Actively Private and Correct MPC Scheme in $t < n/2$ from Passively Secure Schemes with Small Overhead

Dai IKARASHI, Ryo KIKUCHI, Koki HAMADA, and Koji CHIDA

NTT Corporation,
{ikarashi.dai, kikuchi.ryo, chida.koji, hamada.koki}@lab.ntt.co.jp

Abstract. Recently, several efforts to implement and use an unconditionally secure multi-party computation (MPC) scheme have been put into practice. These implementations are *passively* secure MPC schemes in which an adversary must follow the MPC schemes. Although passively secure MPC schemes are efficient, passive security has the strong restriction concerning the behavior of the adversary. We investigate how secure we can construct MPC schemes while maintaining comparable efficiency with the passive case, and propose a construction of an *actively* secure MPC scheme from passively secure ones. Our construction is secure in the $t < n/2$ setting, which is the same as the passively secure one. Our construction operates not only the theoretical minimal set for computing arbitrary circuits, that is, addition and multiplication, but also high-level operations such as shuffling and sorting. We do not use the broadcast channel in the construction. Therefore, privacy and correctness are achieved but *robustness* is absent; if the adversary cheats, a protocol may not be finished but anyone can detect the cheat (and may stop the protocol) without leaking secret information. Instead of this, our construction requires $O((c_B n + n^2)\kappa)$ communication that is comparable to one of the best known passively secure MPC schemes, $O((c_M n + n^2)\log n)$, where κ denote the security parameter, c_B denotes the sum of multiplication gates and high-level operations, and c_M denotes the number of multiplication gates. Furthermore, we implemented our construction and confirmed that its efficiency is comparable to the current fastest passively secure implementation.

Keywords: Multi-party computation, Unconditional security, Active adversary

1 Introduction

Multi-party computation (MPC) is a technique that enables parties with inputs to evaluate a function on the inputs while keeping them secret. MPC has been a central themes of cryptographic study because of its applicability and generality, and MPC theory was developed in the period from the mid-1980s to the mid-2000s, Recently, some sophisticated methodologies to construct MPC schemes have been developed. These includes hardware that is much more efficient than that of decades ago, and MPC schemes that efficiently compute "high-level" operations such as bit-decomposition, shuffling and sorting. Thus, several efforts to implement and use MPC have been put into practice [5, 6, 18]. The (k, n)-threshold unconditionally secure MPC is the most frequently used MPC scheme since it is more efficient compared with other schemes. It requires no heavy operations that require milliseconds such as modular exponentiations.

Let n and t denote the number of parties and corrupted parties, respectively. Most MPC implementations for practical use, including the above, are secure against a *passive* adversary regarding corruption of the $t < n/2$ setting, i.e., they are secure against the adversary that follows an MPC scheme with honest majority. Although *active* security, where the adversary can carry out arbitrary behavior, can be achieved, passively secure MPC schemes are much more efficient than actively secure ones, and the current practical results have been passive secure.

However, passive security requires a somewhat strong restriction concerning the behavior of the adversary. Therefore, it should be motivated to replace passively secure MPC schemes to more secure (active) ones for practical use. In other words, *"How secure can we construct MPC schemes while maintaining comparable efficiency to passive ones?"*. If we aim to use actively secure MPC schemes in practice the same way we do passively secure ones, the following three points need to be satisfied. First, the amount of communication should be small and comparable to the passive setting, which is $O(c_M n \log n + n^2 \log n)$ [12], where c_M is the number of multiplication gates. The communication cost is the main bottleneck in unconditionally secure MPC schemes since local operations conducted by parties typically consist of addition/subtraction, and multiplication/division on a small field. Second is that it should have the same threshold, i.e., it should tolerate $t < n/2$ corruptions. An additional party not only increases communications but also results in a complex MPC system. Third is that high-level operations should be possible. Actual application of MPC schemes involves computation of complex functions such as statistical analysis and database operations. In the passive setting, these functions are efficiently computed by using not only an algebraic circuit but also high-level operations. If the above three points are satisfied, it is possible to use actively secure MPC schemes instead of passively secure ones.

Table 1. Comparison of current circuit-based MPC protocols

	Adversary	Robustness	Threshold	Communication (bits)	Building blocks	Security
HM01 [16]	active	yes	$t < n/3$	$O(c_M n^2\kappa) + poly(n\kappa)$	algebraic circuit	unconditional
DN07 [12]	active	yes	$t < n/3$	$O(c_M n \log n + d_M n^2 \log n) + poly(n\kappa)$	algebraic circuit	unconditional
BH08 [2]	active	yes	$t < n/3$	$O(c_M n \log n + d_M n^2 \log n + n^3 \log n)$	algebraic circuit	perfect
CDD+99 [8]	active	yes	$t < n/2$	$O(c_M n^5\kappa + n^4\kappa) + O(c_M n^4\kappa)\mathcal{BC}$	algebraic circuit	unconditional
BH06 [1]	active	yes	$t < n/2$	$O(c_M n^2\kappa + n^5\kappa^2) + O(n^3\kappa)\mathcal{BC}$	algebraic circuit	unconditional
BFO12 [3]	active	yes	$t < n/2$	$O(c_M(n\phi + \kappa) + d_M n^2\kappa + n^7\kappa) + O(n^3\kappa)\mathcal{BC}$	algebraic circuit	unconditional
Ours	active	no	$t < n/2$	$O((c_B n + n^2)\kappa)$	passively secure MPC	unconditional
DN07 [12]	passive	-	$t < n/2$	$O((c_M n + n^2)\log n)$	algebraic circuit	perfect

"Active" means an adversary can do arbitrary things, "passive" means the adversary must follow the protocol, "yes" means the protocol must be finished whatever the adversary does, "no" means the protocol may not be finished while the parties can detect and stop the protocol without leaking secret information, t is the number of corrupted parties, n is the number of all parties, c_M is the number of multiplication gates of the circuit, d_M is the multiplicative depth of the circuit, $x\mathcal{BC}$ means that x bits are communicated via the broadcast channel, c_B is the number of building blocks that consist of multiplication gates and high-level operations. Note that in Ref. [8], there are two descriptions of $O(n^4)$ and $O(n^5)$ communication via broadcast. The correct one is the former.

1.1 Related Works and Our Results

There have been studies on the communication cost for actively secure MPC schemes that compute algebraic circuits. We list some studies in Table 1. Regarding $t < n/3$, Damgård and Nielsen [12] and Beerliová-Trubíniová and Hirt [2] proposed unconditional and perfect MPC schemes. Their schemes require a small communication cost that is comparable to passively secure ones but they tolerate smaller corruptions. Regarding $t < n/2$, Beerliová-Trubíniová and Hirt [1] proposed an actively secure MPC scheme with $O(c_M n^2\kappa + n^5\kappa^2) + O(n^3\kappa)\mathcal{BC}$ communications, and Ben-Sasson et al. [3] also proposed a scheme with $O(c_M(n\phi + \kappa) + d_M n^2\kappa + n^7\kappa) + O(n^3\kappa)\mathcal{BC}$ communications, where κ denotes a security parameter, d_M denotes the multiplicative depth of the circuit, ϕ is a larger element either a field size or $\log n$, and $\mathcal{BC}$ denotes a *broadcast channel*. The broadcast channel is a communication channel that guarantees that "all recipients are convinced that all other parties receive the same data that they received." To our knowledge, the broadcast channel costs $O(n^3)$ communication at least [17] and requires trusted setup in the $t < n/2$ setting. Therefore, the communication cost of the above two schemes can be regarded as $O(c_M n^2\kappa + n^5\kappa^2 + n^6\kappa)$ and $O(c_M(n\phi + \kappa) + d_M n^2\kappa + n^7\kappa)$, respectively. If the circuit is large, i.e., c_M is much larger than n^7, and "wide", i.e., d_M is much smaller than c_M, the amortized communication complexity of Ben-Sasson et al. 's scheme is $O(n \log n)$ per multiplication, which is the same as the best passively secure MPC scheme [12].

The above studies mainly focused on the theory of MPC, which is insufficient for practice use. The circuit is not always very large or wide, and the effect of a high-dimensional factor in the communication cost such as $O(n^7\kappa)$ cannot be ignored Furthermore, the MPC scheme that computes a high-level operation, for example, bit-decomposition [11], shuffling [21], or sorting [15], is useful for efficient MPC execution, but the above results only support an algebraic circuit as a building block.

One of the main causes of a high-dimensional factor in communication is the broadcast channel. Therefore, it is natural that one attempts to construct an actively secure MPC scheme without the broadcast channel. There has been much less progress in the direction of constructing actively secure without the broadcast channel regarding the $t < n/2$ setting. One of the reason for this is that in this setting, *robustness* cannot be achieved. Robustness guarantees that "an MPC scheme must be finished correctly whatever an adversary does". However, even in the $t < n/2$ setting, an MPC scheme can achieve *correctness* and *privacy*, which guarantee that if the adversary cheats, everyone can detect it (and may stop the protocol) without leaking secret information. To achieve the objective of constructing an actively secure MPC scheme while maintaining the efficiency of a passively secure one, this setting is worth studying.

From the viewpoint of high-level operation, current MPC schemes that compute high-level operations were designed in the paradigm of computing on shared values. In this paradigm, secret values are preliminarily shared with a secret-sharing scheme to all parties that participate in the MPC schemes. Then the MPC schemes take secretly shared values as inputs from each party and output the result in secretly shared form. Therefore, if we generally use MPC schemes in this paradigm as building blocks for constructing an actively secure MPC scheme, many high-level operations can be used.

We propose a construction of a non-robust, actively and unconditionally secure MPC scheme in the $t < n/2$ setting without the broadcast channel while maintaining efficiency comparable to a passively secure one. Our scheme also achieves comparable communication complexity, $O((c_B n + n^2)\kappa)$, where c_B is the number of building blocks consisting of multiplication gates and high-level operations. We construct the actively secure MPC scheme from passively secure ones whose inputs and outputs are in secret-shared form and which should satisfy a weak tamper-resilience. Intuitively, tamper-resilience means that an adversary can tamper with the results of the protocol only by adding values he/she knows, and in fact, we show that most passively secure MPC protocols satisfy it. Therefore, we can apply various known techniques of passive security to actively secure MPC schemes as a building block. In addition, to our knowledge, our scheme is the

first actively secure MPC scheme that has no assumption in the $t < n/2$ setting since the current results uses the broadcast channel that implicitly requires a trusted setup.

1.2 Brief Explanation of Our Construction

At the start of the protocol, each party has its own input. The parties distribute their inputs through (k, n) threshold secret-sharing, and then check the consistency of the shares. Consistency means that for any subset that contains k honest parties, the revealed values are the same. It is known that the consistency can be easily batch checked if a negligible error is allowed by using a plain randomness and random share. A more detailed description can be found in Appendix G.

Each party has consistent shares at this time. This situation is the same in the paradigm of computing on shared values. We perform a protocol to compute a function by using passively secure MPC schemes as building blocks. More precisely, our scheme takes the following three phases.

Randomization Phase: This phase converts shares into *randomized shared pairs* to prevent an adversary from tampering with shared values. Intuitively speaking, the Randomization Phase generates the shares that can be seen as a MAC or checksum of shared values. In the simplest case, this phase changes $(\![a]\!])$ to $(\![a]\!], [\![ra]\!])$ and stores $[\![r]\!]$, where r is uniformly at random and unknown for any party. We formalize it in general as follows. A randomized shared pair is formed as a pair of an element on a ring $\mathcal{X}$, which parties use to conduct computation, and an element of $\mathcal{X}$*-algebra* $\mathcal{Y}$. Namely, in the simplest case, $[\![a]\!]$ belongs to $\mathcal{X}$ and $[\![ra]\!]$ belongs to $\mathcal{Y}$. This generalization makes it possible to use our construction on not only a field but also a ring, as used in [10], and even if the size of $\mathcal{X}$ is small, our scheme is secure by enlarging $\mathcal{Y}$.

Computation Phase: This phase computes the target function redundantly on $\mathcal{X}$ and $\mathcal{Y}$. We denote a simple case as an example. Let $F = f_1 \circ f_2$ be the target function, Π_{f_1}, Π_{f_2} be MPC schemes that are designed in the paradigm of computing on shared values, and $(\![a]\!], [\![ra]\!])$ be input. The Computation Phase first computes $(\![f_2(a)]\!], [\![r f_2(a)]\!])$ from $(\![a]\!], [\![ra]\!])$ via Π_{f_2} then computes $(\![f_1 \circ f_2(a)]\!], [\![r(f_1 \circ f_2)(a)]\!]) = (\![F(a)]\!], [\![rF(a)]\!])$ via Π_{f_1}. Constitutive (passively secure) MPC schemes, Π_{f_1}, Π_{f_2}, should satisfy two properties. The first is the operation on $\mathcal{Y}$-distribution, i.e., Π_{f_2} can compute from $(\![a]\!], [\![ra]\!])$ to $(\![f_2(a)]\!], [\![r f_2(a)]\!])$. The second is *tamper-simulatability*, which means "An adversary's ability to tamper with the results of the protocol is restricted to the addition of values he/she knows". This second property is needed in the next phase. As long as the above two conditions hold, Π_{f_1}, Π_{f_2} are arbitrary so we use not only multiplication but also high-level operations.

Proof Phase: This phase determines if a computation has been cheated. The results of the computation are all checked at once by proving that the results on $\mathcal{X}$ and $\mathcal{Y}$ are "equal". In the above example, the parties reveal

$$[\![r]\!](\rho_1[\![a]\!] + \rho_2[\![f_2(a)]\!] + \rho_3[\![F(a)]\!]) - (\rho_1[\![ra]\!] + \rho_2[\![r f_2(a)]\!] + \rho_3[\![rF(a)]\!])$$

and check if it is 0 or not, where ρ_i is uniformly at random for $i = \{1, 2, 3\}$. If the adversary changes from $[\![a]\!]$ to $[\![a + \delta]\!]$, this equation does not hold except with negligible probability since tamper-simulatability guarantees that δ is known to an adversary (and it inherently says that δ does not depends on r). The concentration of all proofs on one element of $\mathcal{Y}$ makes the proof very efficient and reduces the number of times unnecessary information can be revealed.

As a result of the above phases, each player has the share of output $[\![F(a)]\!]$. The parties perform an actively secure reveal protocol (described in Appendix G) and obtain the result.

1.3 Paper Organization

In Section 2, we introduce known tools and the notations used in the paper, and in Section 3, we explain the building blocks of our construction. In Section 4, we propose our construction that involves converting passive MPC schemes to an active one. In Section 5, we describe the experimental results to demonstrate the practical efficiency of our construction.

2 Preliminaries

We introduce some preliminaries, an algebra that is an algebraic structure used in our construction, our notations, and the passive unconditionally secure MPC protocols used in the examples of our construction.

2.1 Algebra

We use the notion of *algebra* in our construction. Roughly speaking, an algebra is a vector space whose scalar space is not a field but a ring.

Definition 1. *($\mathcal{X}$-algebra)*
A ring $\mathcal{Y}$ is called an $\mathcal{X}$-algebra if there exist another ring $\mathcal{X}$ and an operation, scalar multiplication, between $\mathcal{X}$ and $\mathcal{Y}$ that satisfies the following condition for any $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$.

$$x(y+y') = xy + xy', (x+x')y = xy + x'y, (xx')y = x(x'y), 1_{\mathcal{X}}y = y$$

Example 1. For any field $\mathcal{F}$, its extension $\mathcal{E}(\mathcal{F})^d$ with arbitrary positive integer d is an $\mathcal{F}$-algebra, where scalar multiplication between $\mathcal{F}$ and $\mathcal{E}(\mathcal{F})^d$ is $xy = (xy_0, \cdots, xy_{d-1})$ for any $x \in \mathcal{F}$ and $y \in \mathcal{E}(\mathcal{F})^d$.

2.2 Secret Sharing

We use the (k, n) threshold secret sharing scheme; a secret is separated into n pieces called shares and sent to parties. Parties can then reveal the secret from k or more shares. We assume a secret sharing scheme is Shamir's on a field, or a replicated secret sharing scheme on a field/ring. However, another secret sharing scheme that satisfies the following requirements can be used instead.

- Perfect privacy: the joint distribution of any $k-1$ shares does not depend on the secret.
- Uniqueness of shared value: any k shares determines a unique shared value.
- Existence of mandatory building blocks: there are MPC protocols called *mandatory building blocks*, described later in Section 3. Roughly speaking, we require passively secure scalar multiplication, scalar sum-product, addition/subtraction, and actively secure random number generation and revealing in the secret sharing scheme.

Note that uniqueness of a shared value implies the existence of the share regeneration algorithm, which computes a share from other k shares. Of course, Shamir's and replicated secret sharing schemes satisfy the above requirements. For example, Shamir's satisfies the second condition since k shares uniquely determine the $k-1$ polynomial on $\mathcal{F}$.

Additionally, we say a (k, n) threshold secret sharing scheme is an LSSS (Linear Secret Sharing Scheme) if both the reconstruction and the share regeneration of the scheme are represented by linear combinations of field/ring elements in shares with fixed coefficients. Shamir's and replicated secret sharing schemes are both LSSS.

2.3 Common Structures and Notations

We use the following structures in this paper.

- $\mathcal{X}$: An arbitrary ring on which parties wish to conduct their computation
- $\mathcal{Y}$: An $\mathcal{X}$-algebra
- $\mathcal{F}$: An arbitrary field as an example of rings
- $\mathcal{E}(\mathcal{F})^d$: A d-degree extension field of $\mathcal{F}$ for some positive integer d as an example of $\mathcal{F}$-algebras
- $\mathcal{X}r$: $\{ar \in \mathcal{Y} \mid a \in \mathcal{X}\}$ for some $r \in_R \mathcal{Y}$

Additionally, we use the following notations in this paper.

- "Share" and "shared value" denote each party's share and the tuple of shares of all parties, respectively. Shares of a shared value x are denoted by $[\![x]\!]$.
- A share of a shared value x for a party P_i is denoted by $[\![x]\!]_i$, and ones for a subset of parties $\mathbb{Q}$ are denoted by $[\![x]\!]_{\mathbb{Q}}$.
- $[\![\mathcal{X}]\!]$ denotes the set of arbitrary shared values of $\mathcal{X}$.
- $\langle\!\langle \mathcal{X}r \rangle\!\rangle$ denotes $[\![\mathcal{X}]\!] \times [\![\mathcal{X}r]\!]$.
- $\langle\!\langle a \rangle\!\rangle_r$ (or $\langle\!\langle a \rangle\!\rangle$) denotes $([\![a]\!], [\![ar]\!]) \in \langle\!\langle \mathcal{X}r \rangle\!\rangle$.
- t, k, and n denote the number of corrupted parties, the threshold of the secret sharing scheme, and the number of parties, respectively. Note that $k = t + 1$.
- F, m, and μ denote the function that parties compute, the number of inputs of F, and the number of outputs of F, respectively.

2.4 Known Protocols used in Passive Setting

Our construction is a conversion to an active scheme from passive schemes; thus, we require protocols in the passive scheme, that is, random number generation, multiplication, and reveal.

Random Number Generation A random number generation (RNG) protocol creates a shared value whose plaintext is uniformly random in $\mathcal{X}$. If one allows the security of pseudorandom numbers, pseudorandom secret sharing [9], which realizes an RNG protocol with no communication, can be used. Otherwise, RNG using a Van der Monde matrix with the $O(n)$ communication and one round [12] (DN-RNG) is an efficient way. When $n = 2t + 1$, although these protocols are very light-weight, both are naturally active secure protocols, even if an overall MPC scheme is passive.

Passive Multiplication Multiplication is the main protocol in most MPC schemes because addition tends to be involved in the homomorphism of the underlying secret sharing scheme or encryption; thus, multiplication is sufficient to compute arbitrary circuits. Unlike RNG, there is no multiplication protocol that satisfies efficiency, active security, and simplicity, especially in the $t < n/2$ setting.

We introduce two protocols: GRR-(passive) multiplication [13] and DN-(passive) multiplication [12]. They are described in detail in Appendix A. Both protocols are based on Shamir's secret sharing. They are passive protocols; however, they have a certain weak tamper-resistance as we will discuss in Section 3. GRR-multiplication is $O(n^2)$ communication and one round, and DN-multiplication is $O(n)$ communication and two rounds. When t is small (e.g., $t = 1$), GRR-multiplication is better in terms of not only round efficiency, but also communication efficiency thanks to its small constant coefficient.

Reveal The reveal protocol reconstructs a shared value and publishes the reconstructed plaintext to all parties. Although there are $O(n)$ passive reveal protocols [12], a reveal protocol in our construction requires correctness against an active adversary. Note that the correct reveal protocol is used only once in the *Proof Phase*, and passively secure protocols are allowed in other parts such as the sub-protocol of DN-multiplication in the *Computation Phase*. An example of a correct reveal protocol in Shamir's secret sharing scheme is given in Appendix A. This costs $O(n^2)$ communications and two rounds.

Passive Shuffling Protocol Recently, the shuffling operation has come to be recognized as a significant operation in MPC. It can be used for data filtering [21] and sorting [15]. Although the shuffling operation can be realized as a logical circuit, it is quite heavy. Therefore, more efficient shuffling protocols have been proposed by Laur et al. [21].

They proposed passive and active protocols. The passive protocols are $t < n/2$ protocols. However, the active protocols are $t < n/3$ protocols. Our construction can convert a passive protocol into a $t < n/2$ non-robust one.

3 Available Building Blocks

In this section, we introduce passive MPC protocols used in our construction as building blocks, and we also introduce the two required conditions for them, *tamper-simulatability* and $\mathcal{Y}$*-distribution*. The building blocks are separated into the following two types.

1. *Mandatory building blocks*, which constitute the two phases of our construction: the Randomization Phase and the Proof Phase. They are required regardless of the function F that parties wish to compute. Mandatory building blocks should satisfy tamper-simulatability, which restricts the adversary's ability to cheat as only the addition of known values. (Only one reveal requires active correctness by itself.)
2. *Optional building blocks*, which are selectively used and constitute the Computation Phase. Parties can construct the circuit that realizes the desired function F through composition of optional building blocks. Optional building blocks should satisfy tamper-simulatability and the existence of $\mathcal{Y}$-distribution, which are their realization on the $\mathcal{X}$-algebra $\mathcal{Y}$.

Readers might assume that the two conditions limit the generality of our construction; however, we show that these conditions are quite easy to satisfy for various well-known primitive operations in unconditionally secure MPC schemes.

3.1 Tamper-Simulatability

In our construction, all building blocks require *tamper-simulatability*, which is a kind of weak tamper-resistance and means that "an adversary's ability to tamper with the results of the protocol is restricted to the addition of values he/she knows." From the viewpoint of correctness, this property provides the following benefit. In the first phase, namely, the randomization Phase, each input $[\![a]\!] \in [\![\mathcal{X}]\!]$ is converted into a randomized shared pair $([\![a]\!], [\![ar]\!]) \in [\![\mathcal{X}]\!] \times [\![\mathcal{Y}]\!]$ by multiplying $[\![r]\!]$, where $r \in \mathcal{Y}$ is a random value that no party knows, and in the following Computation Phase, all computations are conducted in the form of randomized shared pairs. Tamper-simulatability guarantees that even if the adversary tampers with a randomized shared value $([\![a]\!], [\![ar]\!])$ to $([\![a']\!], [\![b']\!])$, a' and b' are always represented as $a + x$ and $ar + y$ using x and y, which are independent of r. Therefore, honest parties can detect the existence of tampering by testing whether $r(a + x) - (ra + y)(= rx - y) = 0$ holds, since $rx - y$ is random for the adversary and he/she cannot force it to be zero.

Tamper-simulatability is defined for protocols whose inputs and outputs are in secret-shared form. We call the difference between a legitimate output and a tampered output *tamper-difference* (i.e., when the adversary tampers with a shared value $[\![f(a)]\!]$ to $[\![f(a) + x]\!]$, the tamper-difference is x). We define tamper-simulatability in the manner that for any adversary, there exists a simulator who has only the same information as the adversary, and he/she can compute the tamper-difference.

Let $\mathbb{I}$ be the set of corrupted parties and $[\![a]\!]_{\mathbb{I}}$ be the set of shares of corrupted parties.

Definition 2. *(tamper-simulatability)*
Let Π_f be a protocol that realizes the function $f : \mathcal{X}^m \to \mathcal{X}^\mu$, $[\![\overrightarrow{a}]\!] = ([\![a_0]\!], \ldots, [\![a_{m-1}]\!])$ be inputs of Π_f, $[\![\overrightarrow{b}]\!] = ([\![b_0]\!], \ldots, [\![b_{\mu-1}]\!])$ be the legitimate outputs of the function $f([\![\overrightarrow{a}]\!])$, and $[\![\overrightarrow{b'}]\!] = ([\![b'_0]\!], \ldots, [\![b'_{\mu-1}]\!])$ be the actual (possibly tampered with) outputs of Π_f conducted with an active adversary. We say that Π_f has tamper-simulatability if and only if, for any adversary with any auxiliary input aux, there exists a simulator $\mathcal{S}$ that satisfies

$$\Pr\left[\overrightarrow{e} \leftarrow \mathcal{S}(aux, [\![\overrightarrow{a}]\!]_{\mathbb{I}}, [\![\overrightarrow{b'}]\!]_{\mathbb{I}}, \textsc{View}_{\mathbb{I}}, \mathrm{R}_{\mathbb{I}}) : \overrightarrow{e} = \overrightarrow{b} - \overrightarrow{b'}\right] = 1,$$

where $\overrightarrow{b} - \overrightarrow{b'}$ is a pair-wise subtraction on $\mathcal{X}$, and $\textsc{View}_{\mathbb{I}}$ and $\mathrm{R}_{\mathbb{I}}$ are the protocol's view and random tapes of corrupted parties, respectively.

(Linear-Combinatorial Protocols)
We consider a class of MPC protocols we call *linear-combinatorial protocols*. Protocols in this class consist of the following two phases.

1. First, in *the offline phase*, each party locally computes his/her inputs of the next online phase from his/her inputs by arbitrary functions.
2. Then, in *the online phase*, each party interacts with other parties freely except that in each round, the party sends only linear combinations of the outputs of the offline phase and received data, where coefficients of the linear combinations are public.

In fact, the class of linear-combinatorial protocols is quite general and contains various primitive protocols frequently used in unconditionally secure MPC schemes such as random number generation, multiplication, reveal, and resharing. (Note that any offline protocols including addition belong to the class of linear-combinatorial protocols since the offline phase allows arbitrary local computations.)

The other significant fact is that any linear-combinatorial protocols on LSSS are tamper-simulatable. Due to space limitations, we give the proof and the formal definition of linear-combinatorial protocols in Appendix B.

Theorem 1. *(informal) Any linear-combinatorial protocols are tamper-simulatable.*

Corollary 1. *GRR-multiplication, DN-multiplication, DN-RNG, and resharing are all tamper-simulatable.*

Next, we claim that *parallel execution* preserves tamper-simulatability. Parallel execution is a concurrent composition of protocols, where each protocol's inputs of honest parties do not depend on the outputs of the other protocols. Intuitively speaking, parallel execution represents that constitutive protocols are executed simultaneously. Note that parallel execution does not include so-called sequential composition.

Lemma 1. *(closure of tamper-simulatability on independent compositions)*
Parallel execution of unconditionally secure tamper-simulatable protocols is tamper-simulatable.

The proof is given in Appendix C.

3.2 Mandatory Building Blocks

The mandatory building blocks are the following seven operations consisting of a uniform RNG, four algebraic operations on $\mathcal{Y}$, and reveal and synchronization (only in an asynchronous setting). They are used in the Randomization Phase and the Proof Phase to guarantee the correctness of the computation.

1. RNG: $[\![r]\!] \leftarrow \mathrm{RAND}_{\mathcal{Y}}$
2. scalar multiplication: $[\![ar]\!]$ for $[\![a]\!] \in [\![\mathcal{X}]\!]$ and $[\![r]\!] \in [\![\mathcal{Y}]\!]$
3. scalar product-sum: $[\![\sum_{i<d} a_i r_i]\!]$ for $d \in \mathbb{N}$, $[\![a_0]\!], \cdots, [\![a_{d-1}]\!] \in [\![\mathcal{X}]\!]$ and $[\![r_0]\!], \cdots, [\![r_{d-1}]\!] \in [\![\mathcal{Y}]\!]$
4. addition/subtraction on $[\![\mathcal{Y}]\!]$
5. multiplication on $[\![\mathcal{Y}]\!]$
6. correct reveal of shared value on $[\![\mathcal{Y}]\!]$
7. synchronization: the protocol SYNC to simulate the synchronous setting

The first five operations require tamper-simulatability. Only reveal in the Proof Phase requires active correctness. Except for SYNC, they are all operations *not* on $\mathcal{X}$, which is the computation space, but on $\mathcal{Y}$. However, there are some pairs of $\mathcal{X}$ and $\mathcal{Y}$ that allow us to efficiently compute the above building blocks. For instance, when $\mathcal{X}$ is a field $\mathcal{F}$ and $\mathcal{Y}$ is its extension $\mathcal{E}(\mathcal{F})^d$ with an arbitrary $d \in \mathbb{N}$, they are constructed by parallel executions of RNG, addition/subtraction, multiplication, and correct reveal on $\mathcal{X}(= \mathcal{F})$, as shown in Appendix D. Therefore, the set of trivial addition/subtraction on Shamir's secret sharing, DN-multiplication, and DN-RNG and correct reveal on LSSS in Appendix D is an example of mandatory building blocks.

Simulating Synchronous Setting in Asynchronous Setting SYNC (Scheme 1) is a protocol to simulate the asynchronous setting and forces honest parties to wait to receive all data before SYNC.

Scheme 1 [Protocol] SYNC
Input: none
Output: none

1: **for each** party P **do**
2: P waits to receive all data before this protocol.
3: If P has received all data, P sends ϕ to all other parties. ϕ is arbitrary fixed data.
4: **for each** party P **do**
5: P waits to receive ϕ from all other parties.
6: If P has received ϕ from all other parties, P proceeds to the next protocol.

3.3 Optional Building Blocks

Optional building blocks are protocols that realize primitive operations in the computation phase. In theory, addition and multiplication are sufficient for computing arbitrary functions. Additionally, our construction allows us to add arbitrary functions that satisfy certain conditions. For function f and its MPC protocol Π_f, f can be used in the Computation Phase in our construction if f and Π_f satisfy the following conditions.

Condition 1 *(conditions of optional building blocks)*

1. *Π_f is tamper-simulatable*
2. *There exists a tamper-simulatable $\mathcal{Y}$-*distribution protocol *$\Pi_{f'}$ of f*

Roughly speaking, $\mathcal{Y}$-distribution represents the existence of a protocol that computes the function f on $\mathcal{E}(\mathcal{F})^d$.

Definition 3. *($\mathcal{Y}$-distribution)*
Let $f : \mathcal{X}^\ell \to \mathcal{X}^m$ be an ℓ-input m-output building block function on a ring $\mathcal{X}$, and let $\mathcal{Y}$ and $\mathcal{Z}$ be an $\mathcal{X}$-algebra and a direct product ring $\mathcal{X} \times \mathcal{Y}$, respectively. We say a function $f' : \mathcal{Z}^\ell \times \mathcal{Y} \to \mathcal{Y}^m$ is a $\mathcal{Y}$-distribution of f if for any $(\vec{a}) \in \mathcal{X}^\ell$ and $r \in \mathcal{Y}$, f' satisfies $f'(\vec{a}, \vec{a}r, r) = f(\vec{a})r$, where $\vec{a}r$ denotes $(a_0 r, \cdots, a_{\ell-1} r)$.

We call a protocol that realizes f' a $\mathcal{Y}$-distribution protocol of f. In contrast with $\mathcal{Y}$-distribution, a protocol that realizes f is called a passive protocol of f.

For example, with a field, $\mathcal{F}$, as $\mathcal{X}$ and its extension $\mathcal{E}(\mathcal{F})^d$ with an arbitrary positive integer d, as $\mathcal{Y}$, linear transformations including addition, multiplication, and resharing have their $\mathcal{E}(\mathcal{F})^d$-distributions, as shown in Appendix E.

4 Proposed Construction

In this section, we explain our construction, which consists of three phases: Randomization, Computation, and Proof. We describe these three phases and the overall construction. Then, we analyze the security, that is, privacy and correctness, of the construction. At the end of this section, we analyze the communication efficiency and the round efficiency of the construction.

4.1 Phase 1: Randomization Phase

The Randomization Phase (Scheme 2) converts shares into *randomized shared pairs* to prevent an adversary from cheating.

In this phase, each input $[\![a_i]\!] \in [\![\mathcal{X}]\!]$ is randomized by $[\![r]\!] \in [\![\mathcal{Y}]\!]$, which is also generated in this phase. The pair $([\![a_i]\!], [\![a_i r]\!])(= \langle\!\langle a_i \rangle\!\rangle)$ is called a randomized shared pair. Randomized shared pairs have some verifiability, which is used in the Proof Phase.

4.2 Phase 2: Computation Phase

The Computation Phase (Scheme 3) computes the target function F redundantly on $\mathcal{X}$ and $\mathcal{Y}$. The target function F is realized by the composition of optional building blocks mentioned in Section 3. After every execution of a building block, the checksum set $C \subseteq \langle\!\langle \mathcal{X}r \rangle\!\rangle$, which will be used in the Proof Phase, is updated.

This phase allows not only multiplication, but also specific efficient protocols as primitive operations if the functions satisfy Condition 1 in the previous section. We have already confirmed that the multiplication, quadratic functions, including product-sum, linear transformations, and resharing, satisfy tamper-simulatability and have $\mathcal{Y}$-distribution protocols. We describe them in detail in Appendix D and Appendix E. Furthermore, reshare-based shuffling [21] is realized by resharing.

Scheme 2 [Phase 1]: Randomization Phase
Parameter: the number of inputs $m \in \mathbb{N}$
Input: $\{[\![a_i]\!]\}_{0 \le i < m} \in [\![\mathcal{X}]\!]^m$
Output: $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m} \in \langle\!\langle \mathcal{X}r \rangle\!\rangle^m$

1: $[\![r]\!] := \mathrm{RAND}_{\mathcal{Y}}$
2: **for each** $i < m$
3: $\quad [\![a_i r]\!] := [\![a_i]\!][\![r]\!]$
4: **for each** $i < m$
5: $\quad \langle\!\langle a_i \rangle\!\rangle := ([\![a_i]\!], [\![a_i r]\!])$
6: Output $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m}$

Scheme 3 [Phase 2]: Computation Phase
Parameter: the number of inputs $m \in \mathbb{N}$, the number of outputs $\mu \in \mathbb{N}$,
and the number of building blocks $\nu \in \mathbb{N}$
Input: $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m} \in \langle\!\langle \mathcal{X}r \rangle\!\rangle^m$,
m-input μ-output function F consists of m_j-input μ_j-output optional building block functions F_j for all $j < \nu$
Output: the computation result $\langle\!\langle F(\{a_i\}_{0 \le i < m}) \rangle\!\rangle \in \langle\!\langle \mathcal{X}r \rangle\!\rangle^\mu$,
the checksum set $C \subseteq \langle\!\langle \mathcal{X}r \rangle\!\rangle$

1: Set C as all input randomized shared pairs $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m}$.
2: **for each** $j < \nu$
3: $\quad$ Let the inputs of the j-th optional building block F_j be $\{\langle\!\langle b_{ij} \rangle\!\rangle\}_{0 \le i < m_j}$.
4: $\quad$ Compute $\{[\![f_i]\!]\}_{0 \le i < \mu} := F_j(\{[\![b_{ij}]\!]\}_{0 \le i < m_j})$ from $\{[\![b_{ij}]\!]\}_{0 \le i < m_j}$ in $\{\langle\!\langle b_{ij} \rangle\!\rangle\}_{0 \le i < m_j}$ using a passive realization Π_{F_j} of F_j.
5: $\quad$ Compute $\{[\![f_i r]\!]\}_{0 \le i < \mu}$ from $\{\langle\!\langle b_{ij} \rangle\!\rangle\}_{0 \le i < m_j}$ using the $\mathcal{Y}$-distribution protocol Π'_j of F_j. (Never compute it from $\{[\![f_i]\!]\}_{0 \le i < \mu}$ or $[\![r]\!]$ by a passive multiplication.)
6: **if** either Π_{F_j} or Π_j is not correct, **then** $C := C \cup \{\langle\!\langle f_i \rangle\!\rangle\}_{0 \le i < \mu}$
7: Output $\{\langle\!\langle f_i \rangle\!\rangle\}_{0 \le i < \mu}$ and C.

4.3 Phase 3: Proof Phase

Finally, the Proof Phase (Scheme 4) guarantees the correctness of all the results of the computation at once by proving that the results on $\mathcal{X}$ and $\mathcal{Y}$ are equal to each other. The concentration of all proofs on one element of $\mathcal{Y}$ makes the proof very efficient and reduces unnecessary revealing of information.

In this phase, shared values $[\![\phi]\!]$ and $[\![\psi]\!]$ are computed from randomized shared pairs in C. If no party cheats with protocols in the Randomization and Computation Phases, $\phi = \psi$ must hold. Otherwise, $\phi \neq \psi$ holds with a high probability, and the adversary's cheating is detected by honest parties.

Note that SYNC is inserted to partially simulate the synchronous setting in the asynchronous setting and is unnecessary in the synchronous setting.

Scheme 4 [Phase 3]: Proof Phase
Parameter: the random shared value $[\![r]\!] \in [\![\mathcal{Y}]\!]$,
the checksum set $C \subseteq \langle\!\langle \mathcal{X}r \rangle\!\rangle$
Input: None
Output: $\top$ if no tampering is detected, $\bot$ otherwise

1: Consider C as $C = \{\langle\!\langle f_0 \rangle\!\rangle, \ldots, \langle\!\langle f_{|C|-1} \rangle\!\rangle\}$
2: **for each** $i < |C|$
3: $\quad [\![\rho_i]\!] := \mathrm{RAND}_{\mathcal{Y}}$
4: $[\![\varphi]\!] := \left(\sum_{i<|C|} [\![f_i]\!][\![\rho_i]\!] \right) [\![r]\!]$
5: $[\![\psi]\!] := \sum_{i<|C|} [\![f_i r]\!][\![\rho_i]\!]$
6: SYNC
7: **if** $\mathrm{REV}_{\mathcal{Y}}([\![\varphi]\!] - [\![\psi]\!]) \neq 0$ **then** Output $\bot$
8: $\quad$ **else** Output $\top$

4.4 Overall Construction

Scheme 5 shows our overall construction. The Randomization, Computation, and Proof Phases are executed simply in series.

Scheme 5 [Overall Construction] Π_F^{act}
Parameters: the number of inputs $m \in \mathbb{N}$ and of outputs $\mu \in \mathbb{N}$
Input: $\{[\![a_i]\!]\}_{0 \le i < m} \in [\![\mathcal{X}]\!]^m$,
m-input μ-output function F consists of optional building block functions
Output: $(\top, [\![F(\{a_i\}_{0 \le i < m})]\!])$ if no tampering is detected,
$\perp$ otherwise

1: run the Randomization Phase for $\{[\![a_i]\!]\}_{0 \le i < m}$ to get $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m}$ and set r as a parameter
2: run the Computation Phase for $\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m}$ to get $F(\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m}) \in \langle\!\langle \mathcal{X}r \rangle\!\rangle^{\mu}$ and set C as a parameter
3: $\{\langle\!\langle f_i \rangle\!\rangle\}_{0 \le i < \mu} := F(\{\langle\!\langle a_i \rangle\!\rangle\}_{0 \le i < m})$
4: run the Proof Phase to obtain $c \in \{\top, \perp\}$
5: **if** $c = \top$ **then** output $(\top, \{[\![f_i]\!]\}_{0 \le i < \mu})$
6: **else** output $\perp$

4.5 Security

Theorem 2. *(correctness)*
Let $\mathcal{F}$ be a finite field whose order is $p \in \mathbb{N}$, and let $\mathcal{E}(\mathcal{F})^d$ be a d-degree extension of $\mathcal{F}$. Then, the output of Π_F^{act} computing a function F is correct in the probability $1 - 2p^{-d} + p^{-2d}$ or higher against an adversary who can control up to t parties. That is, Π_F^{act} has unconditional correctness when considering p^{-d} as a negligible value.

Theorem 3. *(privacy)*
Let $\mathcal{F}$ be a finite field whose order is $p \in \mathbb{N}$, and let $\mathcal{E}(\mathcal{F})^d$ be a d-degree extension of $\mathcal{F}$. Then Π_F^{act} computing a function F is unconditionally private considering p^{-d} as a negligible value against an adversary who can control up to t parties.

The proof is shown in Appendix G.

4.6 Efficiency

We analyzed the performance of our construction with respect to communication efficiency and round efficiency. Our construction is a composition of building blocks; therefore, we can analyze the overall efficiency by enumerating all the building blocks.

1. The Randomization Phase requires one RNG on $\mathcal{Y}$ and m scalar multiplications.
2. The Computation Phase requires the executions of passive protocols and their $\mathcal{Y}$-distribution protocols that depend on the function F.
3. The Proof Phase costs $|C|$ RNG on $\mathcal{Y}$, two scalar sum-products, one multiplication on $\mathcal{Y}$, and requires $\mathrm{REV}_{\mathcal{Y}}$ and one SYNC. Note that the size of the checksum set $|C|$ is the same as the total number of the inputs of F and the outputs elements on $\mathcal{X}$ of F's optional building blocks that are not correct (but are tamper-simulatable).

Communication Efficiency Communication costs that are additional to those in the passive setting are as follows.

1. $|C| + 1$ RNG on $\mathcal{Y}$
2. $\mathcal{Y}$-distribution protocols corresponding to passive building blocks.
3. m scalar multiplications
4. two scalar product-sums
5. one correct $\mathrm{REV}_{\mathcal{Y}}$
6. one SYNC

For example, when $\mathcal{X}$ is a field $\mathcal{F}$, $\mathcal{Y}$ is an extension $\mathcal{E}(\mathcal{F})^d$ of $\mathcal{F}$, pseudorandom numbers are allowed, and optional building blocks that are not correct are multiplication and shuffling, the communication cost of our construction is

$$(d+1)(N_{\mathrm{shf}}C_{\mathrm{shf}} + N_{\mathrm{mul}}C_{\mathrm{mul}}) + (m+2)C_{\mathrm{mul}} + C_{\mathrm{REV}_{\mathcal{Y}}} + C_{\mathrm{SYNC}},$$

where m is the number of inputs of F, C_{shf}, C_{mul}, $C_{\mathrm{REV}_{\mathcal{Y}}}$, and C_{SYNC} are the communication costs of passive shuffling, passive multiplication, $\mathrm{REV}_{\mathcal{Y}}$, and SYNC, respectively, and N_{shf} and N_{mul} are the numbers of shufflings and multiplications in F, respectively. Recall that the communication cost of the product-sum is the same as multiplication and that scalar multiplication and multiplication on $\mathcal{E}(\mathcal{F})^d$ are equivalent to d times the multiplications on $\mathcal{F}$. Furthermore, when F is a circuit that consists of addition and multiplication, the cost is as follows:

$$(d+1)N_{\mathrm{mul}}C_{\mathrm{mul}} + (m+2)C_{\mathrm{mul}} + C_{\mathrm{REV}_{\mathcal{Y}}} + C_{\mathrm{SYNC}}$$

Although $C_{\mathrm{REV}_{\mathcal{Y}}}$ and C_{SYNC} are $O(n^2)$, they are executed only once, in contrast to $(d+1)N_{\mathrm{mul}} + (m+2)$ times of multiplications; thus, the example is a $O((c_M n + n^2)\kappa)$ bits (per multiplication) scheme, where c_M denotes the size of the circuit (i.e., $c_M = N_{\mathrm{mul}}$) and κ denotes the security parameter (i.e., $\kappa = |\mathcal{F}|d$ where $|\mathcal{F}|$ is the bit length of $\mathcal{F}$).

Table 2. Performance of Parallel Multiplications

Number of multiplications	100,000	1,000,000	10,000,000	
setting	processing time [ms]			max. throughput [M/s]
passive	19.7	254.7	1,622.3	6.164
active	100.0	559.3	4,003.3	2.498

Table 3. Performance of Shuffling

data size	100,000	1,000,000	10,000,000	
setting	processing time [ms]			max. throughput [M/s]
passive	48.3	316.0	2,785.3	3.590
active	127.7	802.3	7,134.7	1.402

Round Efficiency Our construction is not only efficient with respect to communication efficiency but also efficient with respect to round efficiency. Passive protocols and $\mathcal{Y}$-distribution protocols in the Computation Phase can be executed in parallel, and the Randomization Phase and Proof Phase include only constant protocols.

For example, in the same condition as the example in Section 4.6, the Randomization Phase costs two rounds, the Computation Phase costs as much as the passive execution of F, and the Proof Phase costs seven rounds. The computation of φ in the Proof Phase can be started two rounds earlier since $[\![\varphi]\!]$ is independent of randomization in the Randomization Phase. Thus,

$$R_{\text{passive}} + 7$$

is the overall round cost of the example, where R_{passive} is the round cost of the passive execution of F, independent of the size of the circuit of F. If we chooses one round multiplication, such as GRR-multiplication, the cost becomes $R_{\text{passive}} + 5$.

5 Experimental Results

We implemented our construction with some concrete building blocks. We show the performance of the implementation in this section.

The setting is as follows.

- $t = 1$ (i.e., $k = 2$) and $n = 3$.
- The security of pseudorandom numbers is allowed.

Although $n = 3$ is the smallest n and is disadvantageous to show an order improvement, it is sufficient to confirm the absolute efficiency, and $n = 3$ is the most practical setting.

The environment is as follows. Each party is realized as a notebook PC connected to other PCs by a network through a switching hub, and all PCs are homogeneous. The specifications of each PC are as follows.

- CPU: Intel Core i7 2640M (2.8 GHz, 2-core)
- RAM: 8 GB
- Network I/F: 1000BASE-T port x 1

Multiplication Table 2 summarizes the performance when F consists of parallel multiplications. $\mathcal{X}$ and $\mathcal{Y}$ are both $\mathbb{Z}_p$, where p is a Mersenne prime $2^{61} - 1$. Multiplication is $O(n^2)$ GRR-multiplication. When $n = 3$, the multiplication is more efficient than $O(n)$ DN-multiplication.

Shuffling Table 3 summarizes the performance when F is shuffling and the condition is the same as multiplication. The passive shuffling protocol as the building block is the reshare-based protocol [21] by Laur et al.

Optimized Configuration for Logical Circuits Table 4 summarizes the performance when F consists of logical gates, more precisely, when F is a 32-bit comparison, $\mathcal{X}$ is $\mathbb{Z}_2$, and $\mathcal{Y}$ is an extension field $GF(2^8)$. On $\mathbb{Z}_2$, we can apply the techniques of XOR-free circuits [20]. Shares are shared using a replicated secret sharing scheme [9]. Although the scheme is not generally efficient, it is sufficiently efficient when $n = 3$. Replicated secret sharing supports $\mathbb{Z}_2$, in contrast to Shamir's scheme and other general schemes [7, 10] [1] that are as efficient as Shamir's secret sharing scheme. Multiplication is shown in Appendix A, and its communication and round costs are the same as GRR-multiplication with $k = 2$ and $n = 3$.

When $\mathcal{X} = \mathbb{Z}_2$ and $\mathcal{Y} = GF(2^8)$, passive execution should be about nine times faster because $d = 8$. However, the actual performance is almost the same as that of active execution. This result requires further investigation.

[1] Although the scheme by Cramer et al. [10] supports an arbitrary ring, the scheme requires a matrix that satisfies the specific condition on the ring and cannot be constructed on $\mathbb{Z}_2$.

Table 4. Performance of Comparison Circuit

data size	100,000	1,000,000	10,000,000	
setting	processing time [ms]			max. throughput [M/s]
passive	183.3	867.3	7,898.3	1.266
active	171.7	937.0	7,682.0	1.302

Comparison with Current High-Performance Passive Implementation For multiplication, shuffling, and comparison, Sharemind is the fastest implementation, and throughputs are about 0.5, 0.4, and about 0.05 M/s on three-party server machine environments [4, 21]. The throughputs on our implementation were about 6.2, 1.4, and 1.3 M/s on a notebook PC environment. Thus, our active multiplication, shuffling, and comparison were faster than throughputs of passive implementations. Therefore, we claim that our non-robust active construction is sufficiently practical with respect to efficiency.

6 Conclusion

We proposed constructing a non-robust, actively, and unconditionally secure MPC scheme from passively secure schemes while maintaining efficiency.

Our construction is secure in the $t < n/2$ setting and can use high-level protocols as optional building blocks if the protocols satisfy tamper-simulatability and have $\mathcal{Y}$-distributions. In addition, the communication cost of our construction is comparable to the known smallest cost in the passive case. We implemented our construction and confirmed its efficiency. As a result, our construction is only several times slower than passively secure MPC schemes in theory and is faster than the current fastest passively secure implementation.

References

1. Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In Halevi and Rabin [14], pages 305–328.
2. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure mpc with linear communication complexity. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2008.
3. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680. Springer, 2012.
4. D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.
5. P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In R. Dingledine and P. Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.
6. M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240. USENIX Association, 2010.
7. R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In Kilian [19], pages 119–136.
8. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In J. Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1999.
9. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005.
10. R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer, 2003.
11. I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi and Rabin [14], pages 285–304.
12. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.
13. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In B. A. Coan and Y. Afek, editors, *PODC*, pages 101–111. ACM, 1998.
14. S. Halevi and T. Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006.
15. K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In T. Kwon, M.-K. Lee, and D. Kwon, editors, *ICISC*, volume 7839 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2012.
16. M. Hirt and U. M. Maurer. Robustness for free in unconditional multi-party computation. In Kilian [19], pages 101–118.
17. M. Hirt and P. Raykov. On the complexity of broadcast setup. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 552–563. Springer, 2013.
18. L. Kamm, D. Bogdanov, S. Laur, and J. Vilo. A new way to protect privacy in large-scale genome-wide association studies. In *Bioinformatics*, 2013.

19. J. Kilian, editor. *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*. Springer, 2001.
20. V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In J. A. Garay, A. Miyaji, and A. Otsuka, editors, *CANS*, volume 5888 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2009.
21. S. Laur, J. Willemson, and B. Zhang. Round-efficient oblivious database manipulation. In X. Lai, J. Zhou, and H. Li, editors, *ISC*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2011.

A Passive Secure Schemes

Here, we describe the tamper-simulatable protocols discussed in the paper. Scheme 6 is GRR-multiplication, Scheme 7 is DN-multiplication (Scheme 8 is a sub-protocol of DN-multiplication), Scheme 9 is RNG in [12], Scheme 10 is passive resharing, and Scheme 11 is a multiplication on $(2,3)$-replicated secret sharing. These protocols are all the linear-combinatorial protocols discussed in Section 3 (Proofs are omitted.). Thus, they are all tamper-simulatable by Theorem 1.

Scheme 6 [Protocol] GRR-multiplication
Parameter: the threshold k, the number of parties n,
the points $x_0, \ldots, x_{n-1} \in \mathcal{F}$ assigned to parties
Parties: $P_0, \ldots, P_{n-1}$
Input: $[\![a]\!], [\![b]\!] \in [\![\mathcal{F}]\!]$, that is, $a_i, b_i \in \mathcal{F}$ for each party P_i
Output: $[\![ab]\!]$

1: Each party P_i where $i < 2k-1$ computes $(2k-1, n)$ share $\langle c \rangle_i := [\![a]\!]_i [\![b]\!]_i$ and shares it so that parties obtain $[\![\langle c \rangle_i]\!]$. ($\langle \cdot \rangle$ denotes $(2k-1, n)$ shared value.)
2: Parties compute Lagrange interpolation $\sum_{i<2k-1} \alpha_i [\![\langle c \rangle_i]\!]$ with proper coefficients $\alpha_0, \cdots, \alpha_{2k-2}$ to reconstruct $(2k-1, 2k-1)$-share from $x_0, \ldots, x_{n-1} \in \mathcal{F}$.

Scheme 7 [Protocol] DN-multiplication
Parameter: the threshold k, the number of parties n,
the points assigned to parties $x_0, \ldots, x_{n-1} \in \mathcal{F}$,
Parties: $P_0, \ldots, P_{n-1}$
Input: $[\![a]\!], [\![b]\!] \in [\![\mathcal{F}]\!]$, that is, $a_i, b_i \in \mathcal{F}$ for each party P_i
Output: $[\![ab]\!]$

1: Parties execute Double Random (Scheme 8) and obtain a (k, n) shared value $[\![r]\!]$ and a $(2k-1, n)$ shared value $\langle r \rangle$, both plaintexts are $r \in \mathcal{F}$. ($\langle \cdot \rangle$ denotes $(2k-1, n)$ shared value.)
2: Each party P_i where $i < 2k-1$ computes $\langle c \rangle_i = [\![a]\!]_i [\![b]\!]_i + \langle r \rangle_i$ and sends it to P_0.
3: P_0 reconstructs the plaintext c from shares $\langle c \rangle_0, \ldots, \langle c \rangle_{2k-2}$ which were received in the previous step.
4: P_0 distributes c to all other parties.
5: Each party P_i computes $c - [\![r]\!]_i$ and outputs it.

Scheme 8 [Protocol] Double Random (passive)
Parameter: the threshold k, the number of parties n,
the points assigned to parties $x_0, \ldots, x_{n-1} \in \mathcal{F}$ and Van der Monde matrix M
Parties: $P_0, \ldots, P_{n-1}$
Input: none
Output: (k, n) random shared values $[\![s_0]\!], \ldots, [\![s_{n-k}]\!]$ and $(2k-1, n)$ random shared values $\langle s_0 \rangle, \ldots, \langle s_{n-k} \rangle$

1: Each party P_i does as follows.
2: generates uniformly random value r_i in $\mathcal{F}$.
3: P_i shares r_i in two manners, (k, n) and $(2k-1)$ Shamir's secret sharing schemes, and each party P_j obtains shares $r'_{k,i,j}$ and $r'_{2k,i,j}$, respectively.
4: P_i obtains $(s_{k,i,0}, \ldots, s_{k,i,n-k}) = M(r'_{k,i,0}, \ldots, r'_{k,i,n-k})$ and $(s_{2k,0}, \ldots, s_{2k,n-k}) = M(r'_{2k,0}, \ldots, r'_{2k,n-k})$, and outputs them.

Scheme 9 [Protocol] Random Number Generation (passive)
Parameter: the threshold k, the number of parties n,
the points assigned to parties $x_0, \ldots, x_{n-1} \in \mathcal{F}$ and Van der Monde matrix M
Parties: $P_0, \ldots, P_{n-1}$
Input: none
Output: random shared values $[\![s_0]\!], \ldots, [\![s_{n-k}]\!]$

1: Each party P_i does as follows.
2: generates uniformly random value r_i in $\mathcal{F}$.
3: P_i shares r_i and each party P_j obtains shares $r'_{k,i,j}$.
4: P_i obtains $(s_{k,i,0}, \ldots, s_{k,i,n-k}) = M(r'_{k,i,0}, \ldots, r'_{k,i,n-k})$ and outputs them.

Scheme 10 [Protocol] passive resharing
Parameter: the threshold k, the number of parties n
Input: $[\![a]\!] \in [\![\mathcal{F}]\!]$ for $P_0, \ldots, P_{k-1}$
Output: $[\![a]\!]$ for all parties (the randomness of shares are different from the input)

1: Parties generate a random shared value $[\![r]\!]$. Let P_i's share be r_i.
2: Parties compute $[\![a']\!] := [\![a]\!] + [\![r]\!]$.
3: Each party P_i sends $[\![a']\!]_i$ to P_0.
4: P_0 reconstructs the plaintext a' from shares $[\![a']\!]_0, \ldots, [\![a']\!]_{k-1}$ which was received in the previous step.
5: P_0 distributes a' to all other parties.
6: Each party P_i computes $a' - [\![r]\!]_i$ and outputs it.

Scheme 11 [Protocol] passive multiplication on $(2, 3)$-replicated sharing
Parties: X, Y, Z
Input: $[\![a]\!], [\![b]\!] \in [\![\mathcal{X}]\!]$, i.e., (a_0, a_1) and (b_0, b_1) for X, (a_1, a_2) and (b_1, b_2) for Y, and (a_2, a_0) and (b_2, b_0) for Z where $a = a_0 + a_1 + a_2$ and $b = b_0 + b_1 + b_2$
Output: $[\![ab]\!]$

1: X, Y, and Z generate r_{ZX}, r_{XY}, and r_{YZ}, respectively.
2: X sends r_{ZX} to Z, Y sends r_{XY} to X, and Z sends r_{YZ} to Y.
3: X sends $c_{XY} := a_0b_1 + a_1b_0 - r_{ZX}$ to Y, Y sends $c_{YZ} := a_1b_2 + a_1b_2 - r_{XY}$ to Z, and Z sends $c_{ZX} := a_2b_0 + a_2b_0 - r_{YZ}$ to X.
4: Let c_0, c_1, and c_2 be $c_0 := a_0b_0 + c_{ZX} + r_{ZX}$, $c_1 := a_1b_1 + c_{XY} + r_{XY}$, and $c_2 := a_2b_2 + c_{YZ} + r_{YZ}$, respectively.
5: X outputs (c_0, c_1), Y outputs (c_1, c_2), and Z outputs (c_2, c_0).

B Linear-Combinatorial Protocols

We prove Theorem 1, which states all linear-combinatorial protocols are tamper-simulatable.

First, we show an intuition. For simplicity, the MPC protocol has three rounds, and the adversary sends only one element in each round. Let (c_1, c_2, c_3) and (c'_1, c'_2, c'_3) be the legitimate values and possibly tampered values, respectively, sent by the adversary in the first, second, and third round.

The simulator can compute (c'_1, c'_2, c'_3) since they depend on only the adversary's knowledge. The simulator can also compute c_1 and c_2 since they depend on only the inputs, auxiliary inputs, and values sent by the honest parties in the first round. The last c_3 is a bit problematic since the values sent by the honest parties in the second round depend on c'_1. However, the simulator can compute $(c'_1 - c_1)$, and the honest parties only compute the linear combination whose coefficient γ is public. Therefore, the simulator can compute $\gamma(c'_1 - c_1)$ then compute $(c'_3 - c_3)$. Consequently, the simulator can compute the tamper-difference.

Next, we formally define linear-combinatorial protocols.

Definition 4. *(linear-combinatorial protocols)*
Let $\mathcal{R}$ be a ring, $m, \mu, \nu \in \mathbb{N}$ be the numbers of inputs, outputs, and rounds, respectively, $\{[\![a_\iota]\!]\}_{\iota<m}$ be the inputs, and $\{[\![b_\iota]\!]\}_{\iota<\mu}$ be the outputs.

A Linear-combinatorial protocol consists of two phases, offline and online. In the offline phase, each party P_i locally computes online inputs $\{z_{i,\iota}\}_{\iota<\upsilon_i} \in \mathcal{R}^{\upsilon_i}$ by an arbitrary function $f_i(\{[\![a_\iota]\!]_i\}_{\iota<m}) \mapsto \{z_{i,\iota}\}_{\iota<\upsilon_i}$, where $\upsilon_i \in \mathbb{N}$ is the number of each party's online inputs.

In each ℓ-th round in the online phase, each party P_i sends $\eta_{\ell,i,j}$ data $\{c_{\ell,i,j,u}\}_{u<\eta_{\ell,i,j}} \in \mathcal{R}^{\eta_{\ell,i,j}}$ to each other party P_j where $\eta_{\ell,i,j} \in \mathbb{N}$. The sent data are linear combinations of offline inputs $\{z_{i,\iota}\}_{\iota<\upsilon_i}$ and all data received $\{c_{\ell',j',i,u'}\}_{\ell'<\ell,j'\neq i,u'<\eta_{\ell',j',i}}$ by the $(\ell-1)$-th round. Thus, the linear combinations are represented as $c_{\ell,i,j,u} = \sum_{\iota<\upsilon_i} \gamma_{\ell,i,j,\iota} z_\iota + \sum_{\substack{\ell'<\ell \\ j'\neq i \\ u'<\eta_{\ell',j',i}}} \gamma'_{\ell,i,j,u,\ell',j',u'} c_{\ell',j',i,u'}$, where each $\gamma_{\ell,i,j,\iota}$ and $\gamma'_{\ell,i,j,u,\ell',j',u'}$ are public coefficients. Furthermore, the outputs $\{[\![b_\iota]\!]_i\}_{\iota<\mu}$ of party P_i are linear combinations of $\{z_{i,\iota}\}_{\iota<\upsilon_i}$ and $\{c_{\ell,j',i,u'}\}_{\ell<\nu,j'\neq i,u'<\eta_{\ell',j',i}}$.

In the online phase, computations are restricted to linear combinations. However, the offline phase is allowed to perform arbitrary functions including multiplication in GRR-multiplication and DN-multiplication, and generation of all random numbers used in the protocol. In addition, the linear combination is sufficient to realize share and reconstruction schemes of Shamir's secret sharing and replicated secret sharing.

Finally, we prove the tamper-simulatability of linear-combinatorial protocols.

Theorem 4. *(formal version of Theorem 1) Any linear-combinatorial protocol is tamper-simulatable when $[\![\cdot]\!]$ represents a shared value of an LSSS.*

Proof. (Theorem 4)
Let each $c'_{\ell,i,j,u}$ denote the actual sent data in the active setting, and let $c_{\ell,i,j,u}$ denote imaginary correct data sent in the passive setting.

First we prove the following lemma by using the induction method on ℓ.

Lemma 2. *The simulator can compute interim tamper-differences $\delta_{\ell,i,j,u} = c'_{\ell,i,j,u} - c_{\ell,i,j,u}$ for any round $\ell < \nu$, any parties P_i and P_j, and any index of data among the data that P_i sends to P_j in the protocol.*

Proof. (Lemma 2)
(i) When $\ell = 0$, no data have been sent yet, and all interim tamper-differences are 0.
(ii) When $\ell > 0$, assuming that all differences $\delta_{\ell',i,j,u} = c'_{\ell',i,j,u} - c_{\ell',i,j,u}$ such that $\ell' < \ell$ can be computed by the simulator for all $i,\ j < n,\ u < \eta_{\ell',i,j}$, we prove that $c'_{\ell,i,j,u} - c_{\ell,i,j,u}$ can be computed by the simulator for all $i,\ j < n,\ u < \eta_{\ell,i,j}$.

When P_i is honest, $\delta_{\ell,i,j,u}$ is computed as follows.

$$\begin{aligned}\delta_{\ell,i,j,u} = c'_{\ell,i,j,u} - c_{\ell,i,j,u} &= \sum_{\iota<\upsilon_i} \gamma_{\ell,i,j,u,\iota} z_\iota + \sum_{\substack{\ell'<l \\ j'\neq i \\ u'<\eta_{\ell',j',i}}} \gamma'_{\ell,i,j,u,l',j',u'} c'_{\ell',j',i,u'} - \sum_{\iota<\upsilon_i} \gamma_{\ell,i,j,u,\iota} z_\iota - \sum_{\substack{\ell'<\ell \\ j'\neq i \\ u'<\eta_{\ell',j',i}}} \gamma'_{\ell,i,j,u,\ell',j',u'} c_{\ell',j',i,u'} \\ &= \sum_{\substack{\ell'<\ell \\ j'\neq i \\ u'<\eta_{\ell',j',i}}} \gamma'_{\ell',j',i,u'} \delta_{\ell',j',i,u'}\end{aligned}$$

Note that the adversary's knowledge of such differences does not imply his/her knowledge of plaintexts. Similarly, when P_i is a corrupted party, $\delta_{\ell,i,j,u}$ is computed as follows.

$$\delta_{\ell,i,j,u} = c'_{\ell,i,j,u} - c_{\ell,i,j,u} = c'_{\ell,i,j,u} - \sum_{\iota<\upsilon_i} \gamma_{\ell,i,j,u,\iota} z_\iota + \sum_{\substack{\ell'<\ell \\ j'\neq i \\ u'<\eta_{\ell',j',i}}} \gamma'_{\ell,i,j,u,\ell',j',u'} (-c'_{\ell',j',i,u'} + \delta_{\ell',j',i,u'})$$

Note that both z_ι and $c'_{\ell',j',i,u'}$ are known by the adversary/simulator since P_i is corrupted.

By the induction hypothesis, Lemma 2 has been proven.

□ Lemma 2

Similarly, the outputs $\{[\![b_\iota]\!]_i\}_{\iota<\mu}$ of party P_i are linear combinations of $\{z_{i,\iota}\}_{\iota<\upsilon_i}$ and $\{c_{\ell,j',i,u'}\}_{\ell<\nu,j'\neq i,u'<\eta_{\ell',j',i}}$, and a reconstruction scheme of LSSS is also a linear combination of shares; therefore, the adversary can compute the differences in the outputs.

□ Theorem 4

C Proof of Lemma 1

Lemma 1. *(closure of tamper-simulatability on independent compositions)*
Parallel execution of unconditionally secure tamper-simulatable protocols is tamper-simulatable.

Proof. Let $\Pi_0, \ldots, \Pi_{\ell-1}$ be constitutive protocols and Π^* be the entire protocol that is a parallel execution of $\Pi_0, \ldots, \Pi_{\ell-1}$, and $aux_0, \ldots, aux_{\ell-1}$ be auxiliary inputs on protocols $\Pi_0, \ldots, \Pi_{\ell-1}$, respectively.

We can construct a simulator $\mathcal{S}^*$ for Π^* for an adversary with an auxiliary input aux^* as follows. First, we consider a tamper in Π_0. The difference between the solo execution of Π_0 and parallel executions for an adversary is that in parallel executions, the adversary can tamper with Π_0 with the help of the views of $\Pi_1, \ldots, \Pi_{\ell-1}$ executions. However, tamper-simulatability guarantees that there exists a simulator for any auxiliary input aux_0 that can contain the views of other executions. Therefore, there exists a simulator $\mathcal{S}_0$ that computes a tamper-difference even in parallel executions. For Π_1 …, $\Pi_{\ell-1}$ there also exists a simulator $\mathcal{S}_1, \ldots, \mathcal{S}_{\ell-1}$ for the same reason.

Consequently, the outputs of $\mathcal{S}^*$ are the sum of the tamper-differences computed by $\mathcal{S}_0, \ldots, \mathcal{S}_{\ell-1}$.

□ (Lemma 1)

D Example of Mandatory Building Blocks

The RNG, scalar multiplication/product-sum, addition/subtraction, multiplication, and reveal are all realized on $[\![\mathcal{F}]\!]^d$ by operations on $[\![\mathcal{F}]\!]$. Note that $\mathcal{F}^d$ is trivially homeomorphic to $\mathcal{E}(\mathcal{F})^d$ as a group.

1. RNG : $[\![r]\!] \leftarrow ([\![r_0]\!], \cdots, [\![r_{d-1}]\!])$, where each $[\![r_i]\!]$ is generated by an RNG on $[\![\mathcal{F}]\!]$, and is also random on $[\![\mathcal{F}]\!]^d$ as $[\![\mathcal{E}(\mathcal{F})^d]\!]$.
2. scalar multiplication : $[\![ar]\!] \leftarrow ([\![ar_0]\!], \cdots, [\![ar_{d-1}]\!])$, where $[\![a]\!] \in [\![\mathcal{F}]\!]$ and $[\![r]\!] \in [\![\mathcal{F}]\!]^d$.
3. scalar product-sum : $[\![\sum_{i<m} a_i r_i]\!] \leftarrow ([\![\sum_{i<m} a_i (r_i)_0]\!], \cdots, [\![\sum_{i<m} a_i (r_i)_{d-1}]\!])$, where $m \in \mathbb{N}$ and for each $i < m$, $[\![a_i]\!] \in [\![\mathcal{F}]\!]$ and $[\![r_i]\!] \in [\![\mathcal{F}]\!]^d$.
4. addition/subtraction : $[\![r + s]\!] \leftarrow ([\![r_0]\!] + [\![s_0]\!], \cdots, [\![r_{d-1}]\!] + [\![s_{d-1}]\!])$, where $[\![r]\!], [\![s]\!] \in [\![\mathcal{F}]\!]^d$.
5. multiplication : $[\![rs]\!] \leftarrow ([\![\sum_{i,j<d} \alpha_{i,j,0} r_i s_j]\!], \cdots, [\![\sum_{i,j<d} \alpha_{i,j,d-1} r_i s_j]\!])$ with a sequence of coefficient matrices $\alpha_0, \cdots, \alpha_{d-1} \in \mathcal{F}^{d^2}$ determined by an irreducible polynomial of $\mathcal{E}(\mathcal{F})^d$. For example, when $d = 2$ and the irreducible polynomial is $X^2 + X + 1$, $\alpha_0 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, $\alpha_1 = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$.
6. (correct) reveal: shown in Scheme 12.

If addition/subtraction and multiplication are both tamper-simulatable, the above mandatory building blocks are also tamper-simulatable since they will be (except for reveal) parallel executions of addition/subtraction and multiplication. Reveal in Scheme 12 is correct by itself.

In LSSS, arbitrary quadratic functions, including product-sum functions, are computed with the same communication and round cost as multiplication in the passive setting. Thus, the communication and round cost of scalar multiplication/product-sum and multiplication on $\mathcal{E}(\mathcal{F})^d$ are the same as d parallel multiplications on $\mathcal{F}$ or 1 multiplication on native $\mathcal{E}(\mathcal{F})^d$.

E Examples of Optional Building Blocks

Linear transformations including addition/subtraction, and quadratic functions including multiplication are already shown as mandatory building blocks in Appendix D; They can be used as optional building blocks.

Resharing used in reshare-based shuffling [21] can also be used as an optional building block. In this shuffling, input shared values $a_0, \ldots, a_{m-1}$ are randomly permuted by k parties with random permutation data π distributed (as plaintexts) among these k parties. Since the offline phase in linear-combinatorial protocols allows arbitrary functions, the permutation by π of shares is also allowed. Thus, resharing in the shuffling protocol is tamper-simulatable if π is distributed before the protocol starts. (In fact, this resharing is tamper-simulatable even if π is dynamically distributed.) Its $\mathcal{E}(\mathcal{F})^d$-distribution is simply d parallel executions of resharing on $\mathcal{F}$.

Scheme 12 [Protocol] Correct Reveal on LSSS: $\mathrm{REV}_{\mathcal{Y}}([\![a]\!])$
Parameter: the threshold k, the number of parties n,
the points assigned to parties $x_0, \ldots, x_{n-1} \in \mathcal{F}$
Parties: $P_0, \ldots, P_{n-1}$
Input: $[\![a]\!] \in [\![\mathcal{F}]\!]$, that is, $a_i \in \mathcal{F}$ for each party P_i
Output: a for each party and $\perp$ if $[\![a]\!]$ is inconsistent or tampered with

for each $i < n$, P_i **do**
  for each $j < n$ **do**
    send $a_{ji} := [\![a]\!]_i$ to P_j
for each $j < n$, P_j **do**
  $c_j :=$ true
  for each $k \leq i < n$
    compute i-th share a'_{ji} from $a_{j0}, \ldots, a_{j(k-1)}$. Note that the secret sharing scheme defined in Section 2 guarantees that k shares determine all other shares uniquely.
    if $a'_{ji} \neq a_{ji}$ **then** $c_j :=$ false
  for each $i < n$
    send $c_{ji} := c_j$ to P_i
for each $i < n$, P_i **do**
  if $\bigwedge_{j<n} c_{ji} =$ false **then** output $\perp$
  else output the plaintext reconstructed from $a_{j0}, \ldots, a_{j(k-1)}$.

F Proof of Theorems 2 and 3

Theorem 2**.** *(correctness)*
Let $\mathcal{F}$ be a finite field whose order is $p \in \mathbb{N}$ and let $\mathcal{E}(\mathcal{F})^d$ be a d-degree extension of $\mathcal{F}$. Then, the output of Π_F^{act} computing a function F is correct in the probability $1 - 2p^{-d} + p^{-2d}$ or higher against an adversary who can control up to t parties. That is, Π_F^{act} has unconditional correctness when considering p^{-d} as a negligible value.
Theorem 3**.** *(privacy)*
Let $\mathcal{F}$ be a finite field whose order is $p \in \mathbb{N}$ and let $\mathcal{E}(\mathcal{F})^d$ be a d-degree extension of $\mathcal{F}$. Then, Π_F^{act} computing a function F is unconditionally private when considering p^{-d} as a negligible value against an adversary who can control up to t parties.

The above correctness and privacy of protocols whose inputs and outputs are both shared values are defined as follows.

Definition 5. *We say that a protocol Π_F with consistent inputs $[\![\overrightarrow{a}]\!]$, outputs $[\![F(\overrightarrow{b})]\!]$, and a functionality F is unconditionally correct if and only if for any set of k parties, the plaintexts of all outputs reconstructed from the k parties' shares are $F(\overrightarrow{a})$ except for a negligible probability.*

Definition 6. *We say that a protocol Π_F with consistent inputs $[\![\overrightarrow{a}]\!]$, outputs $[\![F(\overrightarrow{b})]\!]$, and a functionality F is unconditionally private if and only if there exists a fixed distribution f of an adversary's view and the adversary's actual view $\mathrm{VIEW}_{\mathbb{I}}$ in an execution of a real protocol is statistically indistinguishable from a random variable whose distribution is f. Two random variables A, B on a probability space Ω are said to be statistically indistinguishable if and only if $\sum_{x \in \Omega} |\Pr(A = x) - \Pr(B = x)|$ is negligible.*

The two theorems are related; thus, we prove them together.

Proof. (Theorem 2 and Theorem 3)

First, Π_F^{act} may use one tamper-simulatable building block multiple times as different instances; thus, we distinguish those instances and call them protocol instances. We give the indices to protocol instances in the Randomization Phase and Computation Phase according to the following two rules.
(i) For all protocol instances Π_i, Π_j and their indices $i, j \in \mathbb{N}$, i and j satisfy $i < j$ if any output of Π_i is one of the inputs of Π_j.
(ii) $\Pi_i \neq \Pi_j$ implies $i \neq j$.

Next, we prove privacy before $\mathrm{REV}_{\mathcal{Y}}$, which is necessary for both correctness and overall privacy.

Lemma 3. *(privacy before $\mathrm{REV}_{\mathcal{Y}}$ in the Proof Phase)*
Π_F^{act} before $\mathrm{REV}_{\mathcal{Y}}$ is unconditionally private against the active adversary in Theorem 2 *and Theorem* 3*. Furthermore, r and each ρ_i for all $i < |C|$ are also private before $\mathrm{REV}_{\mathcal{Y}}$ in the Proof Phase.*

Proof. (Lemma 3)
In the synchronous setting, Lemma 3 holds because all the building blocks are unconditionally private. In the asynchronous setting, we also need SYNC. In such a setting, the adversary has a strategy to wait before receiving the data of $\mathrm{REV}_{\mathcal{Y}}$ from honest parties to keep his/her data unsent. The received data may provide some knowledge to the adversary before all the building blocks (except $\mathrm{REV}_{\mathcal{Y}}$) are finished. However, due to the existence of SYNC, each honest party waits for all expected data from all other parties before SYNC. Thus, the adversary cannot obtain any information before $\mathrm{REV}_{\mathcal{Y}}$ starts.

□ (Lemma 3)

An important fact derived from this lemma is that r is not known to the adversary. This means that the tamper-difference is independent of r since the adversary's ability to tamper is at most to add a value x that he/she knows due to tamper-simulatability.

In the Proof Phase, $\varphi - \psi$ is computed. This is the most important value since (i) it is the only reveal of a value that possibly depends on the secrets, and (ii) honest parties judge the correctness of the overall outputs of Π_F^{act} by it.

These φ and ψ values may be tampered with and become tampered values φ' and ψ'. Let each randomized shared pair of outputs of a protocol instance Π_i for any $i < |C|$ be $([\![f_i + x_i]\!], [\![f_i r + y_i]\!])$, where $f_i, x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, $[\![\varphi']\!] - [\![\psi']\!]$ is represented as follows for some $\chi \in \mathcal{F}$ and $\upsilon \in \mathcal{E}(\mathcal{F})^d$. Shares might be inconsistent, i.e., the plaintexts are not uniquely determined. We fix an arbitrary set of k honest parties and define the plaintext of a shared value as the plaintext reconstructed from these k parties' shares.

$$
\begin{aligned}
&[\![\varphi']\!] - [\![\psi']\!] \\
&= \left(\sum_{i<|C|} [\![f_i + x_i]\!][\![\rho_i]\!] + [\![\chi]\!] \right) [\![r]\!] - \sum_{i<|C|} [\![f_i r + y_i]\!][\![\rho_i]\!] + [\![\upsilon]\!] \\
&= [\![\sum_{i<|C|} (x_i r - y_i)\rho_i + (\chi r - \upsilon)]\!] \qquad (1)
\end{aligned}
$$

(i) When the adversary actively attacks Π_F^{act} before the Proof Phase, there must be the first protocol instance Π_ι in which any corrupted party violates the protocol, i.e., Π_i is correctly executed for any $i < \iota$. Since the Randomization Phase and Computation Phase have only correct and tamper-simulatable protocol instances, such a first protocol instance is tamper-simulatable. (Note that for $i > \iota$, x_i and y_i possibly depend on r or secrets.)

Thus, by the definition of tamper-simulatability and privacy before $\mathrm{REV}_{\mathcal{Y}}$ in the Proof Phase, x_ι and y_ι turn out to be values that the adversary/simulator can compute before Π_F^{act} starts. By transforming the plaintext of Formula (1) as follows,

$$
\sum_{i<|C|} (x_i r - y_i)\rho_i + (\chi r - \upsilon) = (x_\iota r - y_\iota)\rho_\iota + \sum_{i \neq \iota} (x_i r - y_i)\rho_i + (\chi r - \upsilon) \qquad (2)
$$

we can discuss its distribution.

When $r \neq y_\iota / x_\iota$ holds, $(x_\iota r - y_\iota)\rho_\iota$ is uniformly random in $\mathcal{Y}$ since ρ_ι is uniformly random in the field $\mathcal{Y}$ and independent of r and all ρ_i such that $i \neq \iota$. On the other hand, when $r = y_\iota / x_\iota$, we cannot ensure the distribution is "good" in regard to security; however, $r = y_\iota / x_\iota$ only occurs in the negligible probability $1/p^d$ since $\mathcal{Y}$ was assumed to be a field.

Therefore, $\varphi - \psi$, which is the only possible reveal of the secrets, is indistinguishable from a uniformly random value in $\mathcal{Y}$ whose order is p^d; hence, Π_F^{act} is unconditionally private.

Furthermore, $\Pr[\varphi - \psi = 0]$, which is the probability that an attack on correctness is successful, is at most the following negligible probability.

$$
\frac{p^d - 1}{p^d} \frac{1}{p^d} + \frac{1}{p^d} = 2p^{-d} - p^{-2d} \qquad (3)
$$

Inversely, if $\top$ is output in the Proof Phase, the outputs of Π_F^{act} are correct in the probability $1 - 2p^{-d} + p^{-2d}$.

(ii) When the adversary cheats only in the Proof Phase, the cheating does not affect the correctness of the outputs. Regarding privacy, we can obtain $\varphi - \psi = \chi r - \nu$ from Formula (1). When the adversary sets χ as 0, $\varphi - \psi = \nu$ holds, and ν is the value the adversary knows. Otherwise, $\varphi - \psi$ is only a uniformly random value.

Finally, since the choice of the set of k honest parties was arbitrary, all plaintexts reconstructed from any k honest parties were correct except for a negligible probability.

□ (Theorem 2 and Theorem 3)

G Consistency Check and Amortized $O(n)$ Communication Correct Reveal

We show a parallel consistency check protocol in Scheme 13. One can use this protocol to check the consistency of inputs before the Randomization Phase in our construction. The communication complexity per input is $O(n^2/m)$ field elements. In a typical multi-party setting where all n parties have their inputs, amortized communication complexity can be written as $O(n)$ field elements. Round complexity is $O(1)$.

Scheme 13 [Parallel Consistency Check]
Input: $[\![a_0]\!], \cdots, [\![a_{m-1}]\!]$
Output: $\top$ if all of $[\![a_0]\!], \cdots, [\![a_{m-1}]\!]$ are consistent, or $\perp$ otherwise

Parties generate a random shared value $[\![r]\!]$.
SYNC
for each $i < n$
    Party P_i generates a random value s_i and distributes it to all other parties.
Each party computes $s := \Sigma_{i<n} s_i$.
Parties compute $[\![c]\!] := \Sigma_{i<m-1} s^{i+1} [\![a_i]\!] + s^{m+1} [\![a_{m-1}]\!]$.
Parties compute $[\![d]\!] := [\![c - r]\!]$.
Parties reveal $[\![c - r]\!]$ by correct reveal (Scheme 12).
If any cheating is detected during the reveal protocol, parties output $\perp$. Otherwise, parties output $\top$.

Scheme 14 is an amortized $O(n)$ communication perfectly correct reveal of consistent shares. Note that in linear secret sharing schemes, the computation of each share from k other shares and the reconstruction are both linear combinations, and thus, they can be executed in parallel on the linear IDA (Information Dispersal Algorithm) shares. (We say an algorithm that satisfies the same condition as LSSS except for privacy is a linear IDA. In (k, n)-linear IDAs, a shared value can store k values.) The total communication complexity of Scheme 14 is $O(n^2)$ field elements, and amortized complexity is $O(n)$ field elements. Although Scheme 14 requires consistent shares, the combination with Scheme 13 becomes a correct reveal of possibly inconsistent shares.

Scheme 14 Efficient Correct Reveal of Consistent Shares
Input: $[\![a_0]\!], \ldots, [\![a_{k-1}]\!]$
Output: $a_0, \ldots, a_{k-1}$

for each $0 \le i \le n-1$, P_i **do**
    $[\![\overrightarrow{a}]\!]_i = ([\![a_0]\!]_i, \ldots, [\![a_{k-1}]\!]_i)$
    share $[\![\overrightarrow{a}]\!]_i$ using a linear IDA scheme. Parties get $[[\![\overrightarrow{a}]\!]_i]$, where $[\cdot]$ denotes a shared value of a linear IDA.
Each party executes the LSSS's correct reconstruction of $[[\![\overrightarrow{a}]\!]]_j$ on the IDA using homomorphism and gets $\lceil \overrightarrow{a} \rceil_j$ as follows.
    for each $j < n$, P_j **do**
        $c_j :=$ true
        for each $k \le i < n$
            compute the i-th share of the LSSS from $[[\![\overrightarrow{a}]\!]_0]_j, \ldots, [[\![\overrightarrow{a}]\!]_{k-1}]_j$ and let the result be a'_{ji}.
            if $a'_{ji} \neq [[\![\overrightarrow{a}]\!]_i]_j$ **then** $c_j :=$ false
        for each $i < n$ s.t. $i \neq j$
            send $c_{ji} := c_j$ to P_i
    for each $i < n$, P_i **do**
        if $\bigwedge_{j<n} c_{ji} =$ false **then** output $\perp$
        else compute the reconstruction of the LSSS on $[[\![\overrightarrow{a}]\!]_0]_j, \ldots, [[\![\overrightarrow{a}]\!]_{k-1}]_j$ and let the result be $\lceil \overrightarrow{a} \rceil_j$.
Parties reconstruct $\lceil \overrightarrow{a} \rceil$ in a correct manner as follows.
    for each $j < n$, P_j **do**
        $c_j :=$ true
        for each $k \le i < n$
            compute the i-th share a''_{ji} from $\lceil \overrightarrow{a} \rceil_0, \ldots, \lceil \overrightarrow{a} \rceil_{k-1}$.
            if $a''_{ji} \neq \lceil \overrightarrow{a} \rceil_i$ **then** $c_j :=$ false
        for each $i < n$ s.t. $i \neq j$
            send $c_{ji} := c_j$ to P_i
    for each $i < n$, P_i **do**
        if $\bigwedge_{j<n} c_{ji} =$ false **then** output $\perp$
        else output $\overrightarrow{a} (= (a_0, \ldots, a_{k-1}))$ by the reconstruction of the IDA.