23] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 1219–1234, New York, USA, May 2012.

[24] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," in *Advances in Cryptology – EUROCRYPT 2016*, pp. 735–763, Springer, 2016.

[25] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012. CRYPTO 2012*, pp. 643–662, Springer, 2012.

[26] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 21–37, Dallas, Texas, USA, October 2017.

[27] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 39–56, Dallas, Texas, USA, October 2017.

[28] H. Carter, B. Mood, P. Traynor, and K. R. B. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," in *USENIX Security*, pp. 289–304, USENIX Association, 2013.

[29] D. Demmler, T. Schneider, and M. Zohner, "Ad-hoc secure two-party computation on mobile devices using hardware tokens," in *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 893–908, San Diego, CA, USA, 2014.

[30] S. Felsen, Á. Kiss, T. Schneider, and C. Weinert, "Secure and private function evaluation with intel SGX," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp. 165–181, London, UK, November 2019.

[31] Y. Lu, B. Zhang, H.-S. Zhou, W. Liu, L. Zhang, and K. Ren, "Correlated randomness teleportation via semi-trusted hardware—enabling silent multi-party computation," in *Computer Security – ESORICS 2021*, pp. 699–720, Springer, 2021.

[32] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *Security Protocols*, pp. 125–136, Springer, 1997.

[33] P. Kocher, J. Horn, A. Fogh et al., "Spectre attacks: exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, San Francisco, CA, USA, May 2019.

[34] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 136–145, Newport Beach, CA, USA, October 2001.

[35] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security - CCS '93*, pp. 62–73, Fairfax, Virginia, USA, December 1993.

[36] R. Canetti and H. Krawczyk, "Universally composable notions of key exchange and secure channels," in *Advances in Cryptology — EUROCRYPT 2002*, pp. 337–351, Springer, 2002.

[37] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*, New York, NY, USA, January 1987.

[38] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167, Toronto, ON, Canada, October 1986.

[39] A. Yang, J. Nam, M. Kim, and K.-K. R. Choo, "Provably-secure (Chinese government) SM2 and simplified SM2 key exchange protocols," *The Scientific World Journal*, vol. 2014, Article ID 825984, 8 pages, 2014.

[40] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology — CRYPTO' 93*, pp. 232–249, Springer, 1993.

[41] M. Bellare and P. Rogaway, "Provably secure session key distribution: the three party case," in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing - STOC '95*, pp. 57–66, Las Vegas, Nevada, USA, May 1995.

[42] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, pp. 479–488, Philadelphia, Pennsylvania, USA, July 1996.

[43] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer extensions with security for malicious adversaries," in *Advances in Cryptology – EUROCRYPT 2015*, pp. 673–701, Springer, 2015.

[44] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology - CRYPTO 2003*, pp. 145–161, Springer, 2003.

[45] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pp. 535–548, Berlin, Germany, November 2013.

[46] M. Keller, E. Orsini, and P. Scholl, "Actively secure ot extension with optimal overhead," in *Advances in Cryptology – CRYPTO 2015. CRYPTO 2015*, pp. 724–741, Springer, 2015.

[47] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 980–997, San Francisco, CA, USA, May 2018.

[48] V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *Advances in Cryptology – CRYPTO 2013*, pp. 54–70, Springer, 2013.

[49] M. Orrù, E. Orsini, and P. Scholl, "Actively secure 1-out-of-n ot extension with application to private set intersection," in *Topics in Cryptology – CT-RSA 2017*, pp. 381–396, Springer, 2017.

[50] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, pp. 186–194, Springer, 1986.

[51] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology — CRYPTO '91*, pp. 420–432, Springer, 1991.

[52] M. Chase and P. Miao, "Private set intersection in the internet setting from lightweight oblivious PRF," in *Advances in Cryptology – CRYPTO 2020*, pp. 34–63, Springer, 2020.

[53] D. Beaver, "Precomputing oblivious transfer," in *Advances in Cryptology — CRYPT0' 95*, pp. 97–109, Springer, 1995.

[54] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.