# Efficient Bit-Decomposition and Modulus Conversion Protocols with an Honest Majority

Ryo Kikuchi[1], Dai Ikarashi[1], Takahiro Matsuda[2], Koki Hamada[1], and Koji Chida[1]

[1] NTT Corporation,
{kikuchi.ryo, ikarashi.dai, hamada.koki, chida.koji}@lab.ntt.co.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST),
t-matsuda@aist.go.jp

1.

**Abstract.** We propose secret-sharing-based bit-decomposition and modulus conversion protocols for a prime order ring $\mathbb{Z}_p$ with an honest majority: an adversary can corrupt $k-1$ parties of $n$ parties and $2k-1 \leq n$. Our protocols are secure against passive and active adversaries depending on the components of our protocols. We assume a secret is an $\ell$-bit element and $2^{\ell+\lceil \log m \rceil} < p$, where $m = k$ in the passive security and $m = \binom{n}{k-1}$ in the active security. The outputs of our bit-decomposition and modulus-conversion protocols are $\ell$ tuple of shares in $\mathbb{Z}_2$ and a share in $\mathbb{Z}_{p'}$, respectively, where $p'$ is the modulus to be converted. If $k$ and $n$ are small, the communication complexity of our passively secure bit-decomposition and modulus-conversion protocols are $O(\ell)$ bits and $O(\lceil \log p' \rceil)$ bits, respectively. Our key observation is that a quotient of additive shares can be computed from the *least* significant $\lceil \log m \rceil$ bits. If a secret $a$ is "shifted" and additively shared by $x_i$ in $\mathbb{Z}_p$ as $2^{\lceil \log m \rceil}a = \sum_{i=0}^{m-1} x_i = 2^{\lceil \log m \rceil}a + qp$, the least significant $\lceil \log m \rceil$ bits of $\sum_{i=0}^{m-1} x_i$ determines $q$ since $p$ is an odd prime and the least significant $\lceil \log m \rceil$ bits of $2^{\lceil \log m \rceil}a$ are 0s.

**Keywords:** bit-decomposition, modulus conversion, secure computation, secret sharing

## 1 Introduction

Secure computation enables *parties* with inputs to compute a function on the inputs while keeping them secret. There are security notions that secure computation should satisfy, e.g., privacy, meaning the protocol reveals nothing except the output, and correctness, meaning the protocol computes the desired function. These notions should be satisfied in the presence of an adversary, and there are two classical adversary models according to adversaries' behaviors: passive (i.e., semi-honest) and active (i.e., malicious). Passive security means an adversary follows the protocol but may try to learn something from the protocol transcript, and active security means the adversary tries to cheat with an arbitrary strategy including deviating from the protocol. Active security provides stronger security guarantee but passive security is sufficient in some cases, e.g., each party somewhat trusts each other but cannot share their information due to privacy regulations, parties cannot tamper with an installed program of secure computation, and the only thing they can do is seeing the input and output.

An adversary can corrupt a party to see its input and output and control its behavior. There are two major settings specifying the number of parties the adversary can corrupt. Honest majority means an adversary can corrupt less than half the parties, and dishonest majority means it can corrupt more than half. Security with a dishonest majority provides stronger security guarantee but security with an honest majority is sufficient in some cases, for example, each party is a "somewhat" trusted authority, such as a government agency of a different country that may not collude with other agencies.

Secure computation can accelerate an application of sensitive data since one can analyze data while they are secret by using secure computation, e.g., detecting tax fraud [3] and aggregating clinical information [17]. Despite the advantage of secure computation, it has not been widely used in practice. One of main reasons is its inefficiency. Secure computation tends to require heavy computations and communication; thus, its performance is typically much lower than that of local computation when the same function is computed. Therefore, to achieve better performance is one of the main challenges in secure computation.

### 1.1 Bit decomposition and modulus conversion

When we are interested in secure computation on an integer input $a \in \mathbb{Z}_p$, there are two major representations to describe an intended function: an arithmetic circuit and a Boolean circuit. An input and output of an arithmetic circuit are represented as elements in $\mathbb{Z}_p$, while those of a Boolean circuit are in $\mathbb{Z}_2$.

Secure computation in better suited representation provides better performance. For example, addition and multiplication (in $\mathbb{Z}_p$) can be computed efficiently by an arithmetic circuit, while not by a Boolean circuit. In contrast, bit-operations, such as comparison and Hamming weight, can be computed efficiently by a Boolean circuit, while those operations are non-trivial tasks for an arithmetic circuit.

To bridge these two representations, Damgård et al. [9] and Schoenmakers and Tuyls [23] proposed bit-decomposition protocols to convert the integer representation into the binary one. The former is a secret-sharing (SS)-based protocol and unconditionally secure with an honest majority, while the latter is a homomorphic-encryption-based protocol and computationally secure without an honest majority. In the honest majority case, several subsequent works have improved the efficiency [21,10,22,25,4].

There are two types in SS-based bit-decomposition protocols based on whether each bit of the bit-decomposition result of an original secret is in $\mathbb{Z}_p$ or in $\mathbb{Z}_2$. If these bits are shared in $\mathbb{Z}_p$, it is easy to convert the bit representation into an integer representation after computations with a Boolean circuit. In contrast, if these bits are shared in $\mathbb{Z}_2$, a Boolean circuit can be computed efficiently since the parties can *locally* compute an XOR gate. In this paper, we focus on the latter type of output; the output of the bit-decomposition protocol is shares in $\mathbb{Z}_2$.

A modulus-conversion protocol is a related protocol that converts a share in $\mathbb{Z}_p$ into that in $\mathbb{Z}_{p'}$ (with $p \neq p'$) without changing an original secret. This protocol corresponds to a type-casting operation (i.e., type conversion) for ordinary computers. In many applications, a user of secure computation may want to obtain values that are *not* reduced by modulus. For example, if we intend to obtain the sum of shared secrets, we want to obtain $\sum a_i$, not $\sum a_i \mod p$. In this case, we have to manage the shared values not to exceed the modulus $p$. However, if we do not know which function will be computed with shared secrets, we cannot determine beforehand how large $p$ should be. Even if we use a large enough $p$ for most functions, the communication complexity of secure computation is at least proportional to $\log p$ and the efficiency therefore decreases. The modulus-conversion protocol can be a solution of this problem; namely, when an output of secure computation will exceed $p$, we can change $p$ into $p'$, which is large enough to represent the output. Another application of a modulus-conversion protocol is the inverse of a bit-decomposition protocol by setting $p = 2$.

### 1.2 Our contribution

We propose an SS-based bit-decomposition protocol for $\mathbb{Z}_p$ and modulus-conversion protocol from $\mathbb{Z}_p$ to $\mathbb{Z}_{p'}$ with low communication complexity and an honest majority, where $p$ and $p'$ are prime numbers. Our basic protocols are passively secure, but can be made actively secure if the number of parties is small. In this paper, we consider active security *with abort* in which an honest party will abort if an adversary cheats. In our protocols, it is assumed that the parties know the bit-length $\ell$ of a secret, i.e., a secret $a$ satisfies $a < 2^{\ell+1}$. Therefore, the output of our bit-decomposition protocol is $\ell$ shares in $\mathbb{Z}_2$.[3] We also assume $\ell + \lceil \log m \rceil < \lceil \log p \rceil$, where $m = k$ in the passive security case and $m = \binom{n}{k-1}$ in the active security case, where $k$ is the number of parties who can reconstruct the secret and $n$ is the number of all parties. It seems natural that $\lceil \log p \rceil$ is somewhat larger than $\ell$ and the parties know $\ell$ to prevent an output of secure computation from exceeding $p$; nevertheless, our protocol supports neither full extraction of the bits of secret nor too many parties in which $\ell + \lceil \log m \rceil \geq \lceil \log p \rceil$. In addition, $\binom{n}{k-1}$ is exponential in $n$ so our actively secure protocol is only for small number of parties.

Our protocols consist of bit-wise share generation, random share generation, and Boolean circuit evaluation. If $p$ is a Merssene prime, both of our protocols can be simplified and their communication complexity is improved in a constant factor. By using ordinary circuits and regarding $k$ and $n$ as constants, the communication complexity of our bit-decomposition protocol is $O(\ell)$ bits, which seems optimal since the output is an $\ell$-tuple of $\mathbb{Z}_2$. For the specific parameters of $(k, n) = (2, 3)$ and when $p$ is a Merssene

[3] If one wants to use Shamir's SS scheme, $\mathrm{GF}(2^{\lceil \log n \rceil + 1})$ can be an alternative option.

prime, the communication complexity is $10\ell + 4$ bits, which is smaller than the best known result [4] of $17\lceil \log p \rceil + 12\lceil \log \lceil \log p \rceil \rceil$ bits, while [4] supports full extraction of bits and uses a different ring $p = 2^d$. Our modulus-conversion protocol has a similar structure, and the communication complexity is $O(\lceil \log p' \rceil)$ bits, which seems also optimal since the output of the protocol is a share in $\mathbb{Z}_{p'}$. We note that out protocols are not constant-round protocols. Nonetheless, the round complexity is comparable to that of constant-round protocols when $(k, n) = (2, 3)$.

### 1.3 Technical overview

A common difficulty in constructing bit-decomposition and modulus-conversion protocols is secure computation of a *quotient*. Let $a \in \mathbb{Z}_p$ be a secret and assume $a$ is *additively* shared as $a = \sum_{i=0}^{m-1} x_i \mod p$. When we intend to obtain a share of $a$ in $\mathbb{Z}_2$, one may try to replace $x_i$ with $x_i \mod 2$. However, it does not work since $\sum_{i=0}^{m-1} x_i \mod 2 = a + qp \mod 2 = a + (q \mod 2)(p \mod 2) \neq a$. Here, $p$ is public, but $q$ is unknown and thus $q$ should be securely computed. A naïve way to obtain $q$ is to securely compute $\sum_{i=0}^{m-1} x_i$ by a Boolean circuit and compare $\sum_{i=0}^{m-1} x_i$ with $p$. However, this naïve method requires $O(\lceil \log p \rceil)$-bit communication to compute $\sum_{i=0}^{m-1} x_i$ by a Boolean circuit.

Our key observation is that a quotient of additive shares can be computed from the *least* significant $u$ bits, and we call this property the *quotient transfer*. In both of our protocols, we first additively share $2^u a$ rather than $a$, i.e., $\sum_{i=0}^{m-1} x_i = 2^u a + qp$. Recall that we assume $\ell + u \leq \lceil \log p \rceil$, and thus $2^u a \mod p = 2^u a$. We observe that the least significant $u$ bits of $\sum_{i=0}^{m-1} x_i$ represents $q$ since $p$ is an odd prime and the least significant $u$ bits of $2^u a$ are 0s. Therefore, we can obtain $q$ by securely computing the least significant $u$ bits of $\sum_{i=0}^{m-1} x_i$. By using the quotient transfer, $\ell + u$ bits and $\lceil \log p' \rceil + u$ bits of $\sum_{i=0}^{m-1} x_i$ are sufficient for our bit-decomposition and modulus-conversion protocols, respectively.

### 1.4 Related work

Damgård et al. [9] proposed a constant round bit-decomposition protocol, which were simplified by Nishide and Ohta [21]. Toft proposed another bit-decomposition protocol [25] with almost linear communication complexity, and Reistad and Toft [22] proposed a bit-decomposition protocol with linear communication complexity while admitting statistical privacy. In these works, the output of a bit-decomposition protocol is shares in $\mathbb{Z}_p$, and linear communication complexity means that the number of invocations of a multiplication protocol is linear in $\lceil \log p \rceil$. In this paper, we measure the communication complexity in bits among the all parties. With respect to the communication complexity in bits, the above mentioned existing protocols incur at least $O(\lceil \log p \rceil^2)$ since a multiplication protocol requires $O(\lceil \log p \rceil)$-bit communication.

A bit-decomposition protocol that outputs XOR-free shares was proposed by From and Jakobson [10]. They use a share in $\mathrm{GF}(2^{256})$ as an output. Bogdanov et al. [4] proposed a bit-decomposition protocol that is dedicated to the replicated SS scheme [8,16] in $(k, n) = (2, 3)$ and $p = 2^d$ for some positive integer $d$. The output of their bit-decomposition protocol is $\lceil \log p \rceil$ shares in $\mathbb{Z}_2$ since they support full extraction.

Regarding modulus conversion, Bogdanov et al. [4] proposed a specific case of a modulus-conversion protocol from $\mathbb{Z}_2^{\lceil \log p \rceil}$ into $\mathbb{Z}_p$. This protocol is also dedicated to the replicated SS scheme in $(k, n) = (2, 3)$ and $p = 2^d$ for some positive integer $d$. This protocol is the inverse of a bit-decomposition protocol.

## 2 Preliminaries

Let $a := b$ denote that $a$ is defined by $b$, and $a||b$ denote the concatenation of $a$ and $b$. If $a$ is an $\ell$-bit element, $a^{(i)}$ denotes the $i$-th bit of $a$, where we count the indices in the right-to-left order with 0 being the initial index, i.e., $a := a^{(\ell-1)}||\cdots||a^{(0)}$. If $\mathsf{A}$ is a probabilistic algorithm, $a \leftarrow \mathsf{A}(b)$ means $a$ is the output of $\mathsf{A}$ on input $b$. The notations $\mathscr{R}$, $\mathbb{Z}$, $\mathbb{Z}_p$, and $\mathbb{Z}_p^m$ denote a ring, the set of integers, $\mathbb{Z}/p\mathbb{Z}$, and $m$-tuple of the elements in $\mathbb{Z}_p$, respectively. For a relation $R$, $\langle R \rangle$ denotes 1 if $R$ is true and 0 otherwise. For example, $\langle a <_? b \rangle$ denotes 1 if $a < b$ and 0 otherwise.

### 2.1 Mersenne prime

A Mersenne prime is a prime number of the form $p = 2^e - 1$ for some integer $e$. It provides efficient modular arithmetic, e.g., [5], since modulo a Mersenne prime can be computed by bit-shifting and addition: If $a = a_0 2^e + a_1$, then $a \mod p = a_0 2^e + a_1 \mod p = a_0 + a_1 \mod p$ holds since $2^e - 1 = 0 \mod p$.

### 2.2 Security model and definition

We consider SS-based secure computation with an honest majority. In this setting, there are $n$ parties $P_0, \ldots, P_{n-1}$, a secret is shared among the $n$ parties via SS, any $k$ parties can reconstruct the secret from their shares, and an adversary corrupts up to $k-1$ parties at the beginning of the protocol, where $2k - 1 \leq n$.

We consider the client/server model. This model is used to outsource secure computation, where any number of clients send shares of their inputs to the servers. Therefore, both the input and output of the servers are shares, and both of our protocols are therefore share-input and share-output protocols.

Regarding adversarial behaviors, we consider two security models: passive and active security with abort. We give a brief explanation here and the formal definitions of security appear in Appendix A.

*Passive security.* In passive security, corrupted parties follow a protocol. Therefore, a passive adversary tries to obtain information about a secret from transcripts that the corrupted parties have. Formally, we say that a protocol is passively secure if there is a simulator that simulates the view of the corrupted parties from the inputs and outputs of the protocol [13].

*Active security with abort.* In this paper, an actively secure protocol is a secure computation *with abort*. This means that if an adversary cheats, an honest party will abort. This security model does not guarantee fairness: An adversary may obtain the outputs of corrupted parties while the honest parties do not.[4] Note that we do not care about fairness even it is possible with an honest majority. This is because efficient circuit evaluation protocols are known [11,12] in this security model, and it may be difficult to reveal a secret without abort efficiently. From here on, in this paper, active security means active security with abort.

### 2.3 Secret sharing

We use an unconditionally secure linear SS scheme [2] that supports the following algorithms, protocols, and local operations.

- Share: On input $a \in \mathscr{R}$, this algorithm outputs shares of $a$. The notation $[a]_i$ denotes $P_i$'s share and $[a]$ denotes a sharing, which is a tuple of all shares. Several rings will appear, and thus we explicitly indicate the ring to which shares/sharings belong. For example, $[a]^{\mathbb{Z}_p}$ denotes a sharing of $a$ in $\mathbb{Z}_p$, while $[a]^{\mathbb{Z}_2}$ denotes a sharing of $a$ in $\mathbb{Z}_2$. In addition, $[a]^{\mathbb{Z}_2^m}$ denotes a tuple of sharings $([a_0]^{\mathbb{Z}_2}, \ldots, [a_{m-1}]^{\mathbb{Z}_2})$, where $a = \sum_{i=0}^{m-1} 2^i a_i$.
- Reconstruction: On input $k$ shares, this algorithm outputs a secret. For any linear SS scheme, a secret can be reconstructed by a linear combination of $k$ shares. For example, we denote the linear combination of the shares of $P_0, \ldots, P_{k-1}$ as $a = \sum_{i=0}^{k-1} \lambda_i [a]_i$ for some $\lambda_i$.[5]
- Reveal: This is a protocol for reconstructing a secret from its shares. The requirements of this protocol are different depending on considered security models. In the presence of a passive adversary, given a sharing of $a$, this protocol guarantees that at the end of the execution, all the parties obtain $a$. When we consider an active adversary, this protocol guarantees that at the end of the execution, if $[a]$ is not correct, i.e., either a secret reconstructed from some $k$ shares is $\perp$ or does not equal to that from other $k$ shares, then all the honest parties will abort. Otherwise, if $[a]$ is correct, then each party will either output $a$ or abort.

[4] In our protocols, the output of our protocols are shares, so the adversary cannot obtain any secret information.
[5] This is a slightly small class of SS schemes compared to [2] with respect that each party has a single share.

**Protocol 1** Share conversion from a linear SS scheme into the replicated SS scheme

**Input:** $[a]^{\mathbb{Z}_p}$
**Output:** $[\![a]\!]^{\mathbb{Z}_p}$
1: The parties call $\mathcal{F}_{\text{rand}}$ and receive $[\![r]\!]^{\mathbb{Z}_p}$.
2: The parties locally convert $[\![r]\!]^{\mathbb{Z}_p}$ into $[r]^{\mathbb{Z}_p}$.
3: The parties reveal $[a-r]^{\mathbb{Z}_p}$ and obtain $a-r$.
4: $[\![a]\!]^{\mathbb{Z}_p} := (a-r) + [\![r]\!]$.
5: The parties output $[\![a]\!]^{\mathbb{Z}_p}$.

- Local operations: Given sharings $[a]$ and $[b]$ and a scalar $\alpha \in \mathscr{R}$, the parties can generate sharings of $[a+b]$, $[\alpha a]$, and $[\alpha + a]$ using only local operations. The notations $[a]+[b]$, $\alpha[a]$, and $\alpha + [a]$ denote these local operations, respectively.
- Multiplication protocol and secure circuit evaluation: Given sharings $[a]$ and $[b]$, the parties can generate $[ab]$ by the multiplication protocol. Combining local operations with the multiplication protocol, we can compute any Boolean circuit over shared data.

Concrete examples of a linear SS scheme is Shamir's scheme [24] and the replicated SS scheme [8,16]. In this paper, we use $\mathbb{Z}_p$, $\mathbb{Z}_{p'}$, and $\mathbb{Z}_2$ as instantiations of a ring, where $p$ and $p'$ are prime numbers. We especially say that $a$ is *additively* shared in $\mathbb{Z}_p$ if $a = \sum_{i=0}^{m-1} x_i \mod p$ for some $m$, and we call $x_i$ a *sub-share*.

Although an input and output of our protocols can be shares of any linear SS scheme, the shares have to be converted into one of the replicated SS scheme in our actively secure protocols. The share size of the replicated SS scheme is exponential in $n$; therefore, our protocols with active security are for a small number of parties, whereas our protocols with passive security do not have this restriction.

*Replicated secret sharing scheme.* The replicated SS scheme [8,16] is an SS scheme in which a secret is represented as an addition of sub-shares and each sub-share corresponds to a maximal unqualified set of parties.

Let $m := \binom{n}{k-1}$ and $\mathbb{T} = \{\mathbb{T}_0, \ldots, \mathbb{T}_{m-1}\}$ be a family of all $(k-1)$-subsets of $\{0, \ldots, n-1\}$. We especially use the notation $[\![\cdot]\!]_i$ (resp. $[\![\cdot]\!]$) for a share (resp. a sharing) of the replicated SS scheme. Shares of the replicated SS scheme in $\mathbb{Z}_p$ are generated as follows. A secret $a$ is additively shared into $m$ sub-shares as $a = \sum_{i=0}^{m-1} x_i \mod p$, and a share for $P_i$ is $[\![a]\!]_i = \{x_j \mid i \notin \mathbb{T}_j, \mathbb{T}_j \in \mathbb{T}\}$. Here, $k-1$ parties cannot obtain any information about $a$ since there exists $\mathbb{T}_j$ that contains all the corrupted parties, and an adversary cannot know $x_j$.

The size of a share of the replicated SS scheme becomes very large for a large number of parties since each party has $\binom{n-1}{k-1}$ sub-shares. However, the replicated SS scheme has an attractive property that the parties can generate a share of a random number *without interaction*, which is called pseudo-random secret sharing (PRSS) [8]. Formally, PRSS securely computes the following functionality $\mathcal{F}_{\text{rand}}$.

**FUNCTIONALITY 2.1 ($\mathcal{F}_{\text{rand}}$ – Generating a share of a random value)**
Upon receiving $id$ from $P_i$ for $0 \le i < n$, $r \leftarrow \mathbb{Z}_p$, generate $[\![r]\!]^{\mathbb{Z}_p}$ by the sharing algorithm, and send $[\![r]\!]_i^{\mathbb{Z}_p}$ to $P_i$

*Share conversion among SS schemes.* It is known that shares can be converted among additive shares, a linear SS scheme, and the replicated SS scheme.

A share of a linear SS scheme $[a]_i$ can be locally converted to additive shares with $k$ sub-shares by setting $x_i := \lambda_i [a]_i$, where $P_i$ has $x_i$ for $0 \le i < k$. On the contrary, when $P_i$ for $0 \le i < k$ has an additive share $x_i$, the shares can be converted by sharing all the sub-shares $x_i$ via a linear SS scheme and adding them all.

A share of the replicated SS scheme can be locally converted into that of a linear SS scheme [8]. On the contrary, a share of a linear SS scheme can be converted into that of the replicated SS scheme by using Protocol 1. This protocol is actively secure in the $\mathcal{F}_{\text{rand}}$-hybrid model since we assume that the reveal protocol is actively secure.

*Secure circuit evaluation on linear SS.* In our protocols, several circuits are securely computed. We consider the sum, carryless-sum, and zero-test circuits. The sum circuit on input $m$ $\ell$-bit elements, outputs $\lceil \log m \rceil + \ell$-bit element that is the sum of the inputs. The carryless-sum circuit is the same as the sum circuit except the output is $\ell$-bit element by discarding the most significant $\lceil \log m \rceil$ bits. The zero-test circuit on input $m$ 1-bit elements, outputs 0 if all the inputs are 0, and 1 otherwise. We construct our protocols in a modular way using the functionalities $\mathcal{F}_{\text{sum}}$, $\mathcal{F}_{\text{clsum}}$, and $\mathcal{F}_{\text{zero}}$ that correspond to the sum, carryless-sum, and zero-test circuits, respectively. Formal description of those functionalities appear in Appendix C.

## 3 Quotient transfer

In this section, we show our key observation that we call *quotient transfer*. Informally, quotient transfer means that, if "shifted" secret $2^u a$ is additively shared as $\sum_{i=0}^{m-1} x_i = 2^u a + qp$, the parties can compute the quotient $q$ from the least significant $u$ bits of $\sum_{i=0}^{m-1} x_i$, where $u = \lceil \log m \rceil$.

**Theorem 3.1.** *Let $m$ be a positive integer, $u = \lceil \log m \rceil$, and $2^u < p$. Let $(x_0, \ldots, x_{m-1})$ be a tuple of elements in $\mathbb{Z}_p$ satisfying $\sum_{i=0}^{m-1} x_i = 2^u a + qp$. Then, the quotient $q$ satisfies*

$$q = (p \mod 2^u)^{-1} \sum_{i=0}^{m-1} x_i \mod 2^u. \tag{1}$$

[**Proof**] We observe

$$\sum_{i=0}^{m-1} x_i \mod 2^u = 2^u a + qp \mod 2^u = q(p \mod 2^u) \mod 2^u$$

since $q \leq m - 1 < 2^u$. In addition, $2^u$ and $(p \mod 2^u)$ are co-prime, and thus $(p \mod 2^u)^{-1}$ exists. ■

The prime number $p$ is public, and thus $(p \mod 2^u)^{-1}$ can be computed by every party. Therefore, Equation 1 means that the quotient $q$ can be computed from the least significant $u$ bits of $\sum_{i=0}^{m-1} x_i$.

For practical applications, protocols with a small number of parties may be used, and we will later consider the case $m = 2$ (i.e. three-party case). Furthermore, for performance reasons, a Mersenne prime is used for $p$. Therefore, in the following, we give specific cases of Theorem 3.1 for these cases. The second equation below shows that the parties can compute the quotient $q$ from the LSB of $\sum_{i=0}^{m-1} x_i$ in a secure three-party computation when $p$ is a Mersenne prime.

**Corollary 3.2.** *If $p$ is a Mersenne prime, i.e., $p = 2^e - 1$, Equation (1) is*

$$q = -\sum_{i=0}^{m-1} x_i \mod 2^u$$

*since $p \mod 2^u = -1$. Furthermore, when $m = 2$,*

$$q = x_0 + x_1 \mod 2.$$

## 4 Bit-decomposition protocol

In this section, we first show a useful equation for our proposed protocols, then show our passively secure bit-decomposition protocol. After that, we discuss a technique to obtain active security. Here, we show the protocol in which $p$ is a Mersenne prime, and will give a general protocol in Appendix B.1.

### 4.1 Equation for Bit-decomposition

The following equation can be derived from quotient transfer.

**Theorem 4.1.** *Let $m, u, p, a, (x_0, \ldots, x_{m-1})$ be the same as Theorem 3.1, and $\ell$ be a positive integer such that $\ell + u \leq |p|$ and $a < 2^{\ell+1}$. Let $r_u = \sum_{i=0}^{m-1} x_i \mod 2^u$, $\tilde{p} = (p \mod 2^u)^{-1} \mod 2^u$, and $q_u$, $z$, and $z'$ be the quotients of $\sum_{i=0}^{m-1} x_i/2^u$, $\tilde{p}\sum_{i=0}^{m-1} x_i/2^u$, and $p\tilde{p}/2^u$ in modulo $2^{\ell+u}$, respectively. Then,*

$$a = q_u - z' r_u - zp \mod 2^\ell.$$

[**Proof**] Let $q$ be a quotient of $\sum_{i=0}^{m-1} x_i$ divided by $p$, i.e., $\sum_{i=0}^{m-1} x_i = qp + 2^u a$. Here, $2^u a = \sum_{i=0}^{m-1} x_i - qp$ in $\mathbb{Z}$, therefore, $2^u a = -qp + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u}$. Recall that $\tilde{p} = (p \mod 2^u)^{-1}$ in modulo $2^u$. From Theorem 3.1,

$$-qp + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u} = -(\tilde{p}\sum_{i=0}^{m-1} x_i \mod 2^u)(p \mod 2^{\ell+u}) + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u}. \tag{2}$$

Recall that $r_u = \sum_{i=0}^{m-1} x_i \mod 2^u$ and $z$ is the quotient of $\tilde{p}\sum_{i=0}^{m-1} x_i/2^u \mod 2^{\ell+u}$. Then, Equation (2) is equal to

$$\begin{aligned}
&-(\tilde{p} r_u \mod 2^u)(p \mod 2^{\ell+u}) + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u} \\
&= -(\tilde{p} r_u - z2^u \mod 2^{\ell+u})(p \mod 2^{\ell+u}) + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u} \\
&= -p\tilde{p} r_u - zp2^u + \sum_{i=0}^{m-1} x_i \mod 2^{\ell+u}
\end{aligned} \tag{3}$$

Recall that $q_u$ and $z'$ are the quotients of $\sum_{i=0}^{m-1} x_i/2^u \mod 2^{\ell+u}$ and $p\tilde{p}/2^u \mod 2^{\ell+u}$, respectively. In addition, $p\tilde{p} = 1 \mod 2^u$. Then, Equation (3) is equal to

$$-(z'2^u + 1)r_u - zp2^u + q_u 2^u + r_u \mod 2^{\ell+u} = (q_u - z' r_u - zp)2^u \mod 2^{\ell+u}.$$

Consequently, we obtain $2^u a = (q_u - z' r_u - zp)2^u \mod 2^{\ell+u}$. By by dividing both sides by $2^u$, We finally obtain

$$a = q_u - z' r_u - zp \mod 2^\ell.$$

This concludes the proof. ∎

We give a specific case of Theorem 4.1 in which $p$ is a Mersenne prime as follows.

**Corollary 4.2.** *Under the same setting as in Theorem 4.1, if $p$ is a Mersenne prime, i.e., $p = 2^e - 1$ for some integer $e$, it holds that*

$$a = q_u + \langle r_u \neq_? 0 \mod 2^u \rangle \mod 2^\ell. \tag{4}$$

[**Proof**] If $p = 2^e - 1$, then $\tilde{p} = 2^u - 1 \mod 2^{\ell+u}$. In addition, $z' = -1 \mod 2^{\ell+u}$ holds since $p\tilde{p} = (2^e - 1)(2^u - 1) \mod 2^{\ell+u} = -2^u + 1 \mod 2^{\ell+u}$.

Recall that $z$ satisfies $\tilde{p} r_u \mod 2^u = \tilde{p} r_u - z2^u \mod 2^{\ell+u}$. By substituting $\tilde{p} = 2^e - 1 \mod 2^{\ell+u}$,

$$-r_u \mod 2^u = (2^u - 1)r_u - z2^u \mod 2^{\ell+u}$$

$$z2^u \mod 2^{\ell+u} = r_u 2^u - (-r_u \mod 2^u) - r_u \mod 2^{\ell+u}$$

Here,

$$-r_u \mod 2^u = -\sum_{i=0}^{m-1} x_i \mod 2^u = \begin{cases} 0 & \text{if } \sum_{i=0}^{m-1} x_i \mod 2^u = 0 \\ 2^u - 1 & \textit{otherwise} \end{cases}$$

Therefore, if $r_u = 0$, then $z = r_u$; otherwise, $z = r_u - 1$. This is equivalent to $z = r_u - \langle r_u \neq_? 0 \rangle$. By substituting the above into Theorem 4.1, we conclude the proof. ■

Theorem 4.1 and Corollary 4.2 show that $a$ can be represented from the $\ell + u$ bits of $\sum_{i=0}^{m-1} x_i$. We further obtain the following corollary since it is convenient that an equation is represented by bit-operations of sub-shares. The following corollary is in fact securely computed in our bit-decomposition protocol.

**Corollary 4.3.** *Let $m, u, p, a, (x_0, \ldots, x_{m-1})$ be the same as Theorem 4.1. Let $q_i$ and $r_i$ be the bits of $x_i$ larger than $u-1$ bit and those smaller than $u$ bit, respectively, and $q_u$ and $r_u$ be the bits of $\sum_{i=0}^{m-1} r_i$ larger than $u-1$ bit and those smaller than $u$ bit, respectively. Then,*

$$a = \sum_{i=0}^{m-1} q_i + q_u + \langle r_u \neq_? 0 \rangle \mod 2^\ell.$$

### 4.2 Passively secure bit-decomposition protocol

Our passively secure bit-decomposition protocol for $\mathbb{Z}_p$ with a Mersenne prime $p$, is derived from Corollary 4.3 as Protocol 2.

**Protocol 2** Passively secure bit-decomposition protocol

**Input:** $[a]^{\mathbb{Z}_p}$
**Output:** $[a]^{\mathbb{Z}_2^\ell}$

1: $P_i$ computes $x_i := 2^u \lambda_i [a]_i \mod p$ for $u = \lceil \log k \rceil$ and $0 \leq i < k$, and let the $j$-th bit of $x_i$ be $x_i^{(j)}$.
2: **for** $0 \leq i < k$ **do**
3: $P_i$ shares $x_i^{(0)}, \ldots, x_i^{(u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[r_i]^{\mathbb{Z}_2^u}$.
4: $P_i$ shares $x_i^{(u)}, \ldots, x_i^{(\ell+u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[q_i]^{\mathbb{Z}_2^\ell}$.
5: The parties call $\mathcal{F}_{\text{sum}}$ on input $[r_i]^{\mathbb{Z}_2^u}$ for $0 \leq i < k$, and receive $[\sum_{i=0}^{k-1} r_i]^{\mathbb{Z}_2^{2u}}$. ($k$ additions yield $2u$-bit output).
6: The parties regard the least $u$ bits of $[\sum_{i=0}^{k-1} r_i]^{\mathbb{Z}_2^{2u}}$ as $[r_u]^{\mathbb{Z}_2^u}$, and the others as $[q_u]^{\mathbb{Z}_2^u}$.
7: The parties call $\mathcal{F}_{\text{zero}}$ on input $[r_u]^{\mathbb{Z}_2^u}$, and receive $[\langle r_u \neq_? 0 \rangle]^{\mathbb{Z}_2}$.
8: The parties call $\mathcal{F}_{\text{clsum}}$ on input $[q_0]^{\mathbb{Z}_2^\ell}, \ldots, [q_{k-1}]^{\mathbb{Z}_2^\ell}$, $[q_u]^{\mathbb{Z}_2^u}$, and $[\langle r_u \neq_? 0 \rangle]^{\mathbb{Z}_2}$, and receive $[a]^{\mathbb{Z}_2^\ell} := [\sum_{i=0}^{k-1} q_i + q_u + \langle r_u \neq_? 0 \rangle]^{\mathbb{Z}_2^\ell}$.
9: The parties output $[a]^{\mathbb{Z}_2^\ell}$.

*Security against a passive adversary.* Protocol 2 consists of share generation and circuit evaluation, and the security of the protocol is therefore directly reduced to them. Informally, share generation does not reveal any information about a secret since SS is unconditionally secure. Therefore, Protocol 2 is passively secure in the $(\mathcal{F}_{\text{sum}}, \mathcal{F}_{\text{clsum}}, \mathcal{F}_{\text{zero}})$-hybrid model.

### 4.3 Efficiency

The communication complexity of our bit-decomposition protocol is $k(\ell+u)\texttt{share}_{\mathbb{Z}_2} + \texttt{sum}_{u,k} + \texttt{clsum}_{\ell,k+2} + \texttt{zerotest}_u$ bits, where $\texttt{share}_{\mathbb{Z}_2}$ denotes the communication complexity to share a bit, $\texttt{sum}_u$ denotes that to securely compute the sum on input $k$ $u$-bit elements, $\texttt{clsum}_{\ell,k+2}$ denotes that to securely compute the carryless-sum circuit on input $k+2$ $\ell$-bit elements[6], and $\texttt{zerotest}_u$ denotes that to securely compute the zero-test circuit on input $u$ 1-bit elements. If $k$ (and $u$) is regarded as a constant, the communication complexity is $O(\ell)$ bits since $\texttt{share}_{\mathbb{Z}_2}$ is constant, $\texttt{clsum}_{\ell,k+2}$ invokes $O(\ell)$ multiplication protocols in $\mathbb{Z}_2$, and a multiplication protocol in $\mathbb{Z}_2$ requires $O(1)$-bits communication per invocation.

[6] Precisely, $k$ $\ell$-bit elements, a $u$-bit element, and a 1-bit element are summed up.

For concrete comparison in a specific parameter, we give a precise communication complexity when $(k, n) = (2, 3)$ and use the replicated SS scheme to share $\mathbb{Z}_2$. We assume that sum and carryless-sum circuits compute a full adder sequentially, and zerotest circuit computes an AND gate sequentially. Here, $u = \lceil \log k \rceil = 1$, $\mathtt{share} = 2$ by using communication-efficient sharing in Appendix D.1, $\mathtt{sum}_{1,2}$ is $\mathtt{Mult}_{\mathbb{Z}_2}$, $\mathtt{clsum}_{\ell,4}$ is $(\ell-1)\mathtt{Mult}_{\mathbb{Z}_2}$, and $\mathtt{zerotest}_u$ requires no communication since $[\langle r_u \neq_? 0 \rangle]^{\mathbb{Z}_2} = [r_u]^{\mathbb{Z}_2} + [q_u]^{\mathbb{Z}_2}$, where $\mathtt{Mult}_{\mathbb{Z}_2}$ denotes the communication complexity of a multiplication protocol in $\mathbb{Z}_2$. If we use the replicated SS scheme, $\mathtt{Mult}_{\mathbb{Z}_2} = 6$ per invocation [15]. Therefore, the communication complexity is $4(\ell + 1) + 6 + 6(\ell - 1) = 10\ell + 4$ bits. This means that, if $\ell \approx \lceil \log p \rceil / 2$, the communication complexity of our bit-decomposition protocol is as large as that of a multiplication protocol in $\mathbb{Z}_p$, which is $6\lceil \log p \rceil$.

There is no bit-decomposition protocol in which $\ell + u < \lceil \log p \rceil$ is assumed and which outputs $[a]^{\mathbb{Z}_2^\ell}$, and thus our protocol is formally incomparable to existing bit-decomposition protocols. If we try to compare our bit-decomposition protocol with existing ones, the most efficient bit-decomposition protocol is [4] and its communication complexity is $5\lceil \log p \rceil + 12(\lceil \log \lceil \log p \rceil \rceil + 1)\lceil \log p \rceil = 17\lceil \log p \rceil + 12\lceil \log \lceil \log p \rceil \rceil$ bits. Even regarding $\lceil \log p \rceil = \ell$, and thus our protocol is about three times faster. However, [4] supports full extraction and $p = 2^m$, and thus it is difficult to simply compare with ours.

The round complexity of our bit-decomposition protocol is $1 + sum_{u,k} + clsum_{\ell,k+2} + zerotest_u$, where $sum_{u,k}$, $clsum_{\ell,k+2}$, and $zerotest_u$ are the round complexities of protocols instantiating $\mathcal{F}_{\mathrm{sum}}$, $\mathcal{F}_{\mathrm{zero}}$, and $\mathcal{F}_{\mathrm{zero}}$, respectively. If $(k, n) = (2, 3)$, the round complexity of our bit-decomposition protocol is $1 + 1 + (\ell - 1) + 0 = \ell + 1$ if we use the same circuits in evaluating the communication complexity.

### 4.4 Achieving active security using replicated SS

We show how to make Protocol 2 secure against an active adversary. Step 1 of the protocol is local computation; therefore, it is secure even against an active adversary. In addition, the steps from Step 5 are secure circuit evaluation. Therefore, if we use an actively secure circuit evaluation protocol, such as [1,19,15,12], these steps are secure against an active adversary, as desired.

The remaining steps are Steps 2, 3, and 4. In general, an adversary may corrupt $P_i$ and share an incorrect $\tilde{x}_i$, and it is difficult to detect it. Therefore, we prevent the adversary from mounting such an attack by making these steps consist only of *local* computations. We show that if a secret is shared via the replicated SS scheme, we can generate a bit-wise share of sub-shares by local computations.

Consequently, our bit-decomposition protocol can be actively secure in the $(\mathcal{F}_{\mathrm{sum}}, \mathcal{F}_{\mathrm{zero}}, \mathcal{F}_{\mathrm{clsum}})$-hybrid model by converting a share by Protocol 1 at first, and then performing local share generation of sub-shares.

The communication complexity of the actively secure version of our protocol is at least $O(\lceil \log p \rceil)$ bits since revealing in Protocol 1 incurs this amount of communication. Therefore, only if a secret is shared via the replicated SS scheme from the beginning, the communication complexity of our actively secure bit-decomposition protocol is $O(\ell)$ bits, while $O(\lceil \log p \rceil)$ for a general linear SS scheme.

*Local share generation of sub-shares in replicated SS.* In the replicated SS scheme, each sub-share $x_i$ is held by $n - k + 1$ parties. To obtain a share of the $j$-th bit of $x_i$, each of the $n - k + 1$ parties sets his sub-share $x'_i$ as the $j$-th bit of $x_i$, and the parties set all the other sub-shares as 0. It trivially holds that $\sum_{i=0}^{m-1} x'_i = x_i$. In general, the parties can locally generate an additive share of $f(x_i) \mod p'$. We give the algorithm in Algorithm 3.

**Algorithm 3** Local share generation of sub-shares in replicated SS

**Input:** The $n - k + 1$ parties have $x_i \in \mathbb{Z}_p$

**Output:** Each $P_i$ has $[\![f(x)]\!]_i^{\mathbb{Z}_{p'}}$

1: The $n - k + 1$ parties who have $x_i$ compute $x'_i = f(x_i) \mod p'$.
2: The parties set the all sub-shares $x'_j$ as 0 except $x'_i$.
3: Each $P_i$ outputs $[\![f(x)]\!]_i = \{x'_j \mid i \notin \mathbb{T}_j, \mathbb{T}_j \in \mathbb{T}\}$.

We give an example to obtain a bit-wise share of sub-shares in the case of $(k, n) = (2, 3)$ and $p' = 2$: Before starting the protocol, $P_0$ has $(x_0, x_1)$, $P_1$ has $(x_1, x_2)$, and $P_2$ has $(x_2, x_0)$, where $a = x_0 + x_1 + x_2$

mod $p$. The parties $P_0$, $P_1$, and $P_2$ regard $(x_0^{(0)}, 0)$, $(0, 0)$, $(0, x_0^{(0)})$ as their shares of $x_0^{(0)}$, respectively. By recursively doing the same procedure for the other bits, the parties obtain the bit-by-bit shares of $x_0$, $x_1$, and $x_2$.

## 5 Modulus-conversion protocol

When we consider modulus conversion, computing the quotient also has an important role. Let us consider the case in which we want to convert a share of $a$ in $\mathbb{Z}_p$ into a share of $a$ in $\mathbb{Z}_{p'}$, and $a$ is additively shared, i.e., $a := \sum_{i=0}^{m-1} x_i \mod p$. In this case, $\sum_{i=0}^{m-1} x_i \mod p' = qp + a \mod p' = (q \mod p')(p \mod p') + (a \mod p')$. Here, $p \mod p'$ can be computed from the public modulus so $q \mod p'$ is the only unknown value. Therefore, by computing $q$ using quotient transfer, we can obtain an efficient modulus-conversion protocol.

In this section, we first give a definition and instantiation of functionality we use in our modulus-conversion protocols. We then propose a special case of our modulus-conversion protocol from $\mathbb{Z}_2^u$ to $\mathbb{Z}_{p'}$. After that, we propose our modulus-conversion protocol from $\mathbb{Z}_p$ to $\mathbb{Z}_{p'}$.

### 5.1 Generating a pair of random shares

In our modulus-conversion protocol, we have to generate $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$ for $r \leftarrow \mathbb{Z}_2$. The functionality that should be realized by such a protocol is defined as $\mathcal{F}_{\text{doublerand}}$ described below. This can be instantiated with $O(\lceil \log p' \rceil)$ bits communication by combining a protocol generating $[r]^{\mathbb{Z}_p}$ ($\text{RAN}_2()$ in [9]) and our bit-decomposition protocol. We further give a more efficient and actively secure version of our modulus-conversion protocol for a number of parties in Appendix D.3.

**FUNCTIONALITY 5.1 ($\mathcal{F}_{\text{doublerand}}$ – Generating pair of random shares)**
Upon receiving $id$ from each party $P_i$, sample $r \leftarrow \mathbb{Z}_2$, generate $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$ via the sharing algorithms, and send $([r]_i^{\mathbb{Z}_2}, [r]_i^{\mathbb{Z}_{p'}})$ to each party $P_i$.

### 5.2 Modulus-conversion protocol from $\mathbb{Z}_2^u$ to $\mathbb{Z}_p$

We now give the formal description of Protocol 4, which is a special case of modulus conversion in which shares $[a]^{\mathbb{Z}_2^u}$ can be converted to $[a]^{\mathbb{Z}_{p'}}$.

**Protocol 4** modulus-conversion protocol from $\mathbb{Z}_2^u$ to $\mathbb{Z}_{p'}$

**Input:** $[a]^{\mathbb{Z}_2^u}$
**Output:** $[a]^{\mathbb{Z}_{p'}}$
1: **for** $0 \le i < u$ **do**
2: The parties call $\mathcal{F}_{\text{doublerand}}$ and receive $([r^{(i)}]^{\mathbb{Z}_2}, [r^{(i)}]^{\mathbb{Z}_{p'}})$.
3: The parties reveal $[a^{(i)} - r^{(i)}]^{\mathbb{Z}_2} = [a^{(i)}]^{\mathbb{Z}_2} - [r^{(i)}]^{\mathbb{Z}_2}$ to obtain $a^{(i)} - r^{(i)}$.
4: **if** $a^{(i)} - r^{(i)} = 0$ **then**
5: The parties set $[a^{(i)}]^{\mathbb{Z}_{p'}} = [r^{(i)}]^{\mathbb{Z}_{p'}}$.
6: **else**
7: The parties set $[a^{(i)}]^{\mathbb{Z}_{p'}} = (1 - [r^{(i)}]^{\mathbb{Z}_{p'}})$.
8: $[a]^{\mathbb{Z}_{p'}} := \sum_{i=0}^{u-1} 2^i [a^{(i)}]^{\mathbb{Z}_{p'}} \mod p'$.
9: The parties output $[a]^{\mathbb{Z}_{p'}}$.

Protocol 4 consists of local operations, revealing, and $\mathcal{F}_{\text{doublerand}}$. Recall that we assume revealing is secure against active adversary. Therefore, Protocol 4 is also actively secure in the $\mathcal{F}_{\text{doublerand}}$-hybrid model.

The communication complexity is $u(\mathtt{drand}_{\mathbb{Z}_{p'}} + \mathtt{reveal}_{\mathbb{Z}_2})$, where $\mathtt{drand}_{\mathbb{Z}_{p'}}$ and $\mathtt{reveal}_{\mathbb{Z}_2}$ are the communication complexities of generating $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$ for $r \leftarrow \mathbb{Z}_2$ and revealing a share in $\mathbb{Z}_2$. If we regard the number of parties as a constant, it is $O(\log \lceil p' \rceil)$ bits. The round complexity is $drand + 1$, where $drand$ is that of generating $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$.

### 5.3 Equation for modulus conversion

Similarly to our bit-decomposition protocol, we first show a useful equation for our protocol.

**Theorem 5.2.** *Let $m, p, a, (x_0, \ldots, x_{m-1}), \tilde{p}$ be the same as Theorem 3.1, $p'$ is a prime number, and $\ell$ be a positive integer such that $\ell + u \leq |p|$. Then,*

$$a = 2^{-u}\left(\sum_{i=0}^{m-1} x_i - (\tilde{p}\sum_{i=0}^{m-1} x_i \mod 2^u)p\right) \mod p'.$$

[**Proof**] It directly follows from Theorem 3.1 and the fact that $2^u$ and $p'$ are co-prime.

$$\sum_{i=0}^{m-1} x_i - (\tilde{p}\sum_{i=0}^{m-1} x_i \mod 2^u)p = \sum_{i=0}^{m-1} x_i - qp = (2^u a + qp) - qp = 2^u a.$$

■

We obtain the following corollary when $p$ is a Mersenne prime.

**Corollary 5.3.** *Let $m, u, p, a, (x_0, \ldots, x_{m-1})$ be the same as Corollary 4.3. Let $r_i$ be the bits of $-x_i \mod 2^u$ smaller than $u$ bit and $a$ be an $\ell$-bit input and $\hat{a} := a2^\ell$. Then,*

$$a = 2^{-u}\left(\sum_{i=0}^{m-1} x_i - p(\sum_{i=0}^{m-1} r_i \mod 2^u)\right) \mod p'.$$

### 5.4 Our modulus-conversion protocol

In this subsection, we give *two* modulus-conversion protocols with a Mersenne prime $p$. The first protocol is passively secure, and the second one is actively secure if the components are actively secure, while the latter assumes the small number of parties due to the use of the replicated SS scheme. A protocol for a general prime appears at Appendix B.2.

The first protocol is as described in Protocol 5. The protocol uses Protocol 4 and share conversion from additive shares to a linear SS scheme. Protocol 5 is passively secure in the $(\mathcal{F}_{\text{clsum}}, \mathcal{F}_{\text{doublerand}})$-hybrid model since the protocol consists of sharing and $\mathcal{F}_{\text{clsum}}$, and Protocol 5 uses $\mathcal{F}_{\text{doublerand}}$.

**Protocol 5** Passively secure modulus-conversion protocol

**Input:** $[a]^{\mathbb{Z}_p}$
**Output:** $[a]^{\mathbb{Z}_{p'}}$

1: $P_i$ computes $x_i := 2^u \lambda_i [a]_i \mod p$ for $u = \lceil \log k \rceil$ and $0 \leq i < k$.
2: $\hat{x}_i := -x_i \mod 2^u$ and let the $j$-th bit of $\hat{x}_i$ be $\hat{x}_i^{(j)}$.
3: **for** $0 \leq i < k$ **do**
4: $\quad P_i$ shares $\hat{x}_i^{(0)}, \ldots, \hat{x}_i^{(u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[r_i]^{\mathbb{Z}_2^u}$.
5: The parties call $\mathcal{F}_{\text{clsum}}$ on input $[r_i]^{\mathbb{Z}_2^u}$ for $0 \leq i < k$, and regard the received value as $[q]^{\mathbb{Z}_2^u} := [\sum_{i=0}^{k-1} r_i]^{\mathbb{Z}_2^u}$.
6: The parties convert $[q]^{\mathbb{Z}_2^u}$ into $[q]^{\mathbb{Z}_{p'}}$ via Protocol 4.
7: $P_i$ computes $x_i := x_i \mod p'$ and shares $x_i$ via sharing algorithm of a linear SS scheme in $\mathbb{Z}_{p'}$ for $0 \leq i < k$.
8: The parties add the received shares as $[\sum_{i=0}^{k-1} x_i]^{\mathbb{Z}_{p'}} = \sum_{i=0}^{k-1} [x_i]^{\mathbb{Z}_{p'}}$.
9: The parties locally compute $[a]^{\mathbb{Z}_{p'}} := 2^{-u}([\sum_{i=0}^{k-1} x_i]^{\mathbb{Z}_{p'}} - p[q]^{\mathbb{Z}_{p'}}) \mod p'$.
10: Each $P_i$ outputs $[a]_i^{\mathbb{Z}_{p'}}$.

The second protocol is as described in Protocol 6. This protocol uses the same idea as our bit-decomposition protocol. We first convert $[a]^{\mathbb{Z}_p}$ into $[\![a]\!]^{\mathbb{Z}_p}$, and locally generate bit-wise shares. Protocol 6 is passively/actively secure in $(\mathcal{F}_{\text{clsum}}, \mathcal{F}_{\text{doublerand}}, \mathcal{F}_{\text{rand}})$-hybrid model since Protocol 1 and Protocol 4 use $\mathcal{F}_{\text{rand}}$ and $\mathcal{F}_{\text{doublerand}}$.

**Protocol 6** Modulus-conversion protocol for small number of parties

**Input:** $[a]^{\mathbb{Z}_p}$
**Output:** $[a]^{\mathbb{Z}_{p'}}$

1: The parties invoke Protocol 1 on input $[a]^{\mathbb{Z}_p}$ and receive $[\![a]\!]^{\mathbb{Z}_p}$, where $m = \binom{n}{k-1}$ and $a = \sum_{i=0}^{m-1} x_i \mod p$.
2: The parties set $\hat{x}_i := -x_i \mod 2^u$ and let the $j$-th bit of $\hat{x}_i$ be $\hat{x}_i^{(j)}$.
3: The parties obtain $[\![\hat{x}_i]\!]^{\mathbb{Z}_2^u}$ for $0 \le i < m$ by Algorithm 3.
4: The parties call $\mathcal{F}_{\text{clsum}}$ on input $[\![r_i]\!]^{\mathbb{Z}_2^u}$ for $0 \le i < m$, and regard the received value as $[\![q]\!]^{\mathbb{Z}_2^u} := [\![\sum_{i=0}^{m-1} r_i]\!]^{\mathbb{Z}_2^u}$.
5: The parties convert $[q]^{\mathbb{Z}_2^u}$ into $[q]^{\mathbb{Z}_{p'}}$ via Protocol 4.
6: The parties locally compute $x_j := x_j \mod p'$ for all their own sub-shares, and regard them as $[\![\sum_{i=0}^{m-1} x_i]\!]^{\mathbb{Z}_{p'}}$.
7: The parties compute $[\![a]\!]^{\mathbb{Z}_{p'}} := 2^{-u}([\![\sum_{i=0}^{m-1} x_i]\!]^{\mathbb{Z}_{p'}} - p[\![q]\!]^{\mathbb{Z}_{p'}}) \mod p'$.
8: The parties locally convert $[\![a]\!]^{\mathbb{Z}_{p'}}$ into $[a]^{\mathbb{Z}_{p'}}$.
9: The parties output $[a]^{\mathbb{Z}_{p'}}$.

### 5.5 Efficiency

The communication complexity of Protocol 5 is $u\ \mathtt{share}_{\mathbb{Z}_2} + \mathtt{clsum}_{u,k} + u(\mathtt{drand}_{\mathbb{Z}_{p'}} + \mathtt{reveal}_{\mathbb{Z}_2}) + k\ \mathtt{share}_{\mathbb{Z}_{p'}}$. If the number of parties is small and regarded as a constant, the communication complexity of $u\ \mathtt{share}_{\mathbb{Z}_2}$, $\mathtt{clsum}_{u,k}$, and $\mathtt{reveal}_{\mathbb{Z}_2}$ are $O(1)$, $\mathtt{drand}_{\mathbb{Z}_{p'}}$ is $O(\lceil \log p' \rceil)$, and $\mathtt{share}_{\mathbb{Z}_{p'}}$ is $O(\lceil \log p' \rceil)$, respectively. Therefore, the total communication complexity is $O(\lceil \log p' \rceil)$.

The communication complexity of Protocol 6 is $\mathtt{toRep}_{\mathbb{Z}_p} + \mathtt{clsum}_{u,m} + u(\mathtt{drand}_{\mathbb{Z}_{p'}} + \mathtt{reveal}_{\mathbb{Z}_2})$, where $\mathtt{toRep}_{\mathbb{Z}_p}$ is that of Protocol 1. If the number of parties is regarded as a constant, the total communication complexity is $O(\lceil \log p \rceil + \lceil \log p' \rceil)$ due to $\mathtt{toRep}_{\mathbb{Z}_p}$. However, if $p' > p$, Protocol 6 can be more efficient than Protocol 5. The number of rounds is $(rand + 1) + 1 + (1 + drand) + 1 = 4 + rand + drand$, where $rand$ is the number of rounds to instantiate $\mathcal{F}_{\text{rand}}$.

## 6 Experiments

We implemented our bit-decomposition and modulus-conversion protocols and compare their efficiency with existing results. As we stated, to the best of our knowledge, there is no bit-decomposition protocol in which $\ell + u < \lceil \log p \rceil$ is assumed and which outputs $[a]^{\mathbb{Z}_2^\ell}$. Therefore, our bit-decomposition protocols are formally incomparable to existing ones. In this paper, we compare experimental results with those of [4] as reference, since it is the most efficient bit-decomposition protocol. We implemented our bit-decomposition protocol with optimizations described in Appendix D. These optimizations only affect the constant factor of the communication complexity.

The details of the machines and network environments used in the experiment are as follows. Each machine had an Intel® Core™ i7 6900K 3.2 GHz × 8 cores. For a gigabit network, we used Intel® I218-LM star network via an L2 Gigabit hub. The ping latency was 0.19 ms.

The experimental results are listed in Table 1. It shows the experimental result of passively secure bit-decomposition protocols in a gigabit network. We measured the processing time of the bit-decomposition protocol of $10^7$ $\ell$-bit elements. To align the setting to [4]. we used $(k, n) = (2, 3)$. Our protocol uses $p = 2^{61} - 1$, and $\ell = 32$, 20, and 2. The setting of $\ell = 32$ is the same message space as [4], while $\ell = 20$ and 2 are favorable for our bit-decomposition protocol. The input and output of our protocol were shares of Shamir's scheme, while those of [4] were shares of the replicated SS scheme.

As shown in Table 1, our bit-decomposition protocol achieves higher performance than that of [4]. Further experiments including modular-conversion protocols appear in Appendix E.

| | Modulus ($p$) | Bit-length of secret ($\ell$) | Processing time (ms) |
|---|---|---|---|
| [4] | $2^{32}$ | 32 | 200,000 |
| Our bit-decomposition protocol | $2^{61}-1$ | 32 | 1,194 |
| | $2^{61}-1$ | 20 | 759 |
| | $2^{61}-1$ | 2 | 123 |

**Table 1.** Processing time (ms) for $10^7$ records in passively secure bit-decomposition protocols in Gigabit network

## 7 Conclusion

We proposed secret-sharing-based bit-decomposition and modulus-conversion protocols for $\mathbb{Z}_p$ with an honest majority. Our protocols are secure against passive and active adversaries depending on the components of our protocols. If $k$ and $n$ are small, the communication complexity of our passively secure bit-decomposition and modulus conversion protocols are $O(\ell)$ bits and $O(\lceil \log p' \rceil)$ bits. While some settings are different from existing works, the communication complexity is smaller than the current best result [4]. Furthermore, we also confirmed with the experimental results that our protocols are highly efficient.

## References


1. T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS*, pages 805–817. ACM, 2016.
2. A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, 1996.
3. D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht. Privacy-preserving tax fraud detection in the cloud with realistic data volumes. Cybernetica research report, 2016.
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. J. W. Bos, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. Efficient SIMD arithmetic modulo a mersenne number. In E. Antelo, D. Hough, and P. Ienne, editors, *20th IEEE Symposium on Computer Arithmetic, ARITH 2011, Tübingen, Germany, 25-27 July 2011*, pages 213–221. IEEE Computer Society, 2011.
6. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.
8. 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.
9. 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 S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.
10. S. L. From and T. Jakobsen. *Secure Multi-Party Computation on Integers*. PhD thesis, University of Aarhus, 2006.
11. J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT 2017*, pages 225–255, 2017.
12. D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer. Circuits resilient to additive attacks with applications to secure computation. In D. B. Shmoys, editor, *STOC*, pages 495–504. ACM, 2014.
13. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
14. S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005.
15. D. Ikarashi, R. Kikuchi, K. Hamada, and K. Chida. Actively private and correct MPC scheme in $t < n/2$ from passively secure schemes with small overhead. *IACR Cryptology ePrint Archive*, 2014:304, 2014.
16. M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *IEICE Transactions*, 72:56–64, 1989.

17. E. Kimura, K. Hamada, R. Kikuchi, K. Chida, K. Okamoto, S. Manabe, T. Kuroda, Y. Matsumura, T. Takeda, and N. Mihara. Evaluation of secure computation in a distributed healthcare setting. In *Proceedings of MIE2016 at HEC2016*, pages 152–156, 2016.
18. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.
19. Y. Lindell and A. Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 259–276. ACM, 2017.
20. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Y. Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.
21. T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Public Key Cryptography*, pages 343–360, 2007.
22. T. I. Reistad and T. Toft. Linear, constant-rounds bit-decomposition. In *Information, Security and Cryptology - ICISC 2009*, pages 245–257, 2009.
23. B. Schoenmakers and P. Tuyls. Efficient binary conversion for paillier encrypted values. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537. Springer, 2006.
24. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
25. T. Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In *Topics in Cryptology - CT-RSA 2009*, pages 357–371, 2009.

# A Definition of Security

## A.1 Passive security

Most of the contents in this section are taken verbatim from [1, Section 3.1].

A protocol is secure in the presence of $k-1$ corrupted parties if the view of the corrupted party in a real protocol execution can be generated by a simulator given only the corrupted parties input and output. The view of $P_i$ during an execution of a protocol $\pi$ on inputs $\overrightarrow{x}$, denoted $\textsc{View}_i^{\pi}(\overrightarrow{x})$, consists of its input $x_i$, its internal random coins $r_i$ and the messages that were received by $P_i$ in the execution. The output of all parties from an execution of $\pi$ is denoted by $\textsc{Output}^{\pi}(\overrightarrow{x})$. Let $I \subset \{0,\ldots,n-1\}$ be the set of indices of the corrupted parties controlled by the adversary.

**Definition A.1.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a probabilistic $n$-ary functionality and let $\pi$ be a protocol. We say that $\pi$ computes $f$ with perfect security in the presence of $k-1$ passive corrupted party for $f$ if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every $k-1$ corrupted parties $P_i$ such that $i \in I$ and $|I| = k-1$, and every $\overrightarrow{x} \in (\{0,1\})^n$, where $|x_0| = |x_1| = \ldots = |x_{n-1}|$:*

$$\{(\mathcal{S}(x_I, f_I(\overrightarrow{x})), f(\overrightarrow{x}))\} \equiv \{(\textsc{View}_I^{\pi}(\overrightarrow{x}), \textsc{Output}^{\pi}(\overrightarrow{x}))\} \tag{5}$$

If Equation (5) holds with perfect/computational indistinguishability, then we say that $\pi$ computes $f$ with perfect/computational security in the presence of $k-1$ passive corrupted parties.

## A.2 Active security

Most of the contents in this section are taken verbatim from [19, Appendix A].

The security parameter is denoted $\kappa$; negligible functions and computational indistinguishability are defined in the standard way, with respect to non-uniform polynomial-time distinguishers.

*Ideal versus real model definition.* We use the ideal/real simulation paradigm in order to define security, where an execution in the real world is compared to an execution in the ideal world where an incorruptible trusted party computes the functionality for the parties [6,13]. We define *security with abort* (and without fairness), meaning that the corrupted parties may receive output while the honest parties do not. Our definition does *not* guarantee *unanimous abort*, meaning that some honest party may receive output while the other does not. It is easy to modify our protocols so that the honest parties unanimously abort by running a single (weak) Byzantine agreement at the end of the execution [14]; we therefore omit this step for simplicity.

*The real model.* In the real model, an $n$-party protocol $\pi$ is executed by the parties. For simplicity, we consider a synchronous network that proceeds in rounds and a rushing adversary, meaning that the adversary receives its incoming messages in a round before it sends its outgoing message. The adversary $\mathcal{A}$ can be active; it sends all messages in place of the corrupted parties, and can follow any arbitrary strategy. The honest parties follow the instructions of the protocol.

Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary controlling $k-1 < \frac{n}{2}$ parties. Let $Real_{\pi,\mathcal{A}(z),I}(x_0,\ldots,x_{n-1},\kappa)$ denote the output of the honest parties and $\mathcal{A}$ in the real execution of $\pi$, with inputs $x_0,\ldots,x_{n-1}$, auxiliary-input $z$ for $\mathcal{A}$, and security parameter $\kappa$.

*The ideal model.* We define the ideal model, for any (possibly reactive) functionality $\mathcal{F}$, receiving inputs from $P_0,\ldots,P_{n-1}$ and providing them with outputs. Let $I \subset \{0,\ldots,n-1\}$ be the set of indices of the corrupted parties controlled by the adversary. The ideal execution proceeds as follows:

- **Send inputs to the trusted party:** Each honest party $P_j$ sends its specified input $x_j$ to the trusted party. A corrupted party $P_i$ controlled by the adversary may either send its specified input $x_i$, some other $x_i'$ or an `abort` message.
- **Early abort option:** If the trusted party received `abort` from the adversary $\mathcal{A}$, it sends $\perp$ to all parties and terminates. Otherwise, it proceeds to the next step.
- **Trusted party sends output to the adversary:** The trusted party computes each party's output as specified by the functionality $\mathcal{F}$ based on the inputs received; denote the output of $P_j$ by $y_j$. The trusted party then sends $\{y_i\}_{i\in I}$ to the corrupted parties.
- **Adversary instructs trusted party to continue or halt:** For each $j \in \{0,\ldots,n-1\}$ with $j \notin I$, the adversary sends the trusted party either $\texttt{abort}_j$ or $\texttt{continue}_j$. For each $j \notin I$:
  - If the trusted party received $\texttt{abort}_j$ then it sends $P_j$ the abort value $\perp$ for output.
  - If the trusted party received $\texttt{continue}_j$ then it sends $P_j$ its output value $y_j$.
- **Outputs:** The honest parties always output the output value they obtained from the trusted party, and the corrupted parties outputs nothing.

Let $\mathcal{S}$ be a non-uniform probabilistic polynomial-time adversary controlling parties $P_i$ for $i \in I$. Let $Ideal_{\mathcal{F},\mathcal{S}(z),I}(x_0,\ldots,x_{n-1},\kappa)$ denote the output of the honest parties and $\mathcal{S}$ in an ideal execution with the functionality $\mathcal{F}$, inputs $x_0,\ldots,x_{n-1}$ to the parties, auxiliary-input $z$ to $\mathcal{S}$, and security parameter $\kappa$.

*Security.* Informally speaking, the definition says that protocol $\pi$ securely computes $f$ if adversaries in the ideal world can simulate executions of the real world protocol. In some of our protocols there is a statistical error that is not dependent on the computational security parameter. As in [20], we formalize security in this model by saying that the distinguisher can distinguish with probability at most this error *plus* some factor that is negligible in the security parameter. This is formally different from the standard definition of security since the statistical error does not decrease as the security parameter increases.

**Definition A.2.** *Let $\mathcal{F}$ be a $n$-party functionality, and let $\pi$ be a $n$-party protocol. We say that $\pi$* securely computes *$f$* with abort in the presence of an adversary controlling $k-1 < \frac{n}{2}$ parties*, if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world, there exists a non-uniform probabilistic polynomial-time simulator/adversary $\mathcal{S}$ in the ideal model with $\mathcal{F}$, such that for every $I \subset \{0,\ldots,n-1\}$,*

$$\left\{Ideal_{F,\mathcal{S}(z),I}(x_1,\ldots,x_n,\kappa)\right\} \overset{c}{\approx} \left\{Real_{\pi,\mathcal{A}(z),I}(x_1,\ldots,x_n,\kappa)\right\}$$

*where $x_0,\ldots,x_{n-1} \in \mathbb{F}^*$ under the constraint that $|x_1| = \cdots = |x_n|$, $z \in \mathbb{F}^*$ and $\kappa \in \mathcal{N}$. We say that $\pi$ securely computes $f$ with abort in the presence of one active party* with statistical error *$2^{-\sigma}$ if there exists a negligible function $\mu(\cdot)$ such that the distinguishing probability of the adversary is less than $2^{-\sigma}+\mu(\kappa)$.*

*The hybrid model.* We prove the security of our protocols in a hybrid model, where parties run a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition theorem of [6] states that one can replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. When the subfunctionality is $g$, we say that the protocol works in the *$g$-hybrid model.*

*Universal Composability [7].* Protocols that are proven secure in the universal composability framework have the property that they maintain their security when run in parallel and concurrently with other secure and insecure protocols. In [18, Theorem 1.5], it was shown that any protocol that is proven secure with a black-box non-rewinding simulator and also has the property that the inputs of all parties are fixed before the execution begins (called input availability or start synchronization in [18]), is also secure under universal composability. Since the input availability property holds for all of our protocols and subprotocols, it is sufficient to prove security in the classic stand-alone setting and automatically derive universal composability from [18]. We remark that this also enables us to call the protocol and subprotocols that we use in parallel and concurrently (and not just sequentially), enabling us to achieve more efficient computation (e.g., by running many executions in parallel or running each layer of a circuit in parallel).

# B Protocols for a general prime number $p$

## B.1 Bit-decomposition protocol

Let $z', \tilde{p}$ be the same as Theorem 4.1. Notice that we can compute $[ab]^{\mathbb{Z}_2^{\ell+\lceil \log p \rceil}}$ from $a \in \mathbb{Z}_p$ and $[b]^{\mathbb{Z}_2^\ell}$ by circuit evaluation.

**Protocol 7** Bit-decomposition protocol for a general prime

**Input:** $[a]^{\mathbb{Z}_p}$

**Output:** $[a]^{\mathbb{Z}_2^\ell}$

1: $P_i$ computes $x_i := 2^u \lambda_i [a]_i \mod p$ for $u = \lceil \log k \rceil$ and $0 \le i < k$, and let the $j$-th bit of $x_i$ be $x_i^{(j)}$.
2: **for** $0 \le i < k$ **do**
3: $P_i$ shares $x_i^{(0)}, \ldots, x_i^{(u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[r_i]^{\mathbb{Z}_2^u}$.
4: $P_i$ shares $x_i^{(u)}, \ldots, x_i^{(\ell+u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[q_i]^{\mathbb{Z}_2^\ell}$.
5: The parties call $\mathcal{F}_{\text{sum}}$ on input $[r_i]^{\mathbb{Z}_2^u}$ for $0 \le i < m$, and receive $[\sum_{i=0}^{m-1} r_i]^{\mathbb{Z}_2^{2u}}$. ($m$ additions yield $2u$-bit output)
6: The parties regard the least $u$-bit of $[\sum_{i=0}^{m-1} r_i]^{\mathbb{Z}_2^{2u}}$ as $[r_u]^{\mathbb{Z}_2^u}$, and the others as $[q_u]^{\mathbb{Z}_2^u}$.
7: The parties compute $[z' r_u]^{\mathbb{Z}_2^{\ell+u}}$ and $[p r_u]^{\mathbb{Z}_2^{\ell+u}}$ from $z'$, $p'$, and $[r_u]^{\mathbb{Z}_2^u}$.
8: The parties regard the most significant $\ell$ bits of $[p r_u]^{\mathbb{Z}_2^{\ell+u}}$ as $[z]^{\mathbb{Z}_2^\ell}$.
9: The parties compute $[zp]^{\mathbb{Z}_2^{\ell+2u}}$ from $p \mod 2^{\ell+u}$ and $[z]^{\mathbb{Z}_2^u}$, and regard the least significant $\ell+u$ bits as $[zp]^{\mathbb{Z}_2^{\ell+u}}$.
10: The parties call $\mathcal{F}_{\text{clsum}}$ on input $[q_0]^{\mathbb{Z}_2^\ell}, \ldots, [q_{k-1}]^{\mathbb{Z}_2^\ell}$, $[q_u]^{\mathbb{Z}_2^u}$, $-[z' r_u]^{\mathbb{Z}_2^{\ell+u}}$, and $[zp]^{\mathbb{Z}_2^{\ell+2u}}$, and receive $[a]^{\mathbb{Z}_2^\ell} := [\sum_{i=0}^{m-1} q_i + q_u - z' r_u + zp]^{\mathbb{Z}_2^\ell}$.
11: Each $P_i$ outputs $[a]_i^{\mathbb{Z}_2^\ell}$.

## B.2 Modulus-conversion protocol

The difference is just $\hat{x}_i := -x_i \mod 2^u$ is replaced by $\hat{x}_i := \tilde{p} x_i \mod 2^u$.

# C Functionalities of circuit evaluation

In this section, we give the formal descriptions of $\mathcal{F}_{\text{sum}}$, $\mathcal{F}_{\text{clsum}}$, and $\mathcal{F}_{\text{zero}}$ as Functionalities C.1, C.2, and C.3, respectively.

# D Optimizations

In this section, we show several optimizations to improve the communication complexity. Some of optimizations use PRSS, and PRSS requires a pseudo-random function. Therefore, if we use those optimizations, the whole protocol will be computationally secure.

**Protocol 8** Modulus conversion protocol for a general prime $p$

**Input:** $[a]^{\mathbb{Z}_p}$
**Output:** $[a]^{\mathbb{Z}_{p'}}$

1: $P_i$ computes $x_i := 2^u \lambda_i [a]_i \mod p$ for $u = \lceil \log k \rceil$ and $0 \le i < k$
2: $\hat{x}_i := \tilde{p} x_i \mod 2^u$ and let the $j$-th bit of $\hat{x}_i$ be $\hat{x}_i^{(j)}$.
3: **for** $0 \le i < k$ **do**
4: $P_i$ shares $\hat{x}_i^{(0)}, \ldots, \hat{x}_i^{(u-1)}$ bit-by-bit in $\mathbb{Z}_2$, and the parties regard them as $[q]^{\mathbb{Z}_2^u}$.
5: The parties call $\mathcal{F}_{\mathrm{clsum}}$ on input $[r_i]^{\mathbb{Z}_2^u}$ for $0 \le i < k$, and regard the received value as $[q]^{\mathbb{Z}_2^u} := [\sum_{i=0}^{m-1} r_i]^{\mathbb{Z}_2^u}$.
6: The parties convert $[q]^{\mathbb{Z}_2^u}$ into $[q]^{\mathbb{Z}_{p'}}$ via Protocol 4.
7: $P_i$ computes $x_i := x_i \mod p'$ for $0 \le i < k$.
8: The parties call $\mathcal{F}_{\mathrm{2lin}}$ on inputs $x_i$ for $P_i$, $0 \le i < k$, and receives $[\sum_{i=0}^{k-1} x_i]^{\mathbb{Z}_{p'}}$.
9: The parties locally compute $[a]^{\mathbb{Z}_{p'}} := 2^{-u}([\sum_{i=0}^{k-1} x_i]^{\mathbb{Z}_{p'}} - p[q]^{\mathbb{Z}_{p'}}) \mod p'$.
10: Each $P_i$ outputs $[a]_i^{\mathbb{Z}_{p'}}$.

**FUNCTIONALITY C.1 ($\mathcal{F}_{\mathrm{sum}}$ – Summation circuit)**
Upon receiving $[a_0]_i^{\mathbb{Z}_2^\ell}, \ldots, [a_{m-1}]_i^{\mathbb{Z}_2^\ell}$ from each party $P_i$, check the correctness of shares and output $\perp$ if the shares are incorrect. If not, reconstruct the secrets, compute $a = \sum_{i=0}^{m-1} a_i$, generate $[a]^{\mathbb{Z}_2^{2\ell}}$ via the sharing algorithm, and send $[a]_i^{\mathbb{Z}_2^{2\ell}}$ to each party $P_i$.

First example of optimizations is to a multiplication protocol. If we use PRSS, a multiplication protocol of the replicated SS scheme requires 3 elements per invocation [15], while 6 elements without PRSS.

### D.1 Communication-efficient sharing for the replicated secret-sharing scheme

When $P_0$ tries to share $a$ in the setting of $(k, n) = (2, 3)$, a trivial way is to generate a share of $a$ and distribute it. In this trivial way, $P_0$ has to send 4 elements. Instead, there is a communication-efficient sharing that reduces the communication complexity by *half*. Let $(x_0, x_1)$, $(x_1, x_2)$, and $(x_2, x_0)$ be shares of $a$ for $P_0$, $P_1$, and $P_2$, respectively. $P_0$ sends $x_0 := a - r$ to $P_2$ and $x_1 := r$ to $P_1$ for a random number $r$, and $x_2$ is set as 0. In the above, $P_0$ sends only 2 elements. Furthermore, if $P_0$ and $P_1$ share a key of a pseudorandom function for PRSS, $r$ can be generated by the pseudorandom function, and $P_0$ has to send only a single element to $P_2$.

### D.2 Communication-efficient multiplication for locally generated shares

If we multiply three shares generated by the local share generation, we can multiply three shares with the same communication complexity as a *single* execution of a multiplication protocol. Let $r_0$, $r_1$, and $r_2$ be elements in $\mathbb{Z}_p$. Assume $P_0$ has $(r_0, 0), (0, r_1), (0, 0)$, $P_1$ has $(0, 0), (r_1, 0), (0, r_2)$, and $P_2$ has $(0, r_0), (0, 0), (r_2, 0)$, which are shares of $r_0$, $r_1$, and $r_2$, respectively. To multiply these three shares, the three parties proceed as follows.

1. $P_0$ chooses $s_0 \leftarrow \mathbb{Z}_p$, and sends $r_0 r_1 - s_0$ to $P_1$ and $s_0$ to $P_2$.
2. $P_0$, $P_1$, and $P_2$ regard $(s_0, r_0 r_1 - s_0)$, $(r_0 r_1 - s_0, 0)$, and $(0, s_0)$ as their shares of $r_0 r_1$, respectively.
3. $P_1$ chooses $s_1 \leftarrow \mathbb{Z}_p$, and sends $s_1$ to $P_0$ and $r_2(r_0 r_1 - s_0) - s_1$ to $P_1$.
4. $P_2$ chooses $s_2 \leftarrow \mathbb{Z}_p$, and sends $s_2$ to $P_1$ and $r_2 s_0 - s_2$ to $P_0$.
5. $P_0$, $P_1$, and $P_2$ regard $(r_2 s_0 - s_2, s_1)$, $(s_1, r_2(r_0 r_1 - s_0) - s_1 + s_2)$, and $(r_2(r_0 r_1 - s_0) - s_1 + s_2, r_2 s_0 - s_2)$ as their shares of $r_0 r_1 r_2$, respectively.

**FUNCTIONALITY C.2 ($\mathcal{F}_{\mathrm{clsum}}$ – Carryless-sum circuit)**
Upon receiving $[a_0]_i^{\mathbb{Z}_2^\ell}, \ldots, [a_{m-1}]_i^{\mathbb{Z}_2^\ell}$ from each party $P_i$, check the correctness of shares and output $\perp$ if the shares are incorrect. If not, reconstruct the secrets, compute $a = \sum_{i=0}^{m-1} a_i \mod 2^\ell$ (and discard bits after the $\ell$-th bit), generate $[a]^{\mathbb{Z}_2^\ell}$ via the sharing algorithm, and send $[a]_i^{\mathbb{Z}_2^\ell}$ to each party $P_i$.

**FUNCTIONALITY C.3 ($\mathcal{F}_{\mathbf{zero}}$ – Zero-test circuit)**
Upon receiving $[a_0]_i^{\mathbb{Z}_2}, \ldots, [a_{m-1}]_i^{\mathbb{Z}_2}$ from each party $P_i$, check the correctness of shares and output $\perp$ if the shares are incorrect. If not, reconstruct the secrets, compute $a = \prod_{i=0}^{m-1}(1 - a_i)$, generate $[a]^{\mathbb{Z}_2}$ via the sharing algorithm, and send $[a]_i^{\mathbb{Z}_2^\ell}$ to each party $P_i$.

In the above, 6 elements are sent among the parties, which is the communication complexity of a multiplication protocol. If each pair of the parties shares a key of a pseudorandom function for PRSS, $s_0, s_1$, and $s_2$ can be generated by the pseudorandom function and 3 elements are sent among the parties, which is also the same communication complexity of a multiplication protocol with PRSS.

### D.3 Efficient generation of a pair of random shares for small number of parties

Protocol 9 shows an efficient protocol to generate $[r]^{\mathbb{Z}_2}$ and $[r]^{\mathbb{Z}_{p'}}$ for $r \leftarrow \mathbb{Z}_2$ for a small number of parties. Here, $\mathcal{F}_{\text{xor}}$ is the functionality of circuit evaluation that computes exclusive-or (addition in $\mathbb{Z}_2$) of the inputs. An XOR gate of two inputs in $\mathbb{Z}_p$ can be computed by a single multiplication as $r_0 + r_1 \mod 2 = r_0 + r_1 - r_0 r_1 \mod p$.

If we use PRSS, only a protocol instantiating $\mathcal{F}_{\text{xor}}$ requires communication. An ordinary circuit that computes an XOR gate sequentially requires $m$ multiplication. However, the optimization in Appendix D.2 can be used so the communication complexity is the same as $\lceil \frac{m}{3} \rceil$ invocations of a multiplication protocol.

**Protocol 9** Generating $[r]^{\mathbb{Z}_2}$ and $[r]^{\mathbb{Z}_{p'}}$ for small number of parties

**Input:** nothing
**Output:** $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$
1: The parties obtain $\mathcal{F}_{\text{rand}}$ and receive $[\![r]\!]^{\mathbb{Z}_2}$, where the sub-shares are $r_0, \ldots, r_{m-1}$, i.e., $r = \sum_{i=0}^{m-1} r_i \mod 2$.
2: The parties use Algorithm 3 and obtain $[\![r_0]\!]^{\mathbb{Z}_p}, \ldots, [\![r_{m-1}]\!]^{\mathbb{Z}_p}$.
3: The parties call $\mathcal{F}_{\text{xor}}$ on input $[\![r_0]\!]^{\mathbb{Z}_p}, \ldots, [\![r_{m-1}]\!]^{\mathbb{Z}_p}$, and receives $[\![r]\!]^{\mathbb{Z}_p} = [\![\sum_{i=0}^{m-1} x_i \mod 2]\!]$.
4: The parties convert $([\![r]\!]^{\mathbb{Z}_2}, [\![r]\!]^{\mathbb{Z}_{p'}})$ into $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$ using the local conversion technique.
5: The parties output $([r]^{\mathbb{Z}_2}, [r]^{\mathbb{Z}_{p'}})$.

## E Experimental results in various settings

Tables 2, 3, and 4 list the experimental results of our both protocols in 10G, 1G, and (simulated) Internet connections. Machine environment is the same as Section 6. For any experiment, $(k, n) = (2, 3)$ and $10^7$ records are decomposed/converted. For a 10 Gigabit network, we used Intel® X550T 10G-Ether ring network with 0.055 ms of ping (roundtrip) latency. For Internet simulation, we limit the network connection by 50 Mbps with 20 ms of roundtrip latency. For the experiment of an actively secure protocol, we use the technique of [15] to securely compute circuit evaluation with statistical error $\kappa = 8$. We implemented Protocol 6 as a modulus-conversion protocol, and set $p = 2^{61} - 1$ and $p' = 2^{127} - 1$.

| | Bit-length ($\ell$) | Passive security | Active security with abort |
|---|---|---|---|
| Bit-decomposition | 32 | 364 | 5334 |
| | 20 | 234 | 3446 |
| | 2 | 62 | 761 |
| Modulus-conversion | - | 1,684 | 4,094 |

**Table 2.** Processing time (ms) in 10G network

| | Bit-length ($\ell$) | Passive security | Active security with abort |
|---|---|---|---|
| Bit-decomposition | 32 | 1,194 | 1,878 |
| | 20 | 759 | 12,258 |
| | 2 | 123 | 486 |
| Modulus-conversion | - | 6,189 | 15,298 |

**Table 3.** Processing time (ms) in Gigabit network

| | Bit-length ($\ell$) | Passive security | Active security with abort |
|---|---|---|---|
| Bit-decomposition | 32 | 13,572 | 286,812 |
| | 20 | 8,538 | 184,108 |
| | 2 | 977 | 36,684 |
| Modulus-conversion | - | 73,162 | 234,805 |

**Table 4.** Processing time (ms) in (simulated) Internet