Constant-Rounds, Linear Multi-party Computation for Exponentiation and Modulo Reduction with Perfect Security*

Chao Ning** and Qiuliang Xu***

School of Computer Science and Technology, Shandong University,
Jinan, 250101, China
ncnfl@mail.sdu.edu.cn, xql@sdu.edu.cn

Abstract. Bit-decomposition is an important primitive in multi-party computation (MPC). With the help of bit-decomposition, we will be able to construct constant-rounds protocols for various MPC problems, such as *equality test*, *comparison*, *public modulo reduction* and *private exponentiation*, which are four main applications of bit-decomposition. However, when considering perfect security, bit-decomposition does *not* have a linear communication complexity; thus any protocols involving bit-decomposition inherit this inefficiency. Constructing protocols for MPC problems without relying on bit-decomposition is a meaningful work because this may provide us with perfectly secure protocols with linear communication complexity. It is already proved that *equality test*, *comparison* and *public modulo reduction* can be solved without involving bit-decomposition and the communication complexity can be reduced to linear. However, it remains an open problem whether *private exponentiation* could be done without relying on bit-decomposition. In this paper, maybe somewhat surprisingly, we show that it can. That is to say, we construct a *constant-rounds, linear, perfectly secure* protocol for private exponentiation *without* relying on bit-decomposition though it seems essential to this problem.

In a recent work, Ning and Xu proposed a generalization of bit-decomposi-tion and, as a simplification of their generalization, they also proposed a linear protocol for public modulo reduction. In this paper, we show that their generalization can be further generalized; more importantly, as a simplification of our further generalization, we propose a public modulo reduction protocol which is more efficient than theirs.

Keywords: Multi-party Computation, Perfectly Secure, Constant-Rounds, Linear, Exponentiation, Modulo Reduction, Bit-Decomposition.

* Supported by the National Natural Science Foundation of China Grant 60873232, 61173139, the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

** Chao Ning is now at ITCS, part of the IIIS at Tsinghua University, China.
*** Corresponding author.

D.H. Lee and X. Wang (Eds.): ASIACRYPT 2011, LNCS 7073, pp. 572–589, 2011.

1 Introduction

Multi-party computation (MPC) is a powerful and interesting tool in cryptology. It allows a set of n mutually un-trusted parties to compute a predefined function f with their private information as inputs. After running the MPC protocol, the parties obtains only the predefined outputs but nothing else, and the privacy of their inputs is guaranteed. Although generic solutions for MPC (which can compute any function f) already exist [3,9], these solutions tend to be inefficient and thus not applicable for practical use. So, to fix this problem, we focus on constructing efficient protocols for specific functions.

Recently, in the work [6], Damgård *et al.* proposed a novel technique called *bit-decomposition* which can, in constant rounds, convert a polynomial sharing of secret x into the sharings of the bits of x. Bit-decomposition (which will often be referred to as **BD** hereafter for short) is a very useful tool for MPC. For example, after getting the sharings of the bits of some shared secrets using BD, we can securely perform Boolean operations on these secrets (such as computing the Hamming Weight, XOR, etc). Thus we can say that BD can be viewed as a "bridge" (in the world of MPC) connecting the arithmetic circuits and the Boolean circuits. What's more, with the help of BD, we can construct constant-rounds protocols for some very important basic problems in MPC, such as *equality test*, *comparison*, *public modulo reduction* and *private exponentiation*, which will be referred to as four main applications of BD. After getting the bit-wise sharings of the shared inputs to these problems (using BD), we will be able to use the *divide and conquer* technique to solve these problems.

However, a problem is, BD is relatively expensive in terms of round and communication complexities, and thus all the protocols relying on BD inherit this inefficiency. For example, the communication complexity of BD (with perfect security) is non-linear, thus all the protocols involving BD have a non-linear communication complexity. A feasible solution for this problem is to construct protocols for MPC problems without relying on BD. It is already proved that, three of the four main applications of BD, i.e. *equality test*, *comparison* and *public modulo reduction*, can be realized without relying on BD [11,12] and the main advantage is that the communication complexity can be reduced to linear (under the premise of ensuring perfect security). Thus a natural problem is whether a similar conclusion can be arrived at for another important application of BD: *private exponentiation*. This is generally believed to be impossible before (e.g. [11], Page 2; [14], Page 2), however, in this paper, we show that it can. What's more, we show an improvement of the public modulo reduction protocol (without BD) proposed in [12]. The details of our results are presented below. Here we'd like to argue that although these four applications of BD can be realized without involving BD, this does *not* mean BD is meaningless for these problems because all these protocols (without relying on BD) depend heavily on the ideas, techniques and sub-protocols of BD and thus can be viewed as an extension of the research on BD.

1.1 Our Results

First we introduce some necessary notations. In this paper, we concern mainly about MPC based on linear secret sharing schemes (LSSS). Assume that the underlying LSSS is built on field $\mathbb{Z}_p$ where p is a prime with bit-length l (i.e. $l = \lceil \log p \rceil$). For an element $x = (x_{l-1}, ..., x_1, x_0) \in \mathbb{Z}_p$, we use $[x]_p$ to denote "*the sharing of* x", and $[x]_B$ to denote "*the bitwise sharing of* x" (which will also be referred to as "*the sharings of the bits of* x" or "*the shared base-2 form of* x" in this paper), i.e. $[x]_B = ([x_{l-1}]_p, ..., [x_1]_p, [x_0]_p)$.

Our work is mainly about two basic problems in MPC: the *private exponentiation problem* and the *public modulo reduction problem*. We construct efficient protocols, which are *constant-rounds, linear and perfectly secure*, for these two problems. The details are presented below.

The *private exponentiation problem* can be formalized as:

$$[x^a \bmod p]_p \leftarrow \text{Private-Exponentiation}([x]_p, [a]_p)$$

where $x, a \in \mathbb{Z}_p$.

Hereafter we will refer to $[x^a \bmod p]_p$ as $[x^a]_p$ for simplicity. For solving this problem, it seems that we must involve BD to get the bitwise sharing of the exponent, i.e. $[a]_B$. This is exactly the case in the private exponentiation protocol in [6]. However, in this paper we show that this is not necessary. That is to say, the private exponentiation problem can also be solved without relying on BD and the communication complexity can also be reduced to linear (*in the input length* l). Compared with the private exponentiation protocol in [6] (denoted as Pri-Expo-BD$(\cdot)$ in this paper), our protocol (denoted as Pri-Expo$^+(\cdot)$) reaches lower round complexity and much lower communication complexity.

The *public modulo reduction problem* (which will be referred to as *Pub-MRP* for short) can be formalized as:

$$[x \bmod m]_p \leftarrow \text{Public-Modulo-Reduction}([x]_p, m)$$

where $x \in \mathbb{Z}_p$ and $m \in \{2, 3, ..., p-1\}$.

Our work on this problem can be viewed as an extension of [12], in which Ning and Xu proposed a generalization of BD and, as a simplification of their generalization, they proposed a linear protocol for Pub-MRP without involving BD (denoted as Pub-MR$(\cdot)$ in this paper). In this paper, we propose a further generalization of their generalization and, similarly and more importantly, as a simplification of our further generalization, we propose a protocol for Pub-MRP with improved efficiency (denoted as Pub-MR$^+(\cdot)$). Specifically, the round complexity of our Pub-MR$^+(\cdot)$ protocol is the same with Pub-MR$(\cdot)$ and, for relatively small m, the communication complexity is reduced by a factor of approximately 4.

We'd like to stress that all the protocols constructed in this paper are constant-rounds and perfectly secure. See Appendix A (Table 1) for an overview of our protocols. What's more, we strongly recommend the interested readers to read [13] which is the full version of this paper. Many of the details are omitted in the present paper due to space constraints.

1.2 Related Work

Both of the two problems considered in this paper, *exponentiation* and *modulo reduction*, are applications of bit-decomposition (BD). The problem of BD was first considered by Algesheimer *et al.* in [1], in which a partial solution was proposed. The first full solution for BD in the secret sharing setting was propose in [6] by Damgård *et al.* The main concern of this work is constant-rounds solution for BD and this is achieved by realizing various constant-rounds subprotocols which are important building blocks for subsequent research including ours. What's more, as an application of BD, they also proposed a private exponentiation protocol which is the foundation of our work. Independently and concurrently, Schoenmakers and Tuyls [16] solved the problem of BD for MPC based on (Paillier) threshold homomorphic cryptosystems [4,7] and they concern mainly about efficient variations of BD for practical use. In the work [11], Nishide and Ohta proposed solutions for interval test, comparison and equality test of shared secrets without relying on the expensive BD protocol although it seems necessary. Their ideas and techniques play an important role in our work. Recently, Toft showed a novel technique that can reduce the communication complexity of BD to *almost linear* [18]. This is a very meaningful work and some key ideas of our work come from it. In a followup work, Reistad and Toft proposed a linear BD protocol [14], however, the security of this protocol is non-perfect.

As for the public modulo reduction problem (Pub-MRP), Guajardo *et al.* proposed a protocol for it in the threshold homomorphic setting without relying on BD [8]. Their protocol is very efficient (thus can be very useful for practical use) and is enlightening to this paper, however, they did not consider the general case (of Pub-MRP) where the inputs can be arbitrary size. In [12], Ning and Xu proposed a generalization of BD, and, as a simplification of their generalization, they proposed a linear protocol (without BD) for Pub-MRP which can deal with arbitrary inputs. Our work on Pub-MRP depends heavily on their work.

2 Preliminaries

In this section we introduce some important notations and known primitives.

2.1 Notations and Conventions

As mentioned above, the MPC considered in this paper is based on LSSS, such as Shamir's [15]. We denote the underlying field (of the LSSS) as $\mathbb{Z}_p$ where p is a prime with bit-length $l = \lceil \log p \rceil$. For a secret $x \in \mathbb{Z}_p$, we use $[x]_p$ to denote the sharing of x and $[x]_B = ([x_{l-1}]_p, ..., [x_1]_p, [x_0]_p)$ to denote the bitwise sharing of x. What's more, assume that there are n participants in the MPC protocol.

As in previous works, such as [6,11], we assume that the underlying LSSS allows to compute $[x + y \bmod p]_p$ from $[x]_p$ and $[y]_p$ without communication, and that it allows to compute $[xy \bmod p]_p$ from (public) $x \in \mathbb{Z}_p$ and $[y]_p$ without communication. We also assume that the LSSS allows to compute $[xy \bmod p]_p$

from $[x]_p$ and $[y]_p$ through communication among the parties and we call this procedure *secure multiplication* (or *multiplication* for simplicity). One invocation of this *multiplication* will be denoted as

$$[xy \bmod p]_p \leftarrow \text{Sec-Mult}([x]_p, [y]_p)$$

in which $[xy \bmod p]_p$ will be referred to as $[xy]_p$ for simplicity. Obviously, for MPC protocols, this multiplication protocol is a dominant factor of complexity as it involves communication. So, as in previous works, the round complexity of the (MPC) protocols is measured by the number of rounds of parallel invocations of multiplication (Sec-Mult$(\cdot)$), and the communication complexity is measured by the number of invocations of multiplication. For example, if in all a protocol involves a multiplications in parallel and then another b multiplications in parallel, then we can say that the round complexity of this protocol is 2 and the communication complexity is $a + b$ multiplications. What's more, if a procedure does not involve any secure multiplication, then it can be viewed as free and will not count for complexity. For example, if we get $[x]_B$, then $[x]_p$ can be freely obtained by a linear combination since $x = \sum_{i=0}^{l-1} x_i \cdot 2^i$.

As in [11], when we write $[C]_p$, where C is a Boolean test, it means that $C \in \{0, 1\}$ and $C = 1$ iff C is true. For example, we use $[x \stackrel{?}{=} y]_p$ to denote the output of the equality test protocol, i.e. $(x \stackrel{?}{=} y) = 1$ iff $x = y$ holds.

Given $[c]_p$, we need a protocol to reveal c, which is denoted by $c \leftarrow \text{Reveal}([c]_p)$. Note that although this protocol involves communication, it does not count for (both round and communication) complexity because the communication it involves can be carried out through a public channel.

As in [17], we will often use the *conditional selection command* below:

$$[C]_p \leftarrow [b]_p \ ? \ [A]_p \ : \ [B]_p$$

in which $A, B, C \in \mathbb{Z}_p$ and $b \in \{0, 1\}$, and which means the following:

If $b = 1$, then C is set to A; otherwise, C is set to B.

It is easy to see that this command can be realized by setting

$$[C]_p \leftarrow [b]_p([A]_p - [B]_p) + [B]_p$$

which costs only 1 round and 1 multiplication. We will frequently use this *conditional selection command* in this paper because it can make our protocols easier to be understood.

2.2 Known Primitives

We will now simply introduce some existing primitives which will be of importance later on. We refer the readers to [6,11,18] for detailed descriptions of these primitives.

- **Random Bit Protocol.** The **Random-Bit$(\cdot)$** protocol has no input and it will output a shared uniformly random bit $[b]_p$ which is unknown to all parties. In the secret sharing setting, it takes only 2 rounds and 2 multiplications [6].

- **Bitwise Less-Than Protocol.** Given two bitwise shared inputs, $[x]_B$ and $[y]_B$, the **Bit-LessThan(·)** protocol can compute a shared bit $[x \overset{?}{<} y]_p$ which identifies whether $x < y$ holds. The complexity of this protocol can be referred to as 8 rounds and $14l$ multiplications when $l \geq 36$ holds which is often the case in practice [18,12].

- **Secure Inversion Protocol.** Given a shared *non-zero* secret $[x]_p$ as input, the secure inversion protocol **Sec-Inver(·)** will output $[x^{-1} \bmod p]_p$. This protocol will cost only 2 rounds and 2 multiplications [2,6,11].

- **Unbounded Fan-In Multiplication.** In this paper, we will often need to perform the unbounded fan-in secure multiplication [2,5], i.e. given l sharings $[A_0]_p, [A_1]_p, ..., [A_{l-1}]_p$ where $A_i \in \mathbb{Z}_p^*$ for $i \in \{0, 1, ..., l-1\}$, computing a sharing $[A]_p$ where $A = \prod_{i=0}^{l-1} A_i \mod p$. By the detailed analysis in [11], we get to know that this protocol, denoted as **Sec-Prod*(·)** in this paper, can be realized in only 3 rounds and $5l$ multiplications.

- **Equal-Zero Test Protocol.** In [11], a linear protocol **Equ-Zero(·)** was proposed for testing whether a given secret $[x]_p$ is 0 or not, i.e. we have $[x \overset{?}{=} 0]_p \leftarrow$ Equ-Zero$([x]_p)$. Obviously, this protocol can also be used to test "whether two shared secrets $[x]_p$ and $[y]_p$ are equal" because "$x = y$" $\Leftrightarrow$ "$(x - y) = 0$". The complexity of this protocol is 8 rounds and $81l$ multiplications.

- **Generation of Bitwise Shared Random Value.** This protocol, denoted by **Solved-Bits(·)**, has no input and can output a bitwise shared random integer $[r]_B$ satisfying $r < p$. The complexity of this protocol can be referred to as 7 rounds and $56l$ multiplications when $l \geq 36$ [18].

- **Bit-Decomposition (BD).** In the secret sharing setting, the function of BD can be described as converting $[x]_p$ to $[x]_B$, i.e. we have $[x]_B \leftarrow \text{BD}([x]_p)$ [6,18]. To the best of our knowledge, currently the most efficient version of BD (*with perfect security*) was proposed in [18], whose complexity can be referred to as 23 rounds and $76l + 31l \log l$ multiplications when $l \geq 36$; in the text when analyzing the complexities of (exponentiation) protocols involving BD, we will refer to the complexity of BD as above. We note that [18] also proposed a BD protocol with *almost-linear* communication complexity (i.e. $O(l \log^* l)$ multiplications or even lower). This is of course a very meaningful work. However, inevitably the round complexity of this version of BD is relatively high and thus for obtaining (private exponentiation) protocols with close and comparable round complexities, (as well as for notational convenience) we do not referred to this BD protocol in detail in the text. (Although we focus mainly on the communication complexity of protocols, the round complexity should also be considered.) We also note that in [14], a linear BD is proposed, however, the security of this BD protocol is (at most) statistical; so in the text we will not refer to this BD protocol in detail neither, because we focus on protocols with perfect security.

3 Multi-party Computation for Private Exponentiation with BD

In [6], a constant-rounds private exponentiation protocol was constructed with the help of BD. This protocol is the foundation of our work and in our exponentiation protocol, we need to use the sub-protocols of it. So, in this section, we describe in detail this private exponentiation protocol *with* BD. We will first introduce two important sub-protocols of it, i.e. the *public exponentiation protocol* and the *bit exponentiation protocol*. All the protocols in this section are re-descriptions of the ones in [6] but with detailed analysis.

3.1 The Public Exponentiation Protocol

With a shared *non-zero* value $[x]_p$ (i.e. $x \in \mathbb{Z}_p^*$) and a public value $a \in \mathbb{Z}$ as inputs, the *public exponentiation protocol*, Pub-Expo$(\cdot)$, can compute $[x^a]_p$. The details are presented in Figure 1. Generally speaking, this protocol is a slightly improved version of the one in [6].

Protocol $[x^a]_p \leftarrow$ **Pub-Expo**$([x]_p, a)$

This protocol requires that $x \neq 0$.

1. Every party P_i $(i \in \{1, 2, ..., n\})$ picks a random integer $r_i \in \mathbb{Z}_p^*$ and computes r_i^{-a}. Then P_i shares r_i and r_i^{-a} between the parties, i.e. the parties get $[r_i]_p$ and $[r_i^{-a}]_p$.
2. The parties compute
   $[r]_p \leftarrow$ Sec-Prod$^*([r_1]_p, [r_2]_p, ..., [r_n]_p)$
   $[r^{-a}]_p \leftarrow$ Sec-Prod$^*([r_1^{-a}]_p, [r_2^{-a}]_p, ..., [r_n^{-a}]_p)$
3. $[xr]_p \leftarrow$ Sec-Mult$([x]_p, [r]_p)$
4. $xr \leftarrow$ Reveal$([xr]_p)$
5. **Return** $[x^a]_p = (xr)^a \cdot [r^{-a}]_p$

Fig. 1. The Public Exponentiation Protocol

As for the correctness, notice that in Step 4 we need to reveal the value of xr where r is non-zero, and it is easy to see that $xr = 0 \Leftrightarrow x = 0$. This is just why this protocol requires $x \neq 0$: if $x = 0$, then the parties will get to know this in this step. Also note that in this protocol the public exponent $(-a)$ is the additive inverse of a in the sense of mod $(p-1)$ *rather than* mod p.

Privacy is straightforward.

The complexity will be discussed in two cases: the semi-honest case and the malicious case. The difference between these two cases lies in Step 1 where every party P_i $(i \in \{1, 2, ..., n\})$ is required to distribute two sharings, $[r_i]_p$ and $[r_i^{-a}]_p$, between the parties. Below we will first analyze the complexity of this step. Before going on, recall the well-known fact that when considering the communication complexity of MPC protocols (in the LSSS setting), 1 invocation of the secure multiplication is equivalent to distributing n sharings between the n parties

and thus the communication complexity of distributing 1 sharing (between the n parties) can be viewed as $\frac{1}{n}$ multiplications.

In the semi-honest case, all the n parties follow the protocol, so every party distributes 2 sharings between all the n parties, thus the complexity of Step 1 is 1 round and $\frac{2}{n} \cdot n = 2$ multiplications.

In the malicious case, as mentioned in [6], the complexity is much higher because we need to involve the *cut-and-choose* technique to make the protocol robust. Specifically, besides $[r_i]_p$ and $[r_i^{-a}]_p$, every party P_i ($i \in \{1, 2, ..., n\}$) is required to distribute another two sharings $[s_i]_p$ and $[s_i^{-a}]_p$. Then the parties involve the Random-Bit$(\cdot)$ protocol to jointly form a shared random bit $[b_i]_p$ and open it. Then they open $([s_i]_p, [s_i^{-a}]_p)$ or compute and open $([s_i r_i]_p, [s_i^{-a} r_i^{-a}]_p)$ according to the value of b_i and then verify that the first value is non-zero and that the second value is the $(-a)$'th power of the first. We call the above process *one instance of cut-and-choose*. For every party P_i ($i \in \{1, 2, ..., n\}$), to get a lower error probability, we can repeat the above process k (which satisfies $k \geq 1$ and which will be referred to as "*the security parameter for cut-and-choose*") times in parallel, leading to an error probability 2^{-k}. Then we can say that in all we need kn instances of cut-and-choose in parallel. As for the complexity of *one instance*, we notice the following facts: distributing $[s_i]_p$ and $[s_i^{-a}]_p$ between the parties involves $\frac{2}{n}$ multiplications; the generation of $[b_i]_p$ involves 2 rounds and 2 multiplications and can be scheduled in parallel with the process of distributing $[s_i]_p$ and $[s_i^{-a}]_p$; the computation of $([s_i r_i]_p, [s_i^{-a} r_i^{-a}]_p)$ involves 1 round and 2 multiplications and, obviously, on average we need only to compute $([s_i r_i]_p, [s_i^{-a} r_i^{-a}]_p)$ once every 2 instants of cut-and-choose because b_i is a uniformly random bit. So, on average, the complexity of one instance is (at most) $2 + 1 = 3$ rounds and $\frac{2}{n} + 2 + 2 \cdot \frac{1}{2} = \frac{2}{n} + 3$ multiplications. Recall that in all we need kn parallel instances of cut-and-choose. What's more, notice that the process of cut-and-choose can be scheduled in parallel with the process of distributing $[r_i]_p$ and $[r_i^{-a}]_p$. So, in the malicious case, the complexity of Step 1 is 3 rounds and $2 + kn \cdot (\frac{2}{n} + 3) = 2 + 2k + 3kn$ multiplications.

Then it is easy to see that, in the semi-honest case, the overall complexity of this Pub-Expo$(\cdot)$ protocol is $R_{pub} \triangleq R_{pub}^{s-h} = 1 + 3 + 1 = 5$ rounds and $C_{pub} \triangleq C_{pub}^{s-h} = 2 + 5n \cdot 2 + 1 = 10n + 3$ multiplications; in the malicious case, the overall complexity is $R_{pub} \triangleq R_{pub}^{mal} = 3 + 3 + 1 = 7$ rounds and $C_{pub} \triangleq C_{pub}^{mal} = (2 + 2k + 3kn) + 5n \cdot 2 + 1 = 3kn + 10n + 2k + 3$ multiplications[1]. Recall that n denotes the number of the parties and k is the security parameter for cut-and-choose. Hereafter, we will generally refer to the complexity of this protocol as R_{pub} rounds and C_{pub} multiplications. The values of R_{pub} and C_{pub}

[1] Thanks to one of the anonymous reviewers, we get to realize that (seen in isolation) we can combine Step 2 and Step 3 (in Figure 1 by viewing $[x]_p$ as one of the inputs of the Sec-Prod$^*()$ protocol for computing $[r]_p$) to save 1 round; this is of course a meaningful improvement for a "constant-rounds" protocol. However, considering the parallel invocations of this Pub-Expo() protocol (e.g. in the forthcoming protocol in Figure 5), we still separate these two steps (Step 2 and Step 3) when analyzing the complexity.

are determined by the adversaries considered; moreover, we can say that both R_{pub} and C_{pub} can be viewed as *constants* because they are independent from (the input length) l.

3.2 The Bit Exponentiation Protocol

With a shared *non-zero* value $[x]_p$ and a bitwise shared value $[a]_B = ([a_{l-1}]_p, ..., [a_1]_p, [a_0]_p)$ as inputs, the *bit exponentiation protocol*, Bit-Expo$(\cdot)$, can compute $[x^a]_p$. The details are seen in Figure 2.

Protocol $[x^a]_p \leftarrow$ **Bit-Expo**$([x]_p, [a]_B)$
This protocol requires that $x \neq 0$.

1. For $i = 0, 1, ..., l-1$ in parallel: $[A_i]_p \leftarrow$ Pub-Expo$([x]_p, 2^i)$
2. For $i = 0, 1, ..., l-1$ in parallel: $[B_i]_p \leftarrow [a_i]_p \ ? \ [A_i]_p \ : \ 1$
3. **Return** $[x^a]_p \leftarrow$ Sec-Prod$^*([B_{l-1}]_p, ..., [B_1]_p, [B_0]_p)$

Fig. 2. The Bit Exponentiation Protocol

Correctness and privacy is straightforward. The complexity of this protocol is $R_{pub} + 1 + 3 = R_{pub} + 4$ rounds and $C_{pub} \cdot l + l + 5l = (C_{pub} + 6)l$ multiplications.

3.3 The Private Exponentiation Protocol with BD

Here we come to the *private exponentiation protocol* relying on BD proposed in [6], which will be denoted by Pri-Expo-BD$(\cdot)$. Given two shared inputs $[x]_p$ and $[a]_p$, Pri-Expo-BD$(\cdot)$ will output $[x^a]_p$. This time, both x and a can be arbitrary values in $\mathbb{Z}_p$. See Figure 3 for the details.

Protocol $[x^a]_p \leftarrow$ **Pri-Expo-BD**$([x]_p, [a]_p)$

1. $[b]_p \leftarrow$ Equ-Zero$([x]_p)$
2. $[\tilde{x}]_p = [x]_p + [b]_p$
3. $[a]_B \leftarrow$ BD$([a]_p)$
4. $[\tilde{x}^a]_p \leftarrow$ Bit-Expo$([\tilde{x}]_p, [a]_B)$
5. **Return** $[x^a]_p = [\tilde{x}^a]_p - [b]_p$

Fig. 3. The Private Exponentiation Protocol *with* BD

As for the correctness, notice that $b = (x \stackrel{?}{=} 0)$ and that $[\tilde{x}]_p = [x]_p + [x \stackrel{?}{=} 0]_p$ is always non-zero and thus can be given to Bit-Expo$(\cdot)$ as the first input. What's more, it can be easily verified that $[x^a]_p = [\tilde{x}^a]_p - [x \stackrel{?}{=} 0]_p$ always holds no matter x is 0 or not. Using $\tilde{x}$ to substitute x to perform the protocol is in fact the "exception trick" proposed in [6] for handling the special case where $x = 0$.

Privacy follows readily from only using private sub-protocols.

The overall complexity of this protocol is $23 + (R_{pub} + 4) = R_{pub} + 27$ rounds and $81l + (76l + 31l \log l) + ((C_{pub} + 6)l) = 163l + C_{pub} \cdot l + 31l \log l$ multiplications. See [13] for the detailed complexity analysis.

4 Linear Multi-party Computation for Private Exponentiation

In this section, we propose a private exponentiation protocol with constant round complexity and linear communication complexity. Specifically, we will first show how to remove the invocation of BD to get a protocol with linear communication complexity. Then we will further improve this linear protocol to reduce the communication complexity considerably.

4.1 The Private Exponentiation Protocol without BD

See Figure 4 for our private exponentiation protocol *without* BD which will be denoted as Pri-Expo$(\cdot)$.

Protocol $[x^a]_p \leftarrow$ **Pri-Expo**$([x]_p, [a]_p)$

1. $[b]_p \leftarrow$ Equ-Zero$([x]_p)$
2. $[\tilde{x}]_p = [x]_p + [b]_p$
3. $[r]_B \leftarrow$ Solved-Bits() $\quad \triangleright$ Recall that $[r]_B$ implies $[r]_p$.
4. $[c]_p = [a]_p + [r]_p$
5. $c \leftarrow$ Reveal$([c]_p)$ $\quad \triangleright\ c = a + r \mod p$
6. $[C]_p \leftarrow$ Pub-Expo$([\tilde{x}]_p, c)$ $\quad \triangleright\ C = \tilde{x}^c \mod p$
7. $[C']_p \leftarrow$ Sec-Mult$([C]_p, [\tilde{x}]_p)$ $\quad \triangleright\ C' = C \cdot \tilde{x} = \tilde{x}^{c+1} = \tilde{x}^{c+1+\varphi(p)} = \tilde{x}^{c+p} \mod p$
8. $[f]_p \leftarrow$ Bit-LessThan$(c, [r]_B)$
9. $[\tilde{C}]_p \leftarrow [f]_p\ ?\ [C']_p\ :\ [C]_p$ $\quad \triangleright\ \tilde{C} = \tilde{x}^{a+r} \mod p$
10. $[R]_p \leftarrow$ Bit-Expo$([\tilde{x}]_p, [r]_B)$
11. $[R^{-1}]_p \leftarrow$ Sec-Inver$([R]_p)$ $\quad \triangleright\ R^{-1} = \tilde{x}^{-r} \mod p$
12. $[\tilde{x}^a]_p \leftarrow$ Sec-Mult$([\tilde{C}]_p, [R^{-1}]_p)$
13. **Return** $[x^a]_p = [\tilde{x}^a]_p - [b]_p$

Fig. 4. The Private Exponentiation Protocol *without* BD

Correctness: As for the correctness, similar to the Pri-Expo-BD$(\cdot)$ protocol (in Figure 3), we use the non-zero $[\tilde{x}]_p$ to substitute $[x]_p$ to perform the main process. The main idea of this protocol is as follows.

First we compute $[\tilde{C}]_p = [\tilde{x}^{a+r}]_p$. Notice that we have $c = a + r \mod p$ and there are two cases: no wrap-around mod p occurs or there is a wrap-around. In the former case, $a + r = c$ holds over the integers (or we can say "$a + r = c$ holds *unconditionally*") and then we have $c \geq r$ because $a \geq 0$; similarly, in

the latter case, $a + r = c + p$ holds over the integers (i.e. $a + r = c + p$ holds *unconditionally*) and then we have $c < r$ because $c = r + (a - p)$ and $a < p$. So, for computing $[\tilde{x}^{a+r}]_p$, we can compute both of the two possible values of it, $[\tilde{x}^c]_p$ and $[\tilde{x}^{c+p}]_p$, and then *select* the correct one; this selection can be carried out by testing whether $c < r$ holds. What's more, when computing $[\tilde{x}^c]_p$ we need to involve the Pub-Expo(·) protocol; however, this is not necessary when computing $[\tilde{x}^{c+p}]_p$ because we have: $\tilde{x}^{c+p} = \tilde{x}^{c+p-(p-1)} = \tilde{x}^{c+1} = \tilde{x}^c \cdot \tilde{x} \mod p$.

Then, in the following steps, after getting $[R]_p = [\tilde{x}^r]_p$ using the Bit-Expo(·) protocol, we can obtain $[\tilde{x}^a]_p$ based on the simple fact $\tilde{x}^a = \tilde{x}^{a+r} \cdot (\tilde{x}^r)^{-1} \mod p$. Then finally $[x^a]_p$ can be easily obtained.

Privacy: Privacy is straightforward.

Complexity: As for the complexity, both in the semi-honest case and the malicious case, the complexity of this protocol (Pri-Expo(·)) can be referred to as $8 + (R_{pub} + 6) + 1 = R_{pub} + 15$ rounds and
$81l + 56l + C_{pub} + 1 + 14l + 1 + (C_{pub} + 6)l + 2 + 1 = 157l + C_{pub} \cdot l + C_{pub} + 5$ multiplications (See [13] for the detailed complexity analysis). Recall that both R_{pub} and C_{pub} can be viewed as *constants*, so this is a constant-rounds protocol with linear communication complexity. Compared with the (perfectly secure) Pri-Expo-BD(·) protocol proposed in [6] (whose complexity is $R_{pub} + 27$ rounds and $163l + C_{pub} \cdot l + 31l \log l$ multiplications), our protocol has a lower round complexity and a significantly lower communication complexity.

4.2 A Further Improvement

In this section, we make a further improvement of our Pri-Expo(·) protocol above by improving one of the sub-protocols of it, Bit-Expo(·), which is often the dominate factor of the communication complexity. The improved version of Pri-Expo(·) and Bit-Expo(·) will be denoted as Pri-Expo$^+$(·) and Bit-Expo$^+$(·) respectively. Generally speaking, by replacing the invocation of Bit-Expo(·) with Bit-Expo$^+$(·) in our Pri-Expo(·) protocol, we get our further improved private exponentiation protocol: Pri-Expo$^+$(·). The details are presented below.

In our Pri-Expo(·) protocol (in Figure 4), Bit-Expo(·) is a very important sub-protocol. Recall that the communication complexity of this sub-protocol is $(C_{pub} + 6)l$ multiplications; what's more, in the semi-honest case $C_{pub} = C_{pub}^{s-h} = 10n + 3$, and in the malicious case $C_{pub} = C_{pub}^{mal} = 3kn + 10n + 2k + 3$ (see Section 3.1). In many cases, Bit-Expo(·) is relatively expensive. For example, in the malicious case, if we set $n = 20$ and $k = 10$, then the communication complexity of Bit-Expo(·) will be $(C_{pub}^{mal} + 6)l = 829l$ multiplications; at the same time, the communication complexity of the (*whole*) Pri-Expo(·) protocol is $(C_{pub}^{mal} + 157)l + C_{pub}^{mal} + 5 = 980l + 828$ multiplications. So we can see that, in this case, Bit-Expo(·) is obviously a dominate factor of the communication complexity of Pri-Expo(·). So, reducing the communication complexity of Bit-Expo(·) is very meaningful.

The communication complexity of Bit-Expo(·) comes mainly from the l invocations of Pub-Expo(·) which is non-trivial (See Figure 2 and Figure 1). Here we

show a technique that can reduce the number of invocations (of Pub-Expo$(\cdot)$) to $2\sqrt{l}$ (with slight increase in round complexity) and thus reduce the communication complexity significantly. The main idea is presented below.

Consider the case that we want to compute x^a. We divide the given exponent $a = (a_{l-1}, ..., a_1, a_0)$ into s blocks, each of which contains t bits. Obviously, we have $s \cdot t = l$ and $1 \leq s, t \leq l$. We denote the i'th block of a as $a_i^{s\times t}$ for $i \in \{0, 1, ..., s-1\}$, and denote the j'th bit of the i'th block as $a_{i,j}^{s\times t}$ for $j \in \{0, 1, ..., t-1\}$. That is to say, we have

$$\begin{aligned} a &= (a_{l-1}, ..., a_1, a_0) = \left(a_{s-1}^{s\times t}, ..., a_1^{s\times t}, a_0^{s\times t}\right) \\ &= \Big(\left(a_{s-1,t-1}^{s\times t}, ..., a_{s-1,1}^{s\times t}, a_{s-1,0}^{s\times t}\right), ..., \left(a_{1,t-1}^{s\times t}, ..., a_{1,1}^{s\times t}, a_{1,0}^{s\times t}\right), \left(a_{0,t-1}^{s\times t}, ..., a_{0,1}^{s\times t}, a_{0,0}^{s\times t}\right) \Big) \end{aligned}$$

Obviously, $a_i^{s\times t}$ can be viewed as the i'th digit of the base-2^t form of a. What's more, we have $a_{i,j}^{s\times t} = a_{i\cdot t+j}$. Now we have the following equations:

$$\begin{aligned} x^a &= x^{\sum_{i=0}^{s-1} a_i^{s\times t}\cdot\left(2^t\right)^i} = \prod_{i=0}^{s-1} x^{a_i^{s\times t}\cdot\left(2^t\right)^i} = \prod_{i=0}^{s-1} \left(x^{a_i^{s\times t}}\right)^{\left(2^t\right)^i} = \prod_{i=0}^{s-1} \left(x^{\sum_{j=0}^{t-1} a_{i,j}^{s\times t}\cdot 2^j} \right)^{\left(2^t\right)^i} \\ &= \prod_{i=0}^{s-1} \left(\prod_{j=0}^{t-1} x^{a_{i,j}^{s\times t}\cdot 2^j} \right)^{\left(2^t\right)^i} = \prod_{i=0}^{s-1} \left(\prod_{j=0}^{t-1} \left(x^{2^j}\right)^{a_{i,j}^{s\times t}} \right)^{\left(2^t\right)^i} \\ &= \prod_{i=0}^{s-1} \left(\prod_{j=0}^{t-1} \left(a_{i,j}^{s\times t} \ ?\ x^{2^j} \ :\ 1\right) \right)^{\left(2^t\right)^i} \end{aligned}$$

Based on the above facts, we propose our improved Bit-Expo$(\cdot)$ protocol, Bit-Expo$^+(\cdot)$, which is presented in Figure 5. Note that in Figure 5, for the convenience of the forthcoming discussions, the two variables, s and t, are not assigned. We will discuss how to assign them when analyzing the complexity of this protocol.

Protocol $[x^a]_p \leftarrow$ **Bit-Expo**$^+([x]_p, [a]_B)$
This protocol requires that $x \neq 0$.

1. For $j = 0, 1, ..., t-1$ in parallel: $[A_j]_p \leftarrow$ Pub-Expo$([x]_p, 2^j)$
2. For $i = 0, 1, ..., s-1$ in parallel do
   For $j = 0, 1, ..., t-1$ in parallel: $[B_{i,j}]_p \leftarrow [a_{i,j}^{s\times t}]_p$? $[A_j]_p$: 1
   $[B_i]_p \leftarrow$ Sec-Prod$^*([B_{i,0}]_p, [B_{i,1}]_p, ..., [B_{i,t-1}]_p)$
   $[C_i]_p \leftarrow$ Pub-Expo$\left([B_i]_p, (2^t)^i\right)$
   End for
3. **Return** $[x^a]_p \leftarrow$ Sec-Prod$^*([C_0]_p, [C_1]_p, ..., [C_{s-1}]_p)$

Fig. 5. The *Improved* Bit Exponentiation Protocol

Correctness and privacy is straightforward. As for the complexity, notice that there are invocations of Pub-Expo($\cdot$) in both Step 1 and Step 2. One important point is, these two places of invocations can be scheduled *partially in parallel*. Specifically, when the invocations (of Pub-Expo($\cdot$)) in Step 1 are proceeding with the first two steps of Pub-Expo($\cdot$) (See Figure 1), the invocations in Step 2 can also proceed with them. That is to say, although these two places of invocations can *not* be scheduled (completely) in parallel, they will cost only 1 more round than one single invocation (note that Step 3 through Step 5 in Pub-Expo($\cdot$) (Figure 1) involve only 1 multiplication). So, the complexity of this protocol is $R_{pub}+1+3+1+3 = R_{pub}+8$ rounds and $C_{pub}\cdot t+t\cdot s+5t\cdot s+C_{pub}\cdot s+5s \leq C_{pub}\cdot(s+t)+11l$ multiplications. (Recall that $s\cdot t = l$ and $1 \leq s \leq l$.)

It remains to assign concrete values to s and t. Note that we have "$s+t \geq 2\sqrt{s\cdot t} = 2\sqrt{l}$" and "$s+t = 2\sqrt{l}$ iff $s = t = \sqrt{l}$". So we should set $s = t = \sqrt{l}$, because in this case the communication complexity of this Bit-Expo$^+(\cdot)$ protocol will be the lowest, i.e. $C_{pub}\cdot 2\sqrt{l}+11l$ multiplications. Then, if we replace the invocation of Bit-Expo($\cdot$) in our Pri-Expo($\cdot$) protocol (in Figure 4) with the Bit-Expo$^+(\cdot)$ protocol here, we will get an improved private exponentiation protocol (denoted as Pri-Expo$^+(\cdot)$) whose complexity is $R_{pub}+19$ rounds and $162l+C_{pub}\cdot 2\sqrt{l}+C_{pub}+5$ multiplications. Compared with the Pri-Expo-BD($\cdot$) protocol in [6] (whose complexity is $R_{pub}+27$ rounds and $163l+C_{pub}\cdot l+31l\log l$ multiplications), our Pri-Expo$^+(\cdot)$ protocol reaches lower round complexity and much lower communication complexity. What's more, we can say that, the larger C_{pub} is (which implies larger n and k), the greater advantage our protocol has. For systems with relatively more participants, higher security requirements and longer input length (i.e. l), our protocol can be of overwhelming advantage. (See Appendix A (Table 1) for an overview.)

See [13] for some further discussions.

5 Further Generalization of BD and Improved Solution for Public Modulo Reduction

In this section, we propose a further generalization of BD and an improved solution for Pub-MRP. The work in this section depends *heavily* on the work in [12] which we'd strongly recommend the readers to read before going on.

Given a sharing of secret x, BD allows the parties to extract the shared *base-2 form* of x in constant rounds. In the work [12], Ning and Xu show us a generalization of BD which is named as "Base-m Digit-Decomposition" (or "Base-m Digit-Bit-Decomposition") and which can extract the shared (or bitwise shared) *base-m form* of x in constant rounds. We note that their generalization can be further generalized to a "*Hybrid-base Digit-Decomposition*" (or "*Hybrid-base Digit-Bit-Decomposition*") protocol which can extract the shared (or bitwise shared) *hybrid-base form* of x; here *hybrid-base* means the base of every digit can be different. For example, if we denote

"9 days 23 hours 59 minutes 59 seconds"

(which could be the "Time Left" before the submission deadline of this

conference) as

$$x = \boxed{\mathbf{9}}\boxed{\mathbf{23}}\boxed{\mathbf{59}}\boxed{\mathbf{59}}$$

then x can be used to represent the *total* seconds (left) and can be viewed as a hybrid-base integer with bases (from left to right) 10, 24, 60, 60. Here the left-most base (i.e. "10") can be set as we wish, but other bases are fixed.

Below we discuss the relationship between "the value of an integer" and "the bases" in another point of view. Specifically, we list 3 cases below.

1. **Getting the base-2 form of** $x \in \mathbb{Z}_p$
   In this case, we get $l = \lceil \log p \rceil$ bits $x_i \in \{0, 1\}$ for $i \in \{0, 1, ..., l-1\}$ satisfying

$$x = \sum_{i=0}^{l-1} \left(x_i \cdot 2^i \right)$$

2. **Getting the base-**m **form of** $x \in \mathbb{Z}_p$
   Similarly, in this case, for the given base $m \geq 2$, we get $l^{(m)} = \lceil \log_m p \rceil$ digits $x_i^{(m)} \in \{0, 1, ..., m-1\}$ for $i \in \{0, 1, ..., l^{(m)} - 1\}$ satisfying

$$x = \sum_{i=0}^{l^{(m)}-1} \left(x_i^{(m)} \cdot m^i \right)$$

3. **Getting the hybrid-base form of** $x \in \mathbb{Z}_p$
   Given an $l^{(M)}$ size "array of bases" $M[\,] = \left[m_{l^{(M)}-1}, ..., m_1, m_0 \right]$ satisfying $m_i \geq 2$ for $i \in \{0, 1, ..., l^{(M)} - 1\}$ and $\prod_{i=0}^{l^{(M)}-2} m_i < p < \prod_{i=0}^{l^{(M)}-1} m_i$, we get $l^{(M)}$ digits $x_i^{(M)} \in \{0, 1, ..., m_i - 1\}$ for $i \in \{0, 1, ..., l^{(M)} - 1\}$ satisfying the following equation (*in which we set* $m_{-1} = 1$)

$$x = \sum_{i=0}^{l^{(M)}-1} \left(x_i^{(M)} \cdot \prod_{j=-1}^{i-1} m_j \right)$$

   Here, we call $\left(x_{l^{(M)}-1}^{(M)}, ..., x_1^{(M)}, x_0^{(M)} \right)$ "*the hybrid-base form of* x *defined by* $M[\,]$".

It is easy to see that in the hybrid-base case (i.e. Case 3) if we set the "array of bases" $M[\,]$ to be $[m, ..., m, m]$ where $l^{(M)} = l^{(m)}$, then we will get the base-m case (i.e. Case 2); if we set $M[\,] = [2, ..., 2, 2]$ where $l^{(M)} = l$, we will get the base-2 case (i.e. Case 1).

Given a shared secret $[x]_p$ and an "array of bases" $M[\,]$, our "Hybrid-base Digit-Decomposition" (or "Hybrid-base Digit-Bit-Decomposition") protocol, whose asymptotic complexity is $O(1)$ rounds and $O(l^{(M)} \log l^{(M)} + l)$ [2] (or $O(l \log l)$) multiplications, can output the shared (or bitwise shared) hybrid-base

[2] This term was mistakenly written as $O(l^{(M)} \log l^{(M)})$ in the submission; thanks to one of the anonymous reviewers for pointing this out.

form of x defined by $M[\,]$ (i.e. the sharings (or bitwise sharings) of all the digits $x_i^{(M)}$ for $i \in \{0, 1, ..., l^{(M)} - 1\}$) which will be referred to as $[x]_D^M$ (or $[x]_{D,B}^M$). That is to say, we have

$$[x]_D^M \leftarrow \text{Hybrid-Base-Digit-Decomposition}([x]_p, M[\,]);$$
$$[x]_{D,B}^M \leftarrow \text{Hybrid-Base-Digit-Bit-Decomposition}([x]_p, M[\,]).$$

The intuition behind our further generalization is similar to that of the generalization of BD in [12]. Specifically, as shown in [12], for getting the shared (or bitwise shared) base-m form of x, we need to randomize $[x]_p$ using a jointly generated random integer r whose bitwise shared base-m form is known to the parties; that is to say, the parties generate an array of bitwise shared base-m digits to form r; here, a *base-m digit* is in fact a non-negative integer less than m and the details of generating such a (bitwise shared) digit can be seen in [12] (the Random-Digit-Bit$(\cdot)$ protocol). Similarly, to obtain the shared (or bitwise shared) hybrid-base form of x (which is defined by $M[\,]$), we should randomize $[x]_p$ using a (jointly generated random) integer r^+ whose bitwise shared hybrid-base form (*which is also defined by* $M[\,]$) is known to the parties. This is the key idea of our further generalization and is also the key difference between the generalization in [12] and our further generalization.

More importantly, as a simplification of our "Hybrid-base Digit-Decomposition" protocol, we can get an *improved* public modulo reduction protocol (denoted as Pub-MR$^+(\cdot)$ here) which is more efficient than the one in [12] (denoted as Pub-MR$(\cdot)$). Specifically, in [12], for solving Pub-MRP (i.e. computing $[x \bmod m]_p$ from $[x]_p$ and $m \in \{2, 3, ..., p-1\}$), Ning and Xu view this problem as extracting only (the sharing of) the least significant base-m digit of x, and thus their modulo reduction protocol (i.e. Pub-MR$(\cdot)$) can be viewed as a simplification of their "Base-m Digit-Decomposition" protocol (which extracts (the sharings of) all the base-m digits of x). In another point of view, we can say that they set $M[\,] = [m, ..., m, m]$ and extract only (the sharing of) $x_0^{(M)}$ (see Case 3 above). This is of course correct because in this case we have $x = \sum_{i=0}^{l^{(M)}-1} \left(x_i^{(M)} \cdot m^i\right)$. However, this is not a must, and, enlightened by [11] and [8], we find that, by setting $M[\,] = [2, ..., 2, 2, m]$ where $l^{(M)} = \left\lceil \log \left\lfloor \frac{p}{m} \right\rfloor \right\rceil + 1$, we can also solve Pub-MRP because in this case we have

$$x = \sum_{i=1}^{l^{(M)}-1} \left(x_i^{(M)} \cdot 2^{i-1} \cdot m\right) + x_0^{(M)}$$

and thus $x_0^{(M)} = (x \bmod m)$. That is to say, in the case where $M[\,] = [2, ..., 2, 2, m]$, if we extract only (the sharing of) the least significant digit of x, which can be viewed as a simplification of our "Hybrid-base Digit-Decomposition", we can also get $[x \bmod m]_p$; this public modulo reduction protocol is just Pub-MR$^+(\cdot)$.

Comparison: Below we show the advantage of our Pub-MR$^+(\cdot)$ protocol over Pub-MR$(\cdot)$. Similar to the generalization and further generalization of BD, when computing $[x \bmod m]_p$ from $[x]_p$ and m, both Pub-MR$(\cdot)$ and our Pub-MR$^+(\cdot)$

need to use a jointly generated random integer to randomize $[x]_p$ [12]. The key difference between these two (modulo reduction) protocols lies just in the generation of this random integer. Specifically, in the Pub-MR$(\cdot)$ protocol, the random integer needed, denoted as r here, should be of a "hybrid-base" form defined by $M[\,] = [m, ..., m, m]$ where $l^{(M)} = l^{(m)}$; whereas in our Pub-MR$^+(\cdot)$ protocol, the random integer needed, denoted as r^+, should be of a hybrid-base form defined by $M[\,] = [2, ..., 2, 2, m]$ where $l^{(M)} = \left\lceil \log \left\lfloor \frac{p}{m} \right\rfloor \right\rceil + 1$. Then obviously, in Pub-MR$(\cdot)$ when generating r, we need to generate $l^{(M)} = l^{(m)}$ (bitwise shared) base-m digits, whereas in our Pub-MR$^+(\cdot)$ when generating r^+, we need only to generate 1 such base-m digit. This is just the advantage of our improved public modulo reduction protocol; reducing the demand for such base-m digits is very meaningful because the generation of them is a non-trivial work. Specifically, when m is a non-power of 2, roughly speaking the generation of 1 such digit will cost 8 rounds and $64L(m)$ multiplications where $L(m) \triangleq \lceil \log m \rceil$ denotes the bit-length of m [12].

Complexity: Finally, we conclude that, the complexity of our Pub-MR$^+(\cdot)$ protocol is 22 rounds and (about) $78l + 276L(m)$ multiplications. Compared with Pub-MR$(\cdot)$ (whose complexity is 22 rounds and (about) $326l + 28L(m)$ multiplications), we can see that, for relatively small m (thus $L(m)$ is very small), the communication complexity is reduced considerably. For example, in the case where $l = 256, m = 100$ (then $L(m) = \lceil \log 100 \rceil = 7$), the communication complexity is reduced by a factor of approximately 3.8.

6 Discussions

We note that using the ideas in [11], the round complexity of our private exponentiation protocols (as well as our public modulo reduction protocol) can be improved; the method is to use preprocessing, i.e. moving all the generation of (shared) random values (e.g. the invocations of Random-Bit$(\cdot)$) to the beginning of the whole protocol. In the analysis of the round complexity of our protocols, we simply ignore this for clarity.

An interesting point is that the communication complexity of our "Hybrid-base Digit-Decomposition" protocol and "Hybrid-base Digit-Bit-Decomposition" protocol can be reduced to "almost linear" using the techniques of [18]. Specifically, the only non-linear part of these two protocols is the computation of a *prefix*-$\circ$ [6,12]; and the techniques proposed in [18], which is used to reduce the complexity of the only non-linear part of BD (computation of a *postfix-comparison*) to "almost linear", can also be used to reduce the complexity of the computation of this *prefix*-$\circ$ to "almost linear".

7 Future Work

In our private exponentiation protocol, we need an important sub-protocol called *public exponentiation protocol* (i.e. Pub-Expo$(\cdot)$) for computing $[x^a \bmod p]_p$ from $[x]_p$ and a public a. A problem is, the communication complexity of this sub-protocol is relatively high and more importantly, the communication complexity

depends on n and k (see Section 3.1 for the details). We leave it an open problem to construct more efficient protocols for this problem and protocols with communication complexity independent from n and k would be most welcome. What's more, in our private exponentiation protocol when involving Pub-Expo$(\cdot)$, the second input (i.e. the public a in Figure 1) is (almost) always a power of 2. So designing more efficient public exponentiation protocols for this special case is also meaningful.

Acknowledgments. We would like to thank the anonymous reviewers for their careful work and helpful comments.

References

1. Algesheimer, J., Camenisch, J.L., Shoup, V.: Efficient Computation Modulo A Shared Secret with Application to the Generation of Shared Safe-Prime Products. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 417–432. Springer, Heidelberg (2002)
2. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In: 8th ACM Symposium on Principles of Distributed Computing, pp. 201–209. ACM Press, New York (1989)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations. In: 20th Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM Press, New York (1988)
4. Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty Computation from Threshold Homomorphic Encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)
5. Chandra, A.K., Fortune, S., Lipton, R.J.: Unbounded Fan-In Circuits and Associative Functions. In: 15th Annual ACM Symposium on Theory of Computing, pp. 52–60. ACM Press, New York (1983)
6. Damgård, I.B., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)
7. Damgård, I.B., Nielsen, J.B.: Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003)
8. Guajardo, J., Mennink, B., Schoenmakers, B.: Modulo Reduction for Paillier Encryptions and Application to Secure Statistical Analysis. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 375–382. Springer, Heidelberg (2010)
9. Goldreich, O., Micali, S., Wigderson, A.: How to Play Any Mental Game or A Complete Theorem for Protocols with Honest Majority. In: 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press, New York (1987)
10. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified Vss and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In: 17th ACM Symposium on Principles of Distributed Computing, pp. 101–110. ACM Press, New York (1998)
11. Nishide, T., Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)

12. Ning, C., Xu, Q.: Multiparty Computation for Modulo Reduction without Bit-Decomposition and A Generalization to Bit-Decomposition. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 483–500. Springer, Heidelberg (2010)
13. Ning, C., Xu, Q.: Constant-Rounds, Linear Multi-party Computation for Exponentiation and Modulo Reduction with Perfect Security. Cryptology ePrint Archive, Report 2011/069 (2011), http://eprint.iacr.org/2011/069
14. Reistad, T., Toft, T.: Linear, Constant-Rounds Bit-Decomposition. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 245–257. Springer, Heidelberg (2010)
15. Shamir, A.: How to Share A Secret. Communications of the ACM 22(11), 612–613 (1979)
16. Schoenmakers, B., Tuyls, P.: Efficient Binary Conversion for Paillier Encrypted Values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006)
17. Toft, T.: Primitives and Applications for Multi-party Computation. PhD thesis, University of Aarhus (2007), http://www.daimi.au.dk/~ttoft/publications/dissertation.pdf
18. Toft, T.: Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 357–371. Springer, Heidelberg (2009)

A An Overview of the New Protocols

The details are presented in Table 1. Below are some notes.

As mentioned in Section 3.1, R_{pub} represents the round complexity of the public exponentiation protocol (i.e. Pub-Expo$(\cdot)$) and C_{pub} represents the communication complexity (of Pub-Expo$(\cdot)$), and the values of R_{pub} and C_{pub} are determined by the adversary considered. Specifically, in the semi-honest case, $R_{pub} = R_{pub}^{s-h} = 5$ and $C_{pub} = C_{pub}^{s-h} = 10n + 3$; in the malicious case, $R_{pub} = R_{pub}^{mal} = 7$ and $C_{pub} = C_{pub}^{mal} = 3kn + 10n + 2k + 3$, in which n denotes the number of the participants of the MPC protocol and k is the security parameter for cut-and-choose. Both R_{pub} and C_{pub} can be viewed as *constants* because they are independent from (the input length) l. What's more, as mentioned in Section 5, $L(m) = \lceil \log m \rceil$ represents the bit-length of m.

Table 1. Overview of The New Protocols

Protocol Description	Rounds	Multiplications
$[x^a]_p \leftarrow$ **Pub-Expo**$([x]_p, a)$	R_{pub}	C_{pub}
$[x^a]_p \leftarrow$ **Bit-Expo**$([x]_p, [a]_B)$	$R_{pub} + 4$	$C_{pub} \cdot l + 6l$
$[x^a]_p \leftarrow$ **Bit-Expo**$^+([x]_p, [a]_B)$	$R_{pub} + 8$	$C_{pub} \cdot 2\sqrt{l} + 11l$
$[x^a]_p \leftarrow$ **Pri-Expo-BD**$([x]_p, [a]_p)$	$R_{pub} + 27$	$163l + C_{pub} \cdot l + 31l \log l$
$[x^a]_p \leftarrow$ **Pri-Expo**$([x]_p, [a]_p)$	$R_{pub} + 15$	$157l + C_{pub} \cdot l + C_{pub} + 5$
$[x^a]_p \leftarrow$ **Pri-Expo**$^+([x]_p, [a]_p)$	$R_{pub} + 19$	$162l + C_{pub} \cdot 2\sqrt{l} + C_{pub} + 5$
$[x \bmod m]_p \leftarrow$ **Pub-MR**$([x]_p, m)$	22	$326l + 28L(m)$
$[x \bmod m]_p \leftarrow$ **Pub-MR**$^+([x]_p, m)$	22	$78l + 276L(m)$