PAPER *Special Section on Discrete Mathematics and Its Applications*

Multi-Party Computation for Modular Exponentiation Based on Replicated Secret Sharing

Kazuma OHARA[†,††a)], ***Nonmember***, **Yohei WATANABE**[††,†††*b)], **Mitsugu IWAMOTO**[††c)], ***Members***, ***and*** **Kazuo OHTA**[††d)], ***Fellow***

SUMMARY In recent years, multi-party computation (MPC) frameworks based on replicated secret sharing schemes (RSSS) have attracted the attention as a method to achieve high efficiency among known MPCs. However, the RSSS-based MPCs are still inefficient for several heavy computations like algebraic operations, as they require a large amount and number of communication proportional to the number of multiplications in the operations (which is not the case with other secret sharing-based MPCs). In this paper, we propose RSSS-based three-party computation protocols for modular exponentiation, which is one of the most popular algebraic operations, on the case where the base is public and the exponent is private. Our proposed schemes are simple and efficient in both of the asymptotic and practical sense. On the asymptotic efficiency, the proposed schemes require $O(n)$-bit communication and $O(1)$ rounds,where n is the secret-value size, in the best setting, whereas the previous scheme requires $O(n^2)$-bit communication and $O(n)$ rounds. On the practical efficiency, we show the performance of our protocol by experiments on the scenario for distributed signatures, which is useful for secure key management on the distributed environment (e.g., distributed ledgers). As one of the cases, our implementation performs a modular exponentiation on a 3,072-bit discrete-log group and 256-bit exponent with roughly 300ms, which is an acceptable parameter for 128-bit security, even in the WAN setting.

key words: *multi-party computation, modular exponentiation, replicated secret sharing*

1. Introduction

1.1 Background

Secure multiparty computation (MPC) enables a set of parties to securely carry out a joint computation of their private inputs without revealing anything but the output. This strong cryptographic guarantee of confidentiality is particularly reliable for protecting confidential data from malicious insiders and malwares or for protecting multiple sources of confidential private data at their joint analysis. The insider threat is an annoying problem since some systems need a human operator to manage their data directly while the operator should not learn the contents of data. It is extremely hard to prevent malwares that exploit 0-day vulnerabilities from intruding systems and consequently stealing data. Combining multiple sources of private data can be effective for obtaining their deeper analysis, but it has a risk of allowing others to learn their private data. Nowadays, trustable operations or hardware security are responsible for mitigating all of these problems, but they have not been fully successful. MPC is the strong cryptographic technology that has a potential to solve these problems in an essential way. It can conceal the contents of data to operators who can operate this data themselves. It can manage data by several different systems with different 0-days so that the malware intruded into a single system cannot breach data. It can manage data by mutually independent systems so that an operator of any one system cannot learn data of others.

In the past few years, the efficiency of secure computation protocols has increased in leaps and bounds, we could expect realistic use of MPCs. In order to demonstrate this progress, it is sufficient to compare the first implementation in 2004 of a semi-honest two-party protocol based on Yao's garbled circuits [18] that computed at a rate of approximately 620 gates per second, to more recent work that processes at a rate of approximately 7 billion gates per second [3]. Of course, this amazing progress is achieved by not only algorithmic technique but also the advent of crypto hardware acceleration in the form of AES-NI and more. Notably, it has been revealed that the MPCs based on secret sharing schemes could achieve high throughput due to its less communication amount than garbled circuits [27], which shows the potentialities of MPCs to us.

In this paper, we focus on MPCs based on secret sharing schemes. Several remarkable works on MPC [2], [3], [7], [19], which can achieve high performance in terms of throughput, deploys *Replicated Secret Sharing* [8] (which constructs the access structure by replicating shares of n-out-of-n threshold scheme among multiple parties). Although this kind of secret sharing schemes has large share size compared with Shamir's secret sharing [24], these techniques show an effect on reducing communication amount of MPC-multiplication.

1.2 Motivation

Basically, when we want to process confidential information

Manuscript received September 25, 2018.
Manuscript revised January 25, 2019.

[†]The author is with the NEC corporation, Tokyo, 108-8001 Japan.
[††]The authors are with the University of Electro-Communications, Chofu-shi, 182-8585 Japan.
[†††]The author is with National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, 135-0064 Japan.
[*]Presently, the author is with National Institute of Information and Communications Technology (NICT).
a) E-mail: k-ohara@ax.jp.nec.com
b) E-mail: yohei.watanabe@nict.go.jp
c) E-mail: mitsugu@uec.ac.jp
d) E-mail: kazuo.ohta@uec.ac.jp

DOI: 10.1587/transfun.E102.A.1079

on distributed systems, MPC could be an answer. As one of notable example of such applications, we can consider distributed ledgers for cryptocurrency, as we know Blockchain. The protection of secret keys in cryptosystems is an critical issue in several systems, in particular distributed system. Since the authority managing secret keys could be a single point of failure, key management is a problem that plague system engineers. However, even today, the signing key of the Blockchain is often managed by depositing with trusted authority such as exchanges in many services, thus it cannot be said that it is managed securely enough.

One of the solutions against this issue, we can consider applying MPCs to compute digital signatures among the nodes of distributed ledger, while concealing its signing keys. These research are also probably best known as distributed signatures or threshold signatures [12], [17], [25]. The secret sharing-based MPCs like [2], [3], [7], [19] can construct distributed signature schemes, since these MPCs can compute any functions by composition of primitive MPC operations.

The MPCs based on RSSSs are generally efficient than other frameworks, but the cryptographic operations are still heavy even for these schemes. To handle cryptographic operations like digital signatures, we should design efficient MPC protocols for algebraic operations beyond the construction from primitive gates like addition/multiplication. In particular, modular exponentiation is one of the most important building blocks to construct various cryptographic tasks. Of course, the modular exponentiation is widely used in many fields including cryptography, and it should be provided as a basic instruction.

Therefore, in this paper, we focus on how to construct MPC for efficient modular exponentiation on RSSS-based MPC frameworks.

1.3 Contribution

In this paper, we propose an efficient MPC protocol for modular exponentiation with public base for MPC framework based on RSSS [2], [3], [7], [19]. These frameworks are known as the best practice for 3-party computation, but still unsuitable for cryptographic operations since these operations require much amount of multiplications.

Our proposed scheme is dedicated for modular exponentiation, which is based on the structure of secret sharing schemes deployed by the frameworks of [2], [3], [7], [19].

More precisely, previous MPCs for modular exponentiation based on RSSS require $O(n^2)$ communication complexity by processing square-and-multiply method on MPC, where n is the size of the secret in bits. On the other hand, the proposed schemes in this paper require $O(n)$ communication without deteriorating the order of round complexity. For more concrete comparison, see Sect. 6.1.

We will show several types of constructions, depending on the size of the modulus. First is the case where the modulus is power of 2, and second is the case for modulus is prime. The second variant can be more optimized if the modulus of exponent is small.

In addition, as an application of the proposed scheme, we consider the case of the distributed signatures, which generates signatures while concealing signing keys. We will show the experimental results assuming a scenario of distributed signatures. Not only in the asymptotic sense, the experiment shows that our scheme is efficient in practical setting. As one of the cases, our implementation performs a modular exponentiation on 3072-bit discrete log group and 256-bit exponent with roughly 300ms, which is an acceptable parameter for 128-bit security, even in WAN setting. In the simulation in a large parameter and WAN setting, the previous scheme takes roughly one hour for only network delay, whereas our protocol can perform roughly 170,000 ms, which is 130 times faster.

1.4 Organization

In Sect. 2, we introduce notations, definitions and building blocks used in this paper. In Sect. 3, we describe an important technique named as “local re-sharing of sub-shares” for RSSS-based MPCs, to describe our proposed scheme. Section 4 shows our proposal schemes for modular exponentiation. Section 6 discuss an application scenario of modular exponentiation MPC, and show the practical efficiency of our proposed scheme on the scenario by implementation results. Finally Sect. 7 concludes this paper.

2. Preliminaries

2.1 Notation

We assume all values we will handle are on $\mathbb{Z}_q$ for some positive integer q. Note that if $q = p$ where p is prime $\mathbb{Z}_p$ is a *field*, and if $q = 2^m$ where $m \in \mathbb{Z}$, $\mathbb{Z}_{2^m}$ is a *ring*.

Let $\mathcal{P}$ be a set of parties and $P_i \in \mathcal{P}$ be a party with the identifier i. In this paper the indices start from the number 0 for the parties and corresponding shares.

Here we let $[x]^q$ denote that $x \in \mathbb{Z}_q$ is shared by a certain secret sharing scheme over $\mathbb{Z}_q$, and $[x]^q_i$ be a share of x for the party P_i. If $[x]^q_i$ is a tuple of t elements, we denote $[x]^q_{i,j}$ be the j-th element of $[x]^q_i$ where $j \in \{0, \ldots, t-1\}$. Let $x|_j$ denotes the j-th least significant bit of the binary expression of x, and $[x]^q_i|_j$ denote the j-th bits of all elements of the share, namely $([x]^q_{i,1}|_j, \ldots, [x]^q_{i,t}|_j)$.

2.2 Secret Sharing

We employ the 2-out-of-3 secret sharing of replicated type described in [8]. In this paper, we follow the RSSS used in Araki et al.'s scheme [3][†].

[†]Araki et al.'s definition of the RSSS scheme is not exactly the same as the original one in [8] but essentially the same, and such a variant was referred as RSSS in several subsequent papers [1], [11], [19]. In this paper, we simply call the variant the RSSS scheme in these papers. We stress that our proposed schemes are applicable even to the original RSSS of [8].

P_0	$[x]^q_0 = ([x]^q_{0,0}, [x]^q_{0,1}) = (x_2{+}x_0, x_0)$
P_1	$[x]^q_1 = ([x]^q_{1,0}, [x]^q_{1,1}) = (x_0{+}x_1, x_1)$
P_2	$[x]^q_2 = ([x]^q_{1,0}, [x]^q_{1,1}) = (x_1{+}x_2, x_2)$

Fig. 1 Reference for the form of 2-out-of-3 replicated secret sharing.

Definition 1: A 2-out-of-3 replicated secret sharing scheme (2-out-of-3 RSSS) [3] is a set of the following two probabilistic algorithms Share and Reconst. We additionally let all indices corresponding to the index space $\{0, 1, 2\}$ are described over modulus 3 and hereafter we omit the description of "mod 3". For example, a certain value x_i indexed by $i \in \{0, 1, 2\}$, x_3 is handled as x_0, and x_{-1} is handled as x_2.

Share: Given a specification of $\mathbb{Z}_q$, an element of $\mathbb{Z}_q$ $x \in \mathbb{Z}_q$, and a random number r, as $(\mathbb{Z}_q, x, r)$, the algorithm Share generates random elements $x_0, x_1, x_2 \in \mathbb{Z}_q$ under the condition of $x_0 + x_1 + x_2 = x$, generates a share of P_i denoted by $[x]^q_i$ as $(x_{i-1} + x_i, x_i)$ for $i \in \{0, 1, 2\}$, and output a set of all shares $[x]^q$. Then, $([x]^q_{i,0}, [x]^q_{i,1}) = (x_{i-1} + x_i, x_i)$.

Reconst: Given $(i, [x]^q_i, [x]^q_{i+1})$ for $i \in \{0, 1, 2\}$, the algorithm Reconst outputs $x = [x]^q_{i,0} + [x]^q_{i+1,1}$ (for arbitrary $i \in \{0, 1, 2\}$).

As is mentioned in [3], we can easily see that this scheme satisfies the following *correctness* and *secrecy* requirements.

Correctness: For any $q \in \mathbb{N} \setminus \{0\}$, $x \in \mathbb{Z}_q$, $r \in \mathbb{Z}_q$, $([x]^q_0, [x]^q_1, [x]^q_2) \leftarrow \mathsf{Share}(\mathbb{Z}_q, x, r)$ and any $i \in \{0, 1, 2\}$, x is recovered by $\mathsf{Reconst}(i, [x]^q_i, [x]^q_{i+1})$ with probability 1.

Secrecy: For any $q \in \mathbb{N} \setminus \{0\}$, $x \in \mathbb{Z}_q$, $r \in \mathbb{Z}_q$, $([x]^q_0, [x]^q_1, [x]^q_2) \leftarrow \mathsf{Share}(\mathbb{Z}_q, x, r)$, no information of x is leaked from any single party's share $[x]^q_i$ where $i \in \{0, 1, 2\}$.

Figure 1 shows a quick reference for the form of the shares in Definition 1. The row indexed by "P_i" represents that the form of the share which is held by P_i.

2.3 A Model of Secure Computation

In this section, we describe the model of MPC in this paper.

(1) Definition of 3-party computation

Here we consider 3-party computation (3PC). Namely, the parties are P_0, P_1 and P_2.

A multi-party protocol is specified by a (possibly probabilistic) procedure referred to as *functionality*. Denote $f : (\{0, 1\}^*)^3 \to (\{0, 1\}^*)^3$ as the 3-ary functionality[†]. Specifically, $f = (f_0, f_1, f_2)$ and each party P_i can obtain distinct outputs $f_i(\vec{x})$ in general.

(2) Definition of Security

The goal of MPC protocol based on secret sharing is to compute shares of outputs from shares of inputs without revealing information on the input anything but the output.

The security of MPC is formalized by simulation-based security. Namely, if there exist simulators who can generate the view of each party in the execution from given inputs and outputs, the MPC protocol is secure. This formalization implies that the parties learn nothing about inputs from the execution of the protocol, except for the information derived from outputs.

Definition 2: Let $A = \{A_i\}_{i \in \{0,1\}^*; n \in \mathbb{N}}$ and $B = \{B_i\}_{i \in \{0,1\}^*; n \in N}$ be probability ensembles indexed by $i \in \{0, 1\}$ and $n \in \mathbb{N}$. Let $\kappa \in \mathbb{N}$ be a security parameter. We say that A and B are computationally indistinguishable, denoted by $\{A_i\}_\kappa \simeq \{B_i\}_\kappa$ (or simply $A \simeq B$), if for every non-uniform polynomial-time algorithm $\mathcal{D}$ there exists a function $p(\cdot)$ such that for every $i \in \{0, 1\}^*$ and every $\kappa \in \mathbb{N}$,

$$|\Pr[\mathcal{D}(A_i) = 1] - \Pr[\mathcal{D}(B_i) = 1]| \leq \frac{1}{p(\kappa)}.$$

Definition 3: Let $f : (\{0, 1\}^*)^3 \to (\{0, 1\}^*)^3$ be a 3-ary functionality, and κ be a security parameter. Let $\vec{x} = (x_0, x_1, x_2)$ be a vector of inputs. Let $\mathsf{view}^\pi_i(\vec{x})$ be the view of the party P_i during an execution of a protocol π on input $\vec{x}$. Let $\mathsf{output}^\pi(\vec{x})$ be the output of all parties from an execution of π. We say that a protocol π *privately computes f in the presence of semi-honest adversaries*, if it is correct and for every $\vec{x} \in (\{0, 1\}^*)^3$ the following properties hold: (1) $\mathsf{output}^\pi(\vec{x}) = f(\vec{x})$ and (2) there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every corrupted party P_i ($i \in \{0, 1, 2\}$) and every $\vec{x} \in (\{0, 1\}^*)^3$ where $|x_0| = |x_1| = |x_2|$:

$$\{\mathcal{S}(x_i, f_i(\vec{x}))\}_{\vec{x},\kappa} \simeq \left\{\mathsf{view}^\pi_i(\vec{x}))\right\}_{\vec{x},\kappa},$$

In addition, we say that π privately computes f in the presence of semi-honest adversaries *in the $\mathcal{F}$-hybrid model* if π contains ideal calls to a trusted party computing a certain functionality $\mathcal{F}$.

Note that the functionalities with only local computation (i.e. no communication among parties) obviously satisfy the above definition, since the view of such functionality is only the information that can be obtained from shares of secret sharing schemes. Such a view leaks no information about inputs due to the security of secret sharing.

(3) Representation of functionalities for secret sharing-based 3PC

For each MPC operation, each party receives the operation

[†]We can also apply the "client-server" model where the parties running the MPC protocol are servers who receive the input shares of multiple clients and compute the output for them. The client-server model for MPC is introduced by Cybernetica in their Sharemind product [4].

code representing a functionality and its input as shares. Note that the operations to be computed are public for every party. Here we call the representation of a functionality as "opcodes". The whole computation among 3 parties is represented by a sequence of such opcodes and its input shares. For each opcode given the parties, they invoke the function corresponding to the opcodes. In the process of this function, the parties communicate with each other as necessary.

For example, we consider the case of MPC for addition represented by the opcode add. When the parties starts MPC-addition with shared input x and y, each party P_i ($i \in \{0,1,2\}$) takes $(\mathsf{add}, [x]^q_i, [y]^q_i)$ as inputs, and invoke corresponding function $\mathsf{add}([x]^q_i, [y]^q_i)$ then get $[z]^q_i$ where $z = x + y$ (see also definition in Sect. 2.4). To simplify the notation, we describe the opcodes and their corresponding functions by the same name.

2.4 3PC for Arithmetic Operations Based on 2-Out-of-3 RSSS

In this section, we explain secure 3PC for addition and multiplication described in [3]. Semi-honest secure addition and multiplication for secret sharing in Definition 1 are given as follows.

Setup: When each P_i for $i \in \{0,1,2\}$ is given (setup, κ), P_i randomly generates $seed_i \in \{0,1\}^\kappa$ and sends it to P_{i+1}. P_i stores $seed_i$ that is generated by him/herself and $seed_{i+1}$ that is received from P_{i-1}. In addition, the parties agree a pseudorandom function $\mathsf{PRF}^q : \{0,1\}^* \times \{0,1\}^\kappa \to \mathbb{Z}_q$[†].

For a public constant $c \in \mathbb{Z}_q$, the share of c is defined by $x_0 = c$ and $x_1 = x_2 = 0$. Namely, $[c]^q = ((c,0),(c,c),(0,0))$. It also can be performed during the execution of MPC operations.

Addition: When each P_i for $i \in \{0,1,2\}$ is given $(\mathsf{add}, [x]^q_i, [y]^q_i)$, all the parties do as follows:

1. For each $i \in \{0,1,2\}$, P_i generates

$$[z]^q_i := ([x]^q_{i,0} + [y]^q_{i,0}, [x]^q_{i,1} + [y]^q_{i,1})$$

and outputs $(\mathsf{add}, [z]^q_i)$.

This operation as the whole is denoted by $[z]^q = \mathsf{add}([x]^q, [y]^q)$.

We can easily check that $[z]^q$ is a valid share of $x + y$, since $[z]^q_{i,0} + [z]^q_{i+1} = ([x]^q_{i,0} + [y]^q_{i,0}) + ([x]^q_{i+1,1} + [y]^q_{i+1,1}) = ([x]^q_{i,0} + [x]^q_{i+1,1}) + ([y]^q_{i,0} + [y]^q_{i+1,1}) = x + y$ for all $i \in \{0,1,2\}$.

Multiplication:

When each P_i for $i \in \{0,1,2\}$ is given $(\mathsf{mult}, [x]^q_i, [y]^q_i)$, all the parties do as follows:

1. With a certain unique nonce vid[††], each party P_i computes $\eta_i = \mathsf{PRF}^q(vid, seed_i) - \mathsf{PRF}^q(vid, seed_{i-1})$ for $i \in \{0,1,2\}$. Note that $\sum_{i\in\{0,1,2\}} \eta_i = 0$, and P_i can compute η_i *without* interaction. In this paper, we call this η_i as *correlated randomness* (followed in [3]).
2. For each $i \in \{0,1,2\}$, P_i generates

$$w_i = [x]_{i,0} \cdot [y]_{i,0} - [x]_{i,1} \cdot [y]_{i,1} + \eta_i,$$

where "$\cdot$" is the multiplication over $\mathbb{Z}_q$, and sends $(\mathsf{mult\_msg}, w_i)$ to P_{i+1}.
3. For each $i \in \{0,1,2\}$, P_i generates

$$[z]^q_i := (w_{i-1} + w_i, w_{i-1})$$

and outputs $(\mathsf{mult}, [z]^q_i)$.

This operation as the whole is denoted by $[z]^q = \mathsf{mult}([x]^q, [y]^q, \eta_1, \eta_2, \eta_3)$.

For the correctness, we recall $x = x_0 + x_1 + x_2 \mod q$, $y = y_0 + y_1 + y_2 \mod q$ and $z = x \cdot y = (x_0 + x_1 + x_2)(y_0 + y_1 + y_2)$. w_i ($i \in \{0,1,2\}$) at Step 2 can be represented as $w_0 = x_2y_2 + x_2y_0 + x_0y_2 + \eta_0$, $w_1 = x_0y_0 + x_1y_0 + x_0y_1 + \eta_1$. $w_2 = x_1y_1 + x_2y_1 + x_1y_2 + \eta_2$, and we can see $z = w_0 + w_1 + w_2 \mod q$. Therefore, the share of Step 3 satisfies the form of RSSS described in Sect. 2.2.

Efficiency of the protocol: Basically, the biggest bottleneck of MPC is communication, since it is necessary to perform cooperative computation among parties. Therefore, the efficiency of MPC protocols is evaluated by two indices: the round complexity and the communication complexity. The round complexity means the number of communication among parties considering parallel execution. The communication complexity means total amount of data to be communicated. As can be easily seen, the addition of this MPC protocol requires no communication. On the other hand, to perform MPC for multiplication, this protocol only requires communication at Step 2 to send w_i to P_{i+1}. Here we denote the size of w_i is $n = \lceil \log q \rceil$. One invocation of mult requires $3n$-bit communication (n-bit per party) and 1 round. Note that generating η_i in mult requires no communication by sharing seeds for the hash function in setup phase.

Security: The proof of security for this protocol was given by [3]. We will discuss in more detail of the security in Sect. 5. In the following, we use the protocol of this section as secure building blocks (i.e., in a black-box way).

2.5 3PC Functionalities for Building Blocks

In this section, we explain several 3PC functionalities for

[†]In this paper, we deploy the computationally-secure variant of [3] for an efficient implementation. For an information-theoretically secure construction, the parties should exchange an n-bit random seed for every n-bit multiplication.

[††]In practice, the vid can be a counter that all parties locally increment at every call to PRF^q. The initial value of vid is agreed at the setup phase.

describing the MPC protocols in later sections.

The concrete evaluation of communication bits and the number of rounds relies on the complexity of the arithmetic operations introduced in Sect. 2.4. Set $n = \lceil \log q \rceil$.

2.5.1 Bit-Decomposition

The bit-decomposition operation is a protocol for converting a single integer share $[x]^q$ into n binary shares $[x|_{n-1}]^2, \ldots, [x|_0]^2$. Note that the outputs of the bit-decomposition are the shares on $\mathbb{Z}_2$.

Efficient decomposition protocols are introduced by [1], [19] for a ring (i.e., q is power of 2), and by [13] for a field (i.e., q is prime).

The concrete constructions for the bit-decomposition protocols [1], [13], [19] are closely related to our proposed scheme. In this section, we only introduce the interface for bit-decomposition, and review the construction in Sect. 3.

When every P_i for $i \in \{0, 1, 2\}$ is given (bit_decomp, $[x]^q_i$), the parties do the MPC for bit-decomposition and output $([x_{n-1}]^2_i, \ldots, [x_0]^2_i)$, which are the shares of binary representation of x over $\mathbb{Z}_2$.

This operation as the whole is denoted by $([x_{n-1}]^2, \ldots, [x_0]^2) = \mathsf{bit\_decomp}([x]^q)$.

Efficiency: The function maj requires 3-bit multiplication per one invocation. For each iteration in bit_decomp, maj is invoked twice. If we apply techniques in [1] or [19] for $\mathbb{Z}_2^m$ where $m \in \mathbb{Z}$, bit_decomp requires $6(n-1)$-bit communications and $(n-1)$-round complexity. If we apply the scheme in [13] for $\mathbb{Z}_p$ where p is prime, bit_decomp requires $(10n+4)$-bit[†] communications and $2(n-1)$-round complexity.

2.5.2 Bit-Addition

The bit-addition protocol is the addition operation over binary expression. Namely, this protocol takes shares of binary representations of two values $(x_{n-1}, \ldots, x_0)$ and $(y_{n-1}, \ldots, y_0)$ where $x = \sum_{i=0}^{n-1} 2^i x|_i$ and $y = \sum_{i=0}^{n-1} 2^i y|_i$, then output a share of the binary representation of $x + y$. In this paper, we only introduce the interface for bit-addition and use as a building block for other protocols.

When every P_i for $i \in \{0, 1, 2\}$ is given (bit_add, $([x|_{n-1}]^2_i, \ldots, [x|_0]^2_i), ([y|_{n-1}]^2_i, \ldots, [y|_0]^2_i)$), the parties do the MPC for bit-addition and output $([z|_{n-1}]^2_i, \ldots, [z|_0]^2_i)$ where $z = x + y$ and $z = \sum_{i=0}^{n-1} 2^i z|_i$.

This operation as the whole is denoted by $([z|_{n-1}]^2, \ldots, [z|_0]^2) = \mathsf{bit\_add}(([x|_{n-1}]^2, \ldots, [x|_0]^2), ([y|_{n-1}]^2, \ldots, [y|_0]^2))$.

Efficiency: Since the round and communication cost depend on the number of MPC-multiplication over $\mathbb{Z}_2$, the cost of bit_add follows standard construction of the full adder circuit. For example, if we deploy the well-known construction for ripple carry adder, bit_add takes $n-1$ rounds and $3(n-1)$ communication complexity for a ring [1], [19], and $2n-1$ rounds and $6(n-1)$ communication complexity for a field [13]. For other constructions, see also [1], [9], [23].

We recall $x = \sum_{i=0}^{n-1} 2^i \cdot x|_i$ where $x \in \mathbb{Z}_q$ and $x|_{n-1}, \ldots, x|_0 \in \mathbb{Z}_2$ (namely, $(x|_{n-1}, \ldots, x|_0)$ is a binary representation of x).

Let $[x]^q|_j$ denote $(([x]^q_0|_j), ([x]^q_1|_j), ([x]^q_2|_j)) = (([x]^q_{0,0}|_j, [x]^q_{0,1}|_j), ([x]^q_{1,0}|_j, [x]^q_{1,1}|_j), ([x]^q_{2,0}|_j, [x]^q_{2,1}|_j))$, and $([x|_j]^q)_{j=0,\ldots,n-1} := ([x|_{n-1}]^q, \ldots, [x|_0]^q)$. Set $n = \lceil \log q \rceil$.

2.5.3 Bit-Injection

The bit incetion protocol is a special instruction for converting a single binary share over $\mathbb{Z}_2$ into a single integer share over $\mathbb{Z}_q$ of the same value. In this paper, we only introduce the interface for bit-injection and use as a building block for other protocols.

When every P_i for $i \in \{0, 1, 2\}$ is given (bit_inject, $[x]^2_i$), where $x \in \mathbb{Z}_2$ and $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$ all parties do the MPC for bit-injection and output $[x]^q_i$.

This operation as the whole is denoted by $[x]^q = \mathsf{bit\_inject}([x]^2)$.

Efficiency: If we follow the protocol of [1], bit_inject requires $6n$-bit communications and 2-round complexity.

3. MPCs Using "Local Re-Sharing of Sub-Shares"

In this section, we introduce an important technique via a local conversion named as "local re-sharing of sub-shares" for RSSS-based MPCs.

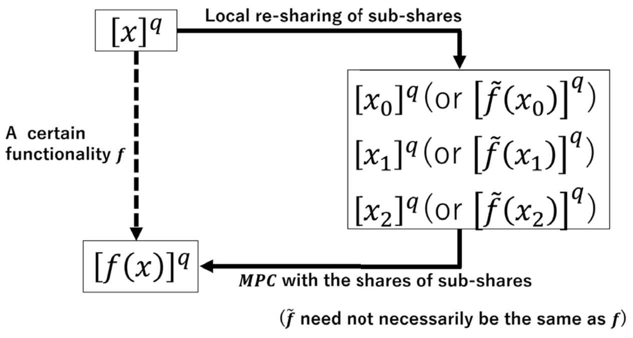
We recall a 2-out-of-3 RSSS share $[x]^q$ consists of x_0, x_1, x_2 as described in Definition 1 (see also Fig. 1).

Here we consider a MPC for computing $[f(x)]^q$ from $[x]^q$ for a function f. Unfortunately, it is generally difficult to efficiently compute $[f(x)]^q$ from $[x]^q$ directly. One promising approach is to go through a certain conversion of $[x]^q$ to get $[f(x)]^q$. As one of the conversions realizing this approach, there is an technique for share generation, which we call "local re-sharing of sub-shares" in this paper.

[1], [13], [19] proposed such a conversion by generating shares of sub-shares on $[x]^q$. More specifically, we call x_0, x_1, x_2 as *sub-shares* of x, and let $\tilde{f}$ be a function. Each party P_i can perform an conversion to obtain $[\tilde{f}(x_i)]$ $(i \in \{0, 1, 2\})$ from $\tilde{f}(x_i)$ which is locally computed. Note that this computation can be done locally. In this paper, we call the above procedure to compute shares $[\tilde{f}(x_0)]^q, [\tilde{f}(x_1)]^q, [\tilde{f}(x_2)]^q$ of the sub-shares x_0, x_1, x_2 (with $\tilde{f}$) is called "local re-sharing of sub-shares".

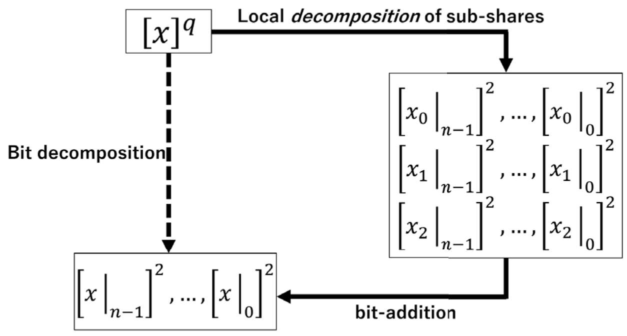
From the shares of $[\tilde{f}(x_0)]^q, [\tilde{f}(x_1)]^q, [\tilde{f}(x_2)]^q$, the parties can perform a MPC for a functionality f. Note that this technique consists of the set of the local conversion from $[x]^q$ to $[\tilde{f}(x_0)]^q, [\tilde{f}(x_1)]^q, [\tilde{f}(x_2)]^q$ and the MPC for

[†]The reason why the communication costs are not multiples of three is that the optimization procedures in [13] is assymetric.



$[x]^q$
Local re-sharing of sub-shares
$[x_0]^q$ (or $[\tilde{f}(x_0)]^q$)
$[x_1]^q$ (or $[\tilde{f}(x_1)]^q$)
$[x_2]^q$ (or $[\tilde{f}(x_2)]^q$)
A certain functionality f
$[f(x)]^q$
MPC with the shares of sub-shares
($\tilde{f}$ need not necessarily be the same as f)

Fig. 2 Local re-sharing of sub-shares.



$[x]^q$
Local *decomposition* of sub-shares
$[x_0|_{n-1}]^2, \ldots, [x_0|_0]^2$
$[x_1|_{n-1}]^2, \ldots, [x_1|_0]^2$
$[x_2|_{n-1}]^2, \ldots, [x_2|_0]^2$
Bit decomposition
$[x|_{n-1}]^2, \ldots, [x|_0]^2$
bit-addition

Fig. 3 Bit-decomposition via "local decomposition" of sub-shares.

$[f(x)]^q$ using $[\tilde{f}(x_0)]^q, [\tilde{f}(x_1)]^q, [\tilde{f}(x_2)]^q$.

If we find a good combination of $\tilde{f}$ and the MPC protocol, we can design more efficient protocol than directly computing $[f(x)]^q$ from $[x]^q$. Especially, this technique has been utilized in the context of bit-decomposition protocols [1], [13], [19] and [13] specified the general case of the local re-sharing of sub-shares for also achieving the security against malicious adversary and constructing modulus conversion.

3.1 How to Generate Shares of Sub-Shares

First, we explain how to generate shares of sub-shares. Here we consider simple case for generating the shares $[x_0]^q, [x_1]^q, [x_2]^q$ as an example.

The key observation for this idea is that every sub-share x_i ($i \in \{0,1,2\}$) is held by two parties, and the parties work on 2-out-of-3 secret sharing. Therefore, we can define the valid shares of sub-shares as $[x_0]^q = ((x_0, 0), (x_0, x_0), (0, 0))$, $[x_1]^q = ((0, 0), (x_1, 0), (x_1, x_1))$, and $[x_2]^q = ((x_2, x_2), (0, 0), (x_2, 0))$. It means that the parties can obtain the valid RSSS shares of x_0, x_1, x_2 without communication. Note that this operation does not leak information about x, since there is no communication (the secrecy is obviously guaranteed from the security of the secret sharing scheme).

In the above procedure, we can see that x_i can be replaced to $\tilde{f}(x_i)$ for arbitrary conversion $\tilde{f}$ and every $i \in \{0,1,2\}$. Thus, when the parties want to perform MPC for a certain functionality f with input $[x]^q$, they can use the shares $[\tilde{f}(x_0)]^q, [\tilde{f}(x_1)]^q, [\tilde{f}(x_2)]^q$ for any MPC, without additional communication cost. (Note that $\tilde{f}$ need not necessarily be the same as the functionality f to be computed.)

In the following, we will see how to use local re-sharing of sub-shares by an example of bit-decomposition protocols.

3.2 An Application to Bit-Decomposition

In this section, we review the bit-decompositions [1], [13], [19] as an example for local re-sharing of sub-shares.

We recall the notation $x = \sum_{i=0}^{n-1} 2^i \cdot x|_i$ where $x \in \mathbb{Z}_q$ and $x|_{n-1}, \ldots, x|_0 \in \mathbb{Z}_2$ (namely, $(x|_{n-1}, \ldots, x|_0)$ is a binary representation of x). The goal of bit-decomposition is to obtain $([x|_{n-1}]^2, \ldots, [x|_0]^2)$ from $[x]^q$, where $(x|_{n-1}, \ldots, x|_0)$ is the binary representation of x.

Figure 3 shows the overview of the protocol in [1], [13], [19]. Their bit-decomposition protocols go through a certain local conversion in the first step, named as "Local decomposition of sub-shares" in Fig. 3 instead of computing bit-decomposition directly. In the latter step, the parties perform MPC from the locally-converted shares and obtain the desired output for bit-decomposition.

We can see that the "local decomposition of sub-shares" is a special case of local-resharing of sub-shares. Namely, all bit-decomposition protocols in [1], [13], [19] can be generalized using the local re-sharing of sub-shares and the bit-addition protocol.

(1) Local Decomposition of Sub-shares

As described in Sect. 3.1, we can easily generate the shares of sub-shares $[x_0]^q, [x_1]^q, [x_2]^q$. Interestingly, this technique also can be applied to *each bit* of the sub-shares (since the party who has x_i also has each bit of x_i), we can obtain the bit-wise shares of sub-shares over $\mathbb{Z}_2$ without communication among parties. Namely, in this case, $\tilde{f}_j(x_i) = x_i|_j$ for $i \in \{0,1,2\}$ and $j \in \{0, \ldots, n-1\}$. In the bit-decomposition protocol, this type of local re-sharing has a key role as the map from $\mathbb{Z}_q$ to $(\mathbb{Z}_2)^n$.

Here we review the procedure of local decomposition of sub-shares described in [1] as follows. ([13], [19] also deploy same procedure except for a little difference of the form of the share depending the scheme)

Let $[x]^q|_j$ denote $(([x]^q_0|_j), ([x]^q_1|_j), ([x]^q_2|_j)) = (([x]^q_{0,0}|_j, [x]^q_{0,1}|_j), ([x]^q_{1,0}|_j, [x]^q_{1,1}|_j), ([x]^q_{2,0}|_j, [x]^q_{2,1}|_j))$, and $([x|_j]^q)_{j=0,\ldots,n-1} := ([x|_{n-1}]^q, \ldots, [x|_0]^q)$, where $n = \lceil \log q \rceil$.

Local Bit-Decomposition of Sub-Shares:

When every P_i for $i \in \{0,1,2\}$ is given (local_decomp, $[x]^q_i$) where $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all the parties do as follows:

Let

$$\begin{aligned} &([x_0|_j]^2)_{j=0,\ldots,n-1} \\ &\quad := ([x_0|_j]^2_0, [x_0|_j]^2_1, [x_0|_j]^2_2)_{j=0,\ldots,n-1} \\ &\quad = ((x_0|_j, 0), (x_0|_j, x_0|_j), (0, 0))_{j=0,\ldots,n-1} \\ &([x_1|_j]^2)_{j=0,\ldots,n-1} \end{aligned}$$

$$:= ([x_1|_j]^2_0, [x_1|_j]^2_1, [x_1|_j]^2_2)_{j=0,\ldots,n-1}$$
$$= ((0,0), (x_1|_j, 0), (x_1|_j, x_1|_j))_{j=0,\ldots,n-1}$$
$$([x_2|_j]^2)_{j=0,\ldots,n-1}$$
$$:= ([x_2|_j]^2_0, [x_2|_j]^2_1, [x_2|_j]^2_2)_{j=0,\ldots,n-1}$$
$$= ((x_2|_j, x_2|_j), (0,0), (x_2|_j, 0))_{j=0,\ldots,n-1}$$

For each $i \in \{0,1,2\}$, P_i generates and outputs (local_decomp, $([x_0|_j]^2_i, [x_1|_j]^2_i, [x_2|_j]^2_i)_{j=0,\ldots,n-1}$). This operation as the whole is denoted by $([x_0|_j]^2, [x_1|_j]^2, [x_2|_j]^2)_{j=0,\ldots,n-1}) =$ local_decomp($[x]^q$).

(2) Bit-Addition for $x_0 + x_1 + x_2$

If we obtain the shares of binary expression of x_0, x_1, x_2, we can straightforwardly compute the share of binary expression of x by performing the MPC for bit-addition described in Sect. 2.5.2. We recall it is exactly desired output of bit_decomp since $x = x_0 + x_1 + x_2$ by definition. Here we conclude the procedure of bit-decomposition.

As seen in the previous paragraph, local_decomp is the local operation. Therefore, the dominant part of the complexity in bit_decomp is bit_add.

3.3 Toward the Application for Modular Exponentiation

As described above, the local re-sharing of sub-shares is an interesting technique relied on the specific structure of RSSSs. However, although the technique seems to be very generic, it's not obvious how this functionality is effective in other MPCs. Its effective applications are not well studied, except for the technique in [1], [13], [19], to the best of our knowledge.

In Sect. 4.2, we introduce a special case of the local re-sharing technique, as named local_expo, and discuss how to apply it to modular exponentiation in the later sections. Our proposed scheme based on local re-sharing is basically simple, but the effect on efficiency is quite large.

4. MPC for Exponentiation

4.1 Known MPC Scheme for Exponentiation

Before describing the proposed schemes, we will see the standard way to compute exponentiation on MPC.

Modular exponentiation with public base: Given a public value $a \in \mathbb{Z}_q$ and a share $[x]^q_i$, the algorithm pub_expo outputs $[y]^q_i = [a^x]^q_i$ which is a share of modular exponentiation for the base a and exponent x (over the modulus q).

More specifically, when every P_i for $i \in \{0,1,2\}$ is given (pub_expo, a, $[x]^q_i$), all parties do as follows:

```
1. $[x_{n-1}]^2_i, \ldots, [x_0]^2_i$ = bit_decomp($[x]^q_i$)
2. for $j = 0, \ldots, n-1$ : $[x_j]^q_i$ = bit_inject($[x_j]^2_i$)
3. $[y]^q_i = [1]^q_i$
4. for $j = 0, \ldots, n-1$ :
       $[y]^q_i = a^{2^j} \cdot [x_j]^q_i \cdot [y]^q_i + (1 - [x_j]^q_i [y]^q_i)$
5. outputs (pub_expo, $[y]^q_i$)
```

The above procedure is an implementation of the "square-and-multiply" method, which is a standard way of computing modular exponentiation. This method can be applied also the base is private.

Efficiency: We can easily see that the dominant part of complexity in this procedure is the for-loop at Step 4. Here we denote R_{bd} and C_{bd} (resp., R_{inj} and C_{inj}) be the round complexity and the communication complecxity of bit-decomposition (resp., bit-injection), respectively. In the above procedure, step 1 takes R_{bd}-round and C_{bd}-bit communication complexity. Step 2 takes R_{inj}-round and $n \cdot C_{inj}$-bit communication complexity (note that each bit_inject can be performed in parallel). Step 3 is a deterministic procedure and hence takes no round and communication complexity. In Step 4, two multiplication are invoked n times sequentially. We recall each multiplication of n-bit ring takes 1-round and $3n$-bit communication complexity. Thus, Step 4 takes n-round and $6n^2$-bit communication complexity. In total, pub_expo takes $(R_{bd} + R_{inj} + n)$-round and $(C_{bd} + nC_{inj} + 6n^2)$-bit communication complexity. If we apply the scheme in [1], [3] (for the efficiency, see Sect. 2.5), it takes $(n-1) + 2 + n = 2n + 1$-round and $6(n-1) + n \cdot 6n + 6n^2 = 12n^2 + 6n - 6$-bit communication complexity.

4.2 Local Exponentiation of Sub-Shares

The overview of the proposed modular exponentiation is shown in Fig. 4. In this section, we define a local operation local_expo, which is a special case of the local re-sharing of sub-shares in Sect. 3.1.

Note that the construction of MPC modular exponentiation is still *not* straightforward even if we introduce local_expo. In the later section, we discuss how to apply local_expo to the modular exponentiation on several settings.

Local Exponentiation of Sub-shares: When every P_i for $i \in \{0,1,2\}$ is given (local_expo, a, $[x]^q_i$) where $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all parties do as follows:

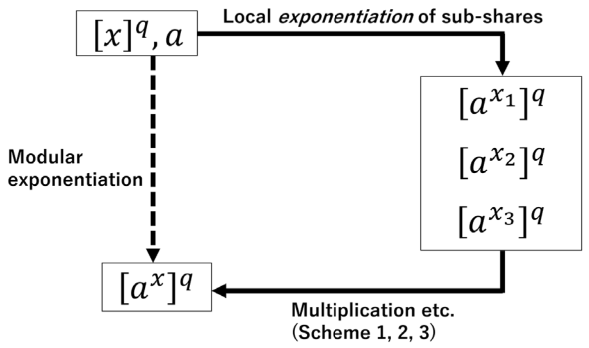


Fig. 4 Proposed modular exponentiation via "local exponentiation" of sub-shares.

1. Let $[a^{x_0}]^q = ((a^{x_0}, a^{x_0}), (a^{x_0}, 0), (0, 0))$
2. Let $[a^{x_1}]^q = ((0, 0), (a^{x_1}, a^{x_1}), (a^{x_1}, 0))$
3. Let $[a^{x_2}]^q = ((a^{x_2}, 0), (0, 0), (a^{x_2}, a^{x_2}))$
4. Output (local_expo, $[a^{x_0}]^q, [a^{x_1}]^q, [a^{x_2}]^q$)

4.3 Proposed Schemes

We recall that what we want to compute is $a^x = a^{x_0+x_1+x_2 \mod m}$. Note that x is *not* equal to $x_0 + x_1 + x_2$ but $x_0 + x_1 + x_2 \mod m$. Therefore, we couldn't compute the share $[a^x]$ by $[a^{x_0}] \cdot [a^{x_1}] \cdot [a^{x_2}]$ naively, without checking whether the sum $x_0 + x_1 + x_2$ goes over modulus.

4.3.1 Scheme 1: The Case Where Modulus is Prime

Here we consider the case the modulus is prime p (namely, $q = p$).

In this case, we can see that $a^x = a^{x'+kp} = a^x a^k$ where $k \in \{0, 1, 2\}$ by Fermat's little theorem. Namely, $x_0 + x_1 + x_2$ can go over the modulus at most twice. Therefore, we should check modulus overflow at the point of computing $x_0 + x_1$ and $(x_0 + x_1) + x_2$. If the these values go over the modulus p, we can fix it by multiplying a^{-1}.

To detecting the modulus overflow, we use bit-decomposition in this protocol. The point is that, if a certain value a is larger than the modulus p, the parity of $a \mod p$ is flipped, since p is *odd*. Therefore, when we check the least significant bit of x_0, x_1, and $x_0 + x_1 \mod p$, if the parity is not consistent, it means that $x_1 + x_2$ exceed p.

When every P_i for $i \in \{1, 2, 3\}$ is given (modexp$_\mathsf{p}$, a, $[x]^q_i$) where $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all parties do as follows:

1. $([a^{x_0}]^q, [a^{x_1}]^q, [a^{x_2}]^q) \leftarrow$ local_expo($[x]^q$)
2. $[s]^q = [a^{x_0}]^q \cdot [a^{x_1}]^q \cdot [a^{x_2}]^q$
3. $[d_{n-1}]^2, \ldots, [d_0]^2 =$ bit_decomp($[x_0]^q$)
4. $[e_{n-1}]^2, \ldots, [e_0]^2 =$ bit_decomp($[x_1]^q$)
5. $[f_{n-1}]^2, \ldots, [f_0]^2 =$ bit_decomp($[x_0 + x_1]^q$)
6. $[b_1]^2 = [x_0 + x_1 > p]^2 = [d_0 \oplus e_0 \neq f_0]^q = [d_0 \oplus e_0 \oplus f_0]^q$
7. $[g_{n-1}]^2, \ldots, [g_0]^2 =$ bit_decomp($[x_2]^q$)
8. $[h_{n-1}]^2, \ldots, [h_0]^2 =$ bit_decomp($[x_0 + x_1 + x_2]^q$)
9. $[b_2]^2 = [x_0 + x_1 + x_2 > p]^2 = [f_0 \oplus g_0 \neq h_0]^q = [f_0 \oplus g_0 \oplus h_0]^q$
10. $[b_1]^q =$ bit_inject($[b_1]^2$)
11. $[b_2]^q =$ bit_inject($[b_2]^2$)
12. $[t]^q = [s]^q \cdot [b_1]^q \cdot a^{-1} + [s]^q(1 - [b_1]^q)$
13. $[t]^q = [t]^q \cdot [b_2]^q \cdot a^{-1} + [t]^q(1 - [b_2]^q)$
14. output (modexp$_\mathsf{p}$, $[t]^q$)

Step 3–6 is the description of modulus overflow check for $x_0 + x_1$, and similarly Step 7–9 is check for $(x_0 + x_1) + x_2$. What we actually need are only least significant bit (LSB) of these values, we don't have to compute full procedure of bit_decomp, but can close the process when we get LSBs of the values.

Efficiency: Each bit_decomp and multiplication can be performed in parallel. In the above procedure, Step 2 takes 2-round and $6n$-bit communication. Each bit_decomp takes $(n+1)$-round and $10n+4$-bit communication (using [13] since q is prime) and Step 3, 4, 5, 7, and 8 can be done in parallel. Steps 10 and 11 take 2-round and $6n$-bit communication respectively and these steps can be done in parallel. Step 12 and 13 take 1-round and $6n$-bit communiation respectively. In total, modexp$_\mathsf{p}$ takes $2 + (n + 1) + 2 + 1 + 1 = (n + 7)$-round and $6n + 5 \cdot (10n + 4) + 2 \cdot 6n + 2 \cdot 6n = (80n + 20)$-bit communication complexity.

4.3.2 Scheme 2: The Case Where Modulus is Power of 2

Next we consider the case where $q = 2^n$ for some $n \in \mathbb{Z}$. To consider this case, we recall Euler's theorem.

Theorem 4 (Euler's theorem): If n and a are coprime positive integers, $a^{\phi(n)} \equiv 1 \pmod{n}$ where $\phi(\cdot)$ is Euler's totient-function.

By Euler's theorem, if a is prime, $a^{2^{n-1}} = 1 \mod 2^n$, which implies $a^{2^n} = 1 \mod 2^n$. Namely, in this case, we don't have to check the overflow of exponent.

When every P_i for $i \in \{0, 1, 2\}$ is given (modexp_2n, a, $[x]^q_i$) where $q = 2^n$ and $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all parties do as follows:

1. $[a^{x_0}]^q, [a^{x_1}]^q, [a^{x_2}]^q =$ local_expo($[x]^q$)
2. $[s]^q = [a^{x_0}]^q \cdot [a^{x_1}]^q \cdot [a^{x_2}]^q$
3. output (modexp_2n, $[s]^q$).

Note that we cannot apply this procedure if a is even. In addition, it is difficult to apply the technique like Scheme 1 since there is no multiplicative inverse for all even value in $\mathbb{Z}_{2^n}$, that is we cannot compute a^{-1} on $\mathbb{Z}_{2^n}$ if a is even. However, if we encounter case to apply the Scheme 2, this is very efficient.

Efficiency: Scheme 2 requires requires only 2 multiplication for n-bit elements and *no* bit_decomp. Total cost of Scheme 2 is 2 rounds and $6n$-bit communication complexity.

4.3.3 Scheme 3: Special Case that the Discrete Logarithm is Small

We can consider the case where the size of the base and the exponent value are different. For example, we consider the case where $p = 2q + 1$, and $x_0, x_1, x_2 \in \mathbb{Z}_q$, $a \in \mathbb{Z}_p$. In such case, $x_0 + x_1 + x_2$ can exceed p at most *once*.

When every P_i is given (modexp_sp, a, $[x]^q_i$) for $i \in \{0, 1, 2\}$ where $p = 2q+1$, $a \in \mathbb{Z}_p$ and $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all parties do as follows:

1. $[a^{x_0}]^p, [a^{x_1}]^p, [a^{x_2}]^p =$ local_expo($[x]^q$)
2. $[s]^p = [a^{x_0}]^p \cdot [a^{x_1}]^p \cdot [a^{x_2}]^p$
3. $[d_{n-1}]^2, \ldots, [d_0]^2 =$ bit_decomp($[x_0 + x_1]^q$)
4. $[e_{n-1}]^2, \ldots, [e_0]^2 =$ bit_decomp($[x_1]^p$)
5. $[f_{n-1}]^2, \ldots, [f_0]^2 =$ bit_decomp($[x_0 + x_1 + x_2]^p$)

Table 1 Complexity of MPC for modular exponentiation over replicated secret sharing.

Method	Round	Communication
Previous (Square-and-Multiply)	$2\lceil\log q\rceil + 1$	$12\lceil\log q\rceil^2 + 6\lceil\log q\rceil - 6$
Scheme 1 (q is prime)	$\lceil\log q\rceil + 7$	$80\lceil\log q\rceil + 20$
Scheme 2 (q is power of 2)	2	$6\lceil\log q\rceil$
Scheme 3 ($2q + 1 < p \le 3q + 1$)	$\lceil\log p\rceil$+4	$36\lceil\log p\rceil$ - 18
Scheme 3 ($3q + 1 \le p$)	2	$6\lceil\log p\rceil$

6. $[b]^2 = [x_0 + x_1 + x_2 > p]^2 = [d_0 \oplus e_0 \oplus f_0]^2$
7. $[b]^p$ = bit_inject($[b]^2$)
8. $[t]^p = [s]^p \cdot [b]^p \cdot a^{-1} + [s]^p(1 - [b]^p)$
9. output (modexp_sp, $[s]^q$).

In addition, if the case $p > 3q + 1$, we don't have to check the overflow of $x_0 + x_1 + x_2$ since $x_0 + x_1 + x_2 \bmod p = x_0 + x_1 + x_2$ in this parameter.

When every P_i for $i \in \{0, 1, 2\}$ is given (modexp_sp2, a, $[x]^q_i$) where $p = 2q + 1$, $a \in \mathbb{Z}_p$ and $([x]^q_{0,1}, [x]^q_{1,1}, [x]^q_{2,1}) = (x_0, x_1, x_2)$, all parties do as follows:

1. $[a^{x_0}]^p, [a^{x_1}]^p, [a^{x_2}]^p$ = local_expo($[x]^p$)
2. $[s]^p = [a^{x_0}]^p \cdot [a^{x_1}]^p \cdot [a^{x_2}]^p$
3. output (modexp_sp2, $[s]^p$).

Efficiency: Scheme 3 requires less number of invocation of bit_decomp. In particular, in the case where $3q + 1 \le p$, we can perform same procedure as Scheme 2. If the case where $2q + 1 < p \le 3q + 1$ total cost of Scheme 3 is obviously $n + 4$ rounds and $(36n - 18)$-bit communication complexity. If the case where $3q + 1 < p$, Scheme 3 takes only 2 rounds and $6n$-bit communication as same as Scheme 2.

4.4 Efficiency Comparison

We summarize the round and communication complexity for each protocol in Table 1. Basically, our proposed schemes requires $O(n)$ round and $O(n)$-bit communication complexity, whereas the previous scheme requires $O(n^2)$-bit communication.

As an example, we show the comparison of communication bits between previous scheme and Scheme 1. We can clearly see how efficient the proposed schemes is compared with the previous scheme. The previous scheme takes over 300kb when $n = 256$ bits. On the other hand, Scheme 1 requires 15,360-bit communication. As for Scheme 2 or 3 with $3q + 1 \le p$ case, these takes only 1,536-bit communication.

5. Proof of Security

In this section, we discuss the proof of the security for the proposed scheme described in Sect. 4.3.

5.1 Universal Composability

First, we confirm how the security of sub-protocols (like

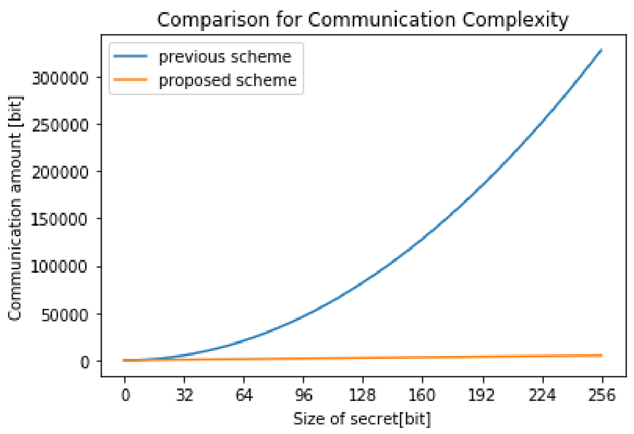


Fig. 5 Comparison for communication bits between previous scheme and scheme 1 in this paper.

addition and multiplication described in Sect. 2.4, and local-resharing of sub-shares) can imply the security of the proposed scheme.

Here we describe the concept of *universal composability* (UC) framework proposed by [6]. Protocols which is secure in UC framework maintain its security even if it is composed with arbitrary other (secure and insecure) protocols. In particular, [14] clarified a condition under which the security of protocols implies the security of these protocols under universal composition as follows.

Proposition 1 (Thm. 1.5 in [14]): Every protocol that is secure in the stand-alone model and has start synchronization and a straight-line black-box simulator is secure under concurrent general composition (universal composition).

In the above theorem, "straight-line" simulator means that the non-rewinding simulator, and "start synchronization" means that the inputs of all parties are fixed before the execution begins (also called as "input availability).

Our protocols and sub-protocols in this paper satisfies start synchronization. Therefore, it is sufficient to prove security in the classic stand-alone setting and automatically derive universal composability.

5.2 Security of Sub-Protocols

The security of sub-protocols described in Sect. 2.4 are proven in [3].

The proof in [3] consist of three steps as follows. Here we denote $\pi^{\mathcal{F}} \equiv f$ to say that π privately computes f in the $\mathcal{F}$-hybrid model.

1. Proving the sub-protocols π privately compute f in the $\mathcal{F}_{mult}$-hybrid model in the presence of one semi-honest corrupted party, where $\mathcal{F}_{mult}$ is an ideal functionality for computing multiplication (namely, $\pi^{\mathcal{F}_{mult}} \equiv f$).
2. Proving a protocol ρ privately computes $\mathcal{F}_{mult}$ in the $\mathcal{F}_{CR}$-hybrid model in the presence of one semi-honest corrupted party, where $\mathcal{F}_{CR}$ is an ideal functionality

for computing correlated randomness (namely, $\mathcal{F}_{mult} \equiv \rho^{\mathcal{F}_{CR}}$).

3. Proving a protocol σ privately computes $\mathcal{F}_{CR}$ in plain model in the presence of one semi-honest corrupted party (namely, $\sigma \equiv \mathcal{F}_{CR}$).

The above three steps and the composition theorem described in Theorem 1 lead $\pi^{\rho^{\sigma}} \equiv f$, which concludes the proof.

5.3 Security of Our Protocols

Adding to the sub-protocols in Sect. 2.4, our protocols contain one more sub-protocol, that is, local exponentiation (or local re-sharing of sub-shares).

Fortunately, we can easily confirm that step 1 of the above proof still can be proven even π contains local re-sharing of sub-shares since this functionality consist of local computation only, as same as the addition protocol. This is not affect the simulation in the $\mathcal{F}_{mult}$-hybrid model and $\pi^{\mathcal{F}_{mult}} \equiv f$. Regarding Step 2 and 3 of the proof, we can apply same proof as [3] for our protocol since we adopt same algorithms for multiplication and correlated randomness.

Finally, we can apply the proof of Sect. 5.2 to our protocol and thus all sub-protocols including local re-sharing of sub-shares are secure in terms of Definition 3. As described above, Theorem 1 [14] guarantee that our all sub-protocols are secure in the UC model. Therefore, by the UC composition theorem [6], we can prove the security of our protocols in Sect. 4.3.

On the Security for Malicious Adversaries We recall that our protocols in this paper are basically secure against *semi-honest* adversaries (see Definition 1). However, the protocols also can be secure in the presence of malicious adversaries by applying the technique of [11] (or its optimized version [2]), which allows us to construct a 3PC for malicious adversaries from 3PC protocols for semi-honest adversaries. If we apply the scheme in [2], the communication complexity is roughly 7 times that of the protocols in Sect 4.

6. Applications

The cryptosystems which are based on discrete logarithm are basically constructed by modular exponentiations of group elements, these are suitable for our schemes. Table 2 shows the key sizes of discrete-log based cryptosystems which are recommended in some evaluation documents. We can see that the size of exponent is much smaller than the size of group elements. In this setting, we can apply Scheme 3 in this paper, which is most efficient one in our proposal.

Table 2 Appropriate data length of discrete-log based cryptosystem for 128/256-bit security.

Method	discrete keys		Logarithm Group	
Security level	128	256	128	256
Lenstra/Verheul [16]	230	474	6790	49979
Lenstra Updated [15]	256	512	4440	26268
ECRYPT [10]	256	512	3072	15360
NIST [20]	256	512	3072	15360
ANSSI [21]	200	200	2048	3072
RFC3776 [22]	256	512	3253	15489

6.1 Experimental Evaluation

(1) Experiment Scenario

We consider a simple scenario for distributed signatures based on discrete logarithm: the key storage server is distributed three parties P_1, P_2, P_3. Let the signing key x of the signature scheme is a element of $\mathbb{Z}_q$ where q is prime. We assume x is shared among P_1, P_2, P_3 by the RSSS. Now, a certain authorized user throw a query to the distributed server to generate own signature $\sigma \in \mathbb{Z}_p$ using shared his/her signing key by MPC, where p is prime satisfying $p > 3q + 1$ (this assumption is reasonable according to Table 2). As signature schemes suitable for such scenario, we can choose BLS signature [5] or Waters signature [26].

(2) Environment and Settings

We run our experiments on a cluster of three servers, each with two 10-core Intel Xeon (E5-2650 v3) processors and 128GB RAM, connected via a 10Gbps Ethernet. (We remark that little RAM was utilized and thus this is not a parameter of importance here.)

Based on the parameter shown in Table 2, we run two experiments assuming 128-bit security and 256-bit security, respectively. From Table 1, we implement and compare Scheme 3 and previous scheme with the field of size $n = \log p$. For each experiment, we measure the latency of one MPC process of modular exponentiation, while fixing the size of q (discrete logarithm) and changing the size of p (i.e., the size of field).

We also run experiments for various network latency using tc[†] command on Linux. In the experiments, we tried three latency settings assuming LAN/WAN: 0.1 ms, 5 ms and 50 ms. We suppose that 0.1 ms is very low-latency of LAN, 5 ms is round-trip delay of 500 km distance (e.g.,between Tokyo-Osaka), and 50 ms is round-trip delay of 5000 km distance (e.g., between Los Angeles-New York) [††].

(3) Results and Discussion

The results on Scheme 3 are shown in Fig. 6 and Fig. 7. We can see that the proposed scheme works even on WAN network at the same speed as the LAN network latency. This characteristic comes from that the round complexity is constant for the size of the field. On the other hand, the previous scheme takes $O(n)$ rounds and therefore the latency is much larger than the result in Figs. 6 and 7. For example, if the size of field is 2048, the round complexity

[†]This command allows us to show/change network traffic settings, like latency, packet loss, etc. (The name means “traffic control”.)

[††]We assume the speed of the light passing through the optical fiber is roughly 200,000 km/s.

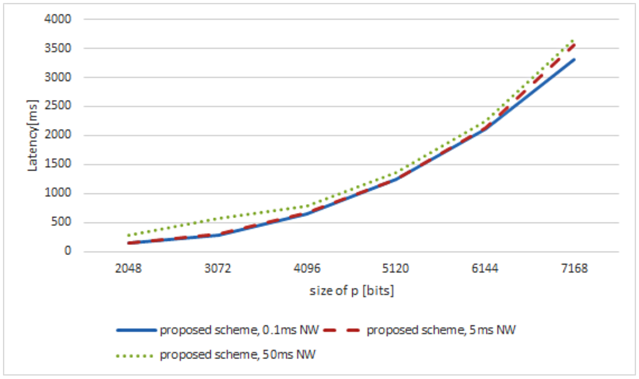


Fig. 6 Latency-field size with 128-bit security parameter ($\log q = 256$ bit).

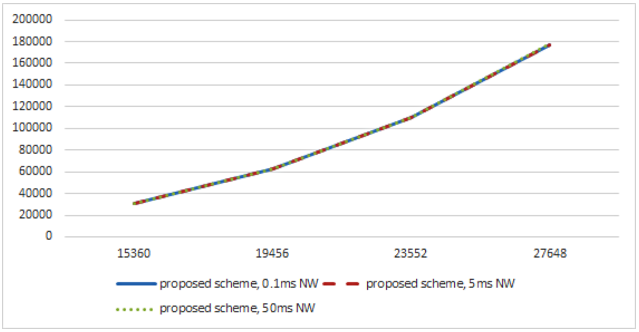


Fig. 7 Latency-field size with 256-bit security parameter ($\log q = 512$ bit).

of proposed scheme is $3 \cdot 2048 + 2 = 6146$ according to Table 2. It takes $0.1 \cdot 6,146 = 614.6$ ms when the case of 0.1 ms-latency network, and $50 \cdot 6146 = 307,300$ ms when the case of 50 ms-latency network ignoring the computation cost etc. In the worst case of this experiment, which is the $\log p = 27,648$ bits with 50 ms-latency network for 256-bit security, it takes $50 \cdot (3 \cdot 27,648 + 2) = 4,147,300$ ms = 69.1 minutes for *only* network delay. Namely, our proposed scheme is roughly one or two order of magnitude faster than previous scheme in a certain setting.

7. Conclusion

We propose MPC protocols for modular exponentiation on the state-of-the-art MPC framework. The proposed schemes is a new application based on the local re-sharing of sub-shares. The construction of our scheme is simple and efficient in both of aysmptotic and practical sense, as shown by the experiment on a scenario on distributed signatures.

From the results of this paper, we can learn the necessity of dedicated MPC instructions for such MPC framework. Basically, the naive construction of MPC based on secret sharing-based technique takes a large number of round. As we discussed in Sect. 6.1, it is serious concern, in particular WAN environment.

However, these secret sharing-based MPCs still do not impair their worth, since we have the possibility to reduce the complexity by dedicated design as we show, and these frameworks naturally have a small communication complexity. We believe our proposal in this paper also supports solving the performance problems on secret sharing-based MPCs.

Acknowledgments

The authors are grateful to the reviewers for their careful reading and comments. This work was partially supported by JSPS KAKENHI Grant Numbers JP17H01752, JP18H05289, JP18K11293, JP18H03238, and 18K19780.

References

[1] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida, "Generalizing the SPDZ compiler for other protocols," IACR Cryptology ePrint Archive, 2018:762, 2018.
[2] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein, "Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier, IEEE S&P 2017, pp.843–862, 2017.
[3] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," ACM CCS 2016, pp.805–817, 2016.
[4] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," ESORICS 2008, pp.192–206, 2008.
[5] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," J. Cryptol., vol.17, no.4, pp.297–319, 2004.
[6] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," FOCS 2001, pp.136–145, 2001.
[7] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, "Fast large-scale honest-majority MPC for malicious adversaries," CRYPTO 2018, pp.34–64, 2018.
[8] R. Cramer, I. Damgård, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," TCC 2005, pp.342–362, 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," TCC 2006, pp.285–304, 2006.
[10] ECRYPT-CSA, Algorithms, key size and protocols report (2018), H2020-ICT-2014 – Project 645421, European Coordination and Support Action in Cryptology, 2018.
[11] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," EUROCRYPT 2017, pp.225–255, 2017.
[12] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," ACNS 2016, pp.156–174, 2016.
[13] R. Kikuchi, D. Ikarashi, T. Matsuda, K. Hamada, and K. Chida, "Efficient bit-decomposition and modulus-conversion protocols with an honest majority," ACISP 2018, pp.64–82, 2018.
[14] E. Kushilevitz, Y. Lindell, and T. Rabin, "Information-theoretically secure protocols and security under composition," SIAM J. Comput., vol.39, no.5, pp.2090–2112, 2010.
[15] A.K. Lenstra, Key Lengths, The Handbook of Information Security, Wiley, 2004.
[16] A.K. Lenstra and E.R. Verheul, "Selecting crytographic key sizes," J. Cryptol., vol.14, no.4, pp.255–293, 2001.
[17] Y. Lindell, "Fast secure two-party ECDSA signing," CRYPTO 2017, pp.613–644, 2017.
[18] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay - secure two-party computation system," Proc. 13th USENIX Security Symposium, pp.287–302, San Diego, CA, USA, Aug. 2004.

[19] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," ACM CCS 2018, pp.35–52, 2018.

[20] NIST, Recommendation for key management, Special Publication 800-57 Party 1 Rev.4, 2016.

[21] NSA, Mécanismes cryptographiques, Règles et recommandations, Rev. 2.03, 2014.

[22] NSA, Commercial national security algorithms, Information Assurance Directorate at the NSA, 2016.

[23] K. Ohara, K. Ohta, K. Suzuki, and K. Yoneyama, "Constant rounds almost linear complexity multi-party computation for prefix sum," AFRICACRYPT 2014, pp.285–299, 2014.

[24] A. Shamir, "How to share a secret," Commun. ACM, vol.22, no.11, pp.612–613, 1979.

[25] Y. Wang, D.S. Wong, Q. Wu, S.S.M. Chow, B. Qin, and J. Liu, "Practical distributed signatures in the standard model," CT-RSA 2014, pp.307–326, 2014.

[26] B. Waters, "Efficient identity-based encryption without random oracles," Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, pp.114–127, Aarhus, Denmark, May 2005.

[27] A.C.-C. Yao, "Protocols for secure computations (extended abstract)," FOCS 1982, pp.160–164, 1982.

Kazuma Ohara received the B.E., and M.E. degrees from the University of Electro-Communications in 2012 and 2014, respectively. He is presently engaged in research on public key cryptography and secure multi-party computation at NEC corporation, since 2014. He received the SCIS Paper Award in 2014, and ACM CCS 2016 Best Paper Award in 2016.

Yohei Watanabe received the B.E., M.E., and Ph.D. degrees in information science from Yokohama National University, Japan, in 2011, 2013, and 2016, respectively. He was also JSPS Research Fellow (DC1) during his Ph.D. course. After spending two years and a half as JSPS Research Fellow (PD) at the University of Electro-Communications, he moved to National Institute of Information and Communications Technology (NICT) on Oct. 2018. His research interests include cryptography and information security, especially unconditional cryptography and pairing-based cryptography. He received several awards including SCIS Paper Prize, CSS 2018 Paper Prize, and IEEE Information Theory Society Japan Chapter Young Researcher Best Paper Award. He was invited to 4th Heidelberg Laureate Forum as one of selected young researchers. He is a member of IEICE, IPSJ, IEEE, and IACR.

Mitsugu Iwamoto received the B.E., M.E., and Ph.D. degrees from the University of Tokyo, Tokyo, Japan, in 1999, 2001, and 2004, respectively. In 2004, he joined the University of Electro-Communications, where he is currently an Associate Professor of Department of Informatics. His research interests include information theory, information security, and cryptography. He is a member of IEICE, IEEE, and IACR.

Kazuo Ohta received the B.S., M.S., and Dr.S. degree from Waseda University, Tokyo, Japan, in 1977, 1979, and 1990 respectively. He has been a Professor at The University of Electro-Communications since 2001. He had been a researcher at NTT laboratories between 1979 and 2001. He is presently engaged in research on information security. He is a fellow of IEICE, and a member of IEEE and IACR.