Multiparty Computation for Modulo Reduction without Bit-Decomposition and a Generalization to Bit-Decomposition[*]

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, which is proposed by Damgård *et al.*, is a powerful tool for multi-party computation (MPC). Given a sharing of secret a, it allows the parties to compute the sharings of the bits of a in constant rounds. With the help of bit-decomposition, constant-rounds protocols for various MPC problems can be constructed. However, bit-decomposition is relatively expensive, so constructing protocols for MPC problems without relying on bit-decomposition is a meaningful work. In multi-party computation, it remains an open problem whether the *modulo reduction problem* can be solved in constant rounds without bit-decomposition.

In this paper, we propose a protocol for (public) modulo reduction without relying on bit-decomposition. This protocol achieves constant round complexity and linear communication complexity. Moreover, we show a generalized bit-decomposition protocol which can, in constant rounds, convert the sharing of secret a into the sharings of the digits of a, along with the sharings of the bits of every digit. The digits can be base-m for any $m \geq 2$.

Keywords: Multiparty Computation, Constant-Rounds, Modulo Reduction, Generalization to Bit-Decomposition.

1 Introduction

Secure multi-party computation (MPC) allows the computation of a function f when the inputs to f are secret values held by distinct parties. After running the MPC protocol, the parties obtains only the predefined outputs but nothing else, and the privacy of their inputs are guaranteed. Although generic solutions for MPC already exist [2,10], the efficiency of these protocols tends to be low. So we focus on constructing efficient protocols for specific functions. More exactly, we are interested in integer arithmetic in the information theory setting [12].

[*] Supported by the National Natural Science Foundation of China under Grant No. 60873232.

[**] Corresponding author.

A proper choice of representation of the inputs can have great influence on the efficiency of the computation [7,18]. For example, when we want to compute the *sum* or the *product* of some private integer values, we'd better represent these integers as elements of a prime field $\mathbb{Z}_p$ and perform the computations using an arithmetic circuit as additions and multiplications are trivial operations in the field. If we use the binary representation of the integers and a Boolean circuit to compute the expected result, then we will get a highly inefficient protocol as the bitwise addition and multiplication are very expensive [4,5]. On the other hand, if we want to *compare* some (private) integer values, the binary representation will be of great advantage as comparison is a *bit-oriented* operation. In this case, the arithmetic circuit over $\mathbb{Z}_p$ will be a bad choice.

To bridge the gap between the arithmetic circuits and the Boolean circuits, Damgård *et al.* [7] proposed a novel protocol, called bit-decomposition, to convert a sharing of secret a into the sharings of the bits of a. This protocol is very useful both in theory and application. However, the bit-decomposition protocol is relatively expensive in terms of round and communication complexities. So the work on constructing (constant-rounds) protocols for MPC problems without relying on bit-decomposition is not only interesting but also meaningful. Recently, in [12], Nishide *et al.* constructed more efficient protocols for comparison, interval test and equality test of shared secrets without relying on the bit-decomposition protocol. However, it remains an open problem whether the *modulo reduction problem* can be solved in constant rounds without bit-decomposition [17]. In this paper, we show a linear protocol for the (public) modulo reduction problem without relying on bit-decomposition. What's more, the bit-decomposition protocol of [7] can only de-composite the sharing of secret a into the sharings of the *bits* of a. However, especially in practice, we may often need the sharings of the *digits* of a. Here the digits can be base-m for any $m \geq 2$. For example, in real life, integers are (almost always) represented as base-10 digits. Then, MPC protocols for practical use may often require the base-10 digits of the secret shared integers. Another example is as follows. If the integers are about *time* and *date*, then base-24, base-30, base-60, or base-365 digits may be required. So, to meet these requirements, we propose a generalization to bit-decomposition in this paper.

1.1 Our Contributions

First we introduce some necessary notations. We focus mainly on the multi-party computation based on linear secret sharing schemes. Assume that the underlying secret sharing scheme is built in field $\mathbb{Z}_p$ where p is a prime with bit-length l (i.e. $l = \lceil \log p \rceil$). For secret $a \in \mathbb{Z}_p$, we use $[a]_p$ to denote the secret sharing of a, and $[a]_B$ to denote the sharings of the bits of a, i.e. $[a]_B = \Big([a_{l-1}]_p, ..., [a_1]_p, [a_0]_p\Big)$.

The *public modulo reduction problem* can be formalized as follows:

$$[x \mod m]_p \leftarrow Modulo - Reduction([x]_p, m)$$

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

In existing public modulo reduction protocols [7,17], the bit-decomposition is involved, incurring $O(l \log l)$ communication complexity. What's more, in the worst case, the communication complexity of this protocol may go up to $\Theta(l^2)$. Specifically, the existing modulo reduction protocol uses the bit-decomposition protocol to reduce the "size" of the problem, and then uses up to l *comparisons*, which is non-trivial, to determine the final result. This is essentially an "exhaustive search". If the bit-length of the inputs to the comparison protocol is relatively long, e.g. $\Theta(l)$ which is often the case, the overall complexity will go up to $\Theta(l^2)$. So, the efficiency of the protocol may be very poor. To solve this problem, we propose a protocol, which achieves constant round complexity and linear communication complexity, for public modulo reduction without relying on bit-decomposition. Besides this, we also propose an enhanced protocol that can output the sharings of the bits of $x \mod m$, i.e. $[x \mod m]_B$.

Moreover, we also construct a generalized bit-decomposition protocol which can, in constant rounds, convert the sharing of secret a into the sharings of the digits of a, along with the sharings of the bits of every digit. The digits can be base-m for any $m \geq 2$. We name this protocol the *Base-m Digit-Bit-Decomposition Protocol*. The asymptotic communication complexity of this protocol is $O(l \log l)$. Obviously, when m is a power of 2, this protocol degenerates to the bit-decomposition protocol.

For illustration, we will show an example here. Pick a binary number

$$a = (11111001)_2 = 249.$$

If $[a]_p$ is given to the bit-decomposition protocol as input, it outputs

$$[a]_B = ([1]_p, [1]_p, [1]_p, [1]_p, [1]_p, [0]_p, [0]_p, [1]_p);$$

if $[a]_p$ and $m = 2$ (or $m = 4, 8, 16, 32, ...$) are given to our *Base-m Digit-Bit-Decomposition* protocol as inputs, it will output the same result with the bit-decomposition protocol above; however, when $[a]_p$ and $m = 10$ are given to our *Base-m Digit-Bit-Decomposition* protocol, it will output

$$([2]_B, [4]_B, [9]_B) = \Big(([0]_p, [0]_p, [1]_p, [0]_p), ([0]_p, [1]_p, [0]_p, [0]_p), ([1]_p, [0]_p, [0]_p, [1]_p)\Big)$$

which is significantly different from the output of bit-decomposition.

We also propose a simplified version of the protocol, called *Base-m Digit-Decomposition Protocol*, which outputs $\Big([2]_p, [4]_p, [9]_p\Big)$ when given $[a]_p$ and $m = 10$ as inputs.

Finally, 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

The problem of bit-decomposition is a basic problem in MPC and was partially solved by Algesheimer *et al.* in [1]. However, their solution is not constant-rounds and can only handle values that are noticeably smaller than p. Damgård *et al.* proposed the first constant-rounds (full) solution to the problem of bit-decomposition in [7]. This ice-break work is based on linear secret sharing schemes [2,11]. Independently, Shoenmakers and Tuyls [16] solved the problem of bit-decomposition for multiparty computation based on (Paillier) threshold homomorphic cryptosystems [3,8]. Somewhat surprisingly, Nishide and Ohta proposed solutions for comparison, interval test and equality test of shared secrets without relying on bit-decomposition [12]. Their techniques are novel, and have enlightened us a lot. Recently, Toft showed a novel technique that can reduce the communication complexity of bit-decomposition to *almost linear* [18]. Although we do not focus on the "almost linear" property of protocols, some techniques proposed in their paper are so inspiring and enlightening to us. In a followup work, Reistad and Toft proposed a linear bit-decomposition protocol [14]. However, the security of their protocol is non-perfect.

As for the problem of modulo reduction (without bit-decomposition), Guajardo *et al.* proposed a partial solution to this problem in the threshold homomorphic setting [9]. In [6], Catrina *et al.* dealt with the non-constant-rounds private modulo reduction protocol with the incomplete accuracy and statistical privacy in the setting where shared secrets are represented as fixed-point numbers.

2 Preliminaries

In this section we introduce some important notations and some known primitives which will be frequently mentioned in the rest of the paper.

2.1 Notations and Conventions

The multiparty computation considered in this paper is based on linear secret sharing schemes, such as Shamir's [15]. As mentioned above, we denote the underlying field as $\mathbb{Z}_p$ where p is a prime with binary length l.

As in previous works, such as [7] and [12], we assume that the underlying secret sharing scheme allows to compute $[a + b \bmod p]_p$ from $[a]_p$ and $[b]_p$ without communication, and that it allows to compute $[ab \bmod p]_p$ from (public) $a \in \mathbb{Z}_p$ and $[b]_p$ without communication. We also assume that the secret sharing scheme allows to compute $[ab \bmod p]_p$ from $[a]_p$ and $[b]_p$ through communication among the parties. We call this procedure the *multiplication protocol.* Obviously, for multiparty computation, the multiplication protocol is a dominant factor of complexity as it involves communication. So, as in previous works, the round complexity of the protocols is measured by the number of rounds of parallel invocations of the multiplication protocol, and the communication complexity is measured by the number of invocations of the multiplication protocol. For example, if a protocol involves a multiplications in parallel and then another b multiplications in parallel, then we can say that the round complexity is 2 and the communication complexity is $a + b$ multiplications. We have to say that the complexity analysis made in this paper is somewhat rough for we focus mainly on the ideas of the solution, but not on the details of the implementation.

As in [12], 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 $[x \overset{?}{<} y]_p$ to denote the output of the comparison protocol, i.e. $(x \overset{?}{<} y) = 1$ iff $x < y$ holds.

For the base $m \in \{2, 3, ..., p-1\}$, define $L(m) = \lceil \log m \rceil$. It is easy to see that we should use $L(m)$ bits to represent a base-m digit. For example, when $m = 10$, we have $L(m) = \lceil \log 10 \rceil = 4$, this means we must use 4 bits to represent a base-10 digit. Notice that we have $2^{L(m)-1} < m \leq 2^{L(m)}$ and $m = 2^{L(m)}$ holds iff m is a power of 2. Moreover, we have $L(m) \leq l$ as $m \leq p-1$.

Define $l^{(m)} = \lceil \log_m p \rceil$. Obviously, $l^{(m)}$ is the length of p when p is coded base-m. Note that $l^{(m)} = \lceil \log_m p \rceil = \left\lceil \frac{\log p}{\log m} \right\rceil = \left\lceil \frac{l}{\log m} \right\rceil \leq l$ as $m \geq 2$.

For any $a \in \mathbb{Z}_p$, the secret sharing of a is denoted by $[a]_p$. We use $[a]_B$ to denote the bitwise sharing of a.

We use $[a]_D^m = \left([a_{l^{(m)}-1}]_p^m, ..., [a_1]_p^m, [a_0]_p^m\right)$ to denote the *digit-wise sharing* of a. For $i \in \{0, 1, ..., l^{(m)} - 1\}$, $[a_i]_p^m$ denotes the sharing of the *i'th* base-m digit of a with $0 \leq a_i \leq (m-1)$.

The *digit-bit-wise sharing* of a, which is denoted by $[a]_{D,B}^m$, is defined as below:

$$[a]_{D,B}^m = \left([a_{l^{(m)}-1}]_B^m, ..., [a_1]_B^m, [a_0]_B^m\right),$$

in which $[a_i]_B^m = \left([a_i^{L(m)-1}]_p, ..., [a_i^1]_p, [a_i^0]_p\right)$ $(i \in \{0, 1, ..., l^{(m)} - 1\})$ denotes the bitwise sharing of the *i'th* base-m digit of a. Note that $[a_i]_B^m$ has $L(m)$ bits.

Sometimes, if m can be inferred from the context, we may write $[a_i]_p^m$ (or $[a_i]_B^m$) as $[a_i]_p$ (or $[a_i]_B$) for simplicity.

In this paper, we often need to get the *digit-wise representation* or the *digit-bit-wise representation* of some public value c, i.e. $[c]_D^m$ or $[c]_{D,B}^m$. This can be done freely as c is public.

It's easy to see that if we have obtained $[x]_B$, then $[x]_p$ can be freely obtained by a linear combination. We can think of this as $[x]_B$ contains "more information" than $[x]_p$. For example, if we get $[a]_{D,B}^m = \left([a_{l^{(m)}-1}]_B^m, ..., [a_1]_B^m, [a_0]_B^m\right)$, then $[a]_D^m = \left([a_{l^{(m)}-1}]_p^m, ..., [a_1]_p^m, [a_0]_p^m\right)$ is implicitly obtained. In protocols that can output both $[x]_B$ and $[x]_p$, which is often the case in this paper, we always omit $[x]_p$ for simplicity.

Given $[c]_p$, we need a protocol to reveal c, which is denoted by $c \leftarrow reveal([c]_p)$.

When we write command $C \leftarrow b?A : B$, where $A, B, C \in \mathbb{Z}_p$ and $b \in \{0, 1\}$, it means the following:

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

We call this command the *conditional selection command.* When all the variables in this command are public, this "selection" can of course be done. When the variables are shared or even bitwise shared, this can also be done. Specifically, the command

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

can be realized by setting

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

which costs 1 round and 1 multiplication; the command

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

can be realized by the following procedure:

```
For $i = 0, 1, ..., l-1$ in parallel: $[C_i]_p \leftarrow [b]_p([A_i]_p - [B_i]_p) + [B_i]_p$
$[C]_B \leftarrow \left([C_{l-1}]_p, ..., [C_1]_p, [C_0]_p\right)$        $\triangleright |A| = |B| = |C| = l$
```

Note that the above procedure costs 1 round, l invocations of multiplication.

Other cases, such as $[C]_D^m \leftarrow [b]_p?[A]_D^m : [B]_D^m$ and $[C]_{D,B}^m \leftarrow [b]_p?[A]_{D,B}^m : [B]_{D,B}^m$ can be realized similarly. We will frequently use this *conditional selection command* in our protocols.

2.2 Known Primitives

We will now simply introduce some existing primitives which are important building blocks of this paper. All these primitives are proposed in [7].

▲***Random-Bit.*** The *Random-Bit* protocol is the most basic primitive which can generate a shared uniformly random bit unknown to all parties. In the linear secret sharing setting, which is the case in this paper, it takes only 2 rounds and 2 multiplications.

▲***Bitwise-LessThan.*** Given two bitwise shared inputs $[x]_B$ and $[y]_B$, the *Bitwise-LessThan* protocol can compute a shared bit $[x \overset{?}{<} y]_p$. We note that using the method of [18], this protocol can be realized in 6 rounds and $13l + 6\sqrt{l}$ multiplications. Notice that $13l + 6\sqrt{l} \leq 14l$ holds for $l \geq 36$ which is often the case in practice. So, for simplicity, we refer to the complexity of this protocol as 6 rounds and $14l$ multiplications.

▲***Bitwise-Addition.*** Given two bitwise shared inputs, $[x]_B$ and $[y]_B$, the *Bitwise-Addition* protocol outputs $[x + y]_B$. An important point of this protocol is that $d = x + y$ holds over the integers, not (only) mod p. This protocol, which costs 15 rounds and $47l \log l$ multiplications, is the most expensive primitive of the bit-decomposition protocol of [7]. We will not use this primitive in this paper, but use *Bitwise-Subtraction* instead. However, the asymptotic complexity of our *Bitwise-Subtraction* protocol is the same with that of the *Bitwise-Addition* since they both involve a *generic prefix protocol* which costs $O(l \log l)$ multiplications. We will introduce our *Bitwise-Subtraction* protocol later.

3 A Simple Introduction to Our New Primitives

In this section, we will simply introduce the new primitives proposed in this paper. We will only describe the inputs and the outputs of the protocols, along with some simple comments. All these new primitives will be described in detail in Section 6.

•***Bitwise-Subtraction.*** The *Bitwise-Subtraction* protocol accepts two bitwise shared values $[x]_B$ and $[y]_B$ and outputs $[x - y]_B$. This protocol is in fact first proposed in [18] and is re-described (in a widely different form) in this paper. In our protocols, we only need a restricted version (of *Bitwise-Subtraction*) which requires $x \geq y$. A run of this *restricted* protocol is denoted by

$$[x-y]_B \leftarrow Bitwise-Subtraction^*([x]_B,[y]_B).$$

It costs 15 rounds and $47l\log l$ multiplications.

•***BORROWS.*** This protocol is used as a sub-protocol in the *Bitwise-Subtraction* protocol above to compute the borrow bits (as well as in the *Bitwise-Subtraction** protocol). Given two bitwise sharings $[x]_B$ and $[y]_B$, this protocol outputs

$$([b_{l-1}]_p,...,[b_1]_p,[b_0]_p) \leftarrow BORROWS([x]_B,[y]_B)$$

where $[b_i]_p$ is the sharing of the borrow bit at bit-position $i \in \{0,1,...,l-1\}$.

•***Random-Digit-Bit.*** Given $m \in \{2,3,...,p-1\}$ as input, the *Random-Digit-Bit* protocol outputs

$$[d]_B^m = \left([d^{L(m)-1}]_p,...,[d^1]_p,[d^0]_p\right) \leftarrow Random-Digit-Bit(m)$$

where $d \in \{0,1,...,m-1\}$ represents a base-m digit. Notice that $[d]_p^m$ is implicitly obtained. The complexity of this protocol is 8 rounds and $16L(m)$ multiplications.

•***Digit-Bit-wise-LessThan.*** The *Digit-Bit-wise-LessThan* protocol accepts two digit-bit-wise shared values $[x]_{D,B}^m$ and $[y]_{D,B}^m$ and outputs

$$[x \overset{?}{<} y]_p \leftarrow Digit-Bit-wise-LessThan([x]_{D,B}^m,[y]_{D,B}^m).$$

The complexity of this protocol is 6 rounds and $14l$ multiplications.

•***Random-Solved-Digits-Bits.***Using the above two primitives as sub-protocols, we can construct the *Random-Solved-Digits-Bits* protocol which, when given $m \in \{2,3,...,p-1\}$ as input, outputs a digit-bit-wise shared random value $[r]_{D,B}^m$ satisfying $r<p$. We denote a run of this protocol by

$$[r]_{D,B}^m \leftarrow Random-Solved-Digits-Bits(m).$$

This protocol takes 14 rounds and $312l$ multiplications.

•***Digit-Bit-wise-Subtraction.*** This protocol is a novel generalization to the bitwise subtraction protocol and is very important to this paper. It accepts two digit-bit-wise shared values $[x]_{D,B}^m$ and $[y]_{D,B}^m$ and outputs $[x-y]_{D,B}^m$. Again, in this paper, we need only a restricted version which requires $x \geq y$. A run of this restricted protocol is denoted by

$$[x-y]_{D,B}^m \leftarrow Digit-Bit-wise-Subtraction^*([x]_{D,B}^m,[y]_{D,B}^m).$$

This restricted protocol costs 30 rounds and $47l\log l + 47l\log(L(m))$ multiplications. What's more, if we don't need $[x-y]_{D,B}^m$ but (only) need $[x-y]_D^m$ instead, then this restricted protocol can be (further) simplified. We denote a run of this (further) simplified protocol by

$$[x-y]_D^m \leftarrow Digit-Bit-wise-Subtraction^{*-}([x]_{D,B}^m,[y]_{D,B}^m).$$

The complexity of this protocol goes down to 21 rounds and $16l+47l^{(m)}\log\left(l^{(m)}\right)$ multiplications.

With the above primitives, we can construct our *Modulo-Reduction* protocol and *Base-m Digit-Bit-Decomposition* protocol, which will be described in detail separately in Section 4 and Section 5.

4 Multiparty Computation for Modulo Reduction without Bit-Decomposition

In this section, we give out our (public) *Modulo-Reduction* protocol which is realized without relying on bit-decomposition. This protocol is constant-rounds and involves only $O(l)$ multiplications. Informally speaking, our *Modulo-Reduction* protocol is essentially the *Least Significant Digit Protocol* and is a natural generalization to the *Least Significant Bit Protocol* (i.e. the *LSB* protocol) in [12]. Recall that for an integer a, the sharing of the least significant base-m digit of a is denoted by $[a_0]_p^m$, and the bitwise sharing of the least significant base-m digit of a is denoted by $[a_0]_B^m$. The protocol is described in detail in **Protocol 1**.

Protocol 1. The modulo reduction protocol, $Modulo - Reduction(\cdot)$, for computing the residue of a shared integer modulo a public integer.

```
**Input**: $[x]_p$ with $x \in \mathbb{Z}_p$ and $m \in \{2, 3, ..., p-1\}$.
**Output**: $[x \mod m]_p$.
**Process**:
$[r]_{D,B}^m \leftarrow Random - Solved - Digits - Bits(m)$
$c \leftarrow reveal([x]_p + [r]_{D,B}^m)$    $\triangleright$Note that $[r]_{D,B}^m$ implies $[r]_p^m$.    (1.a)
$[X_1]_p^m \leftarrow [c_0]_p^m - [r_0]_B^m$    $\triangleright$ $[r_0]_B^m$ implies $[r_0]_p^m$.
$[X_2]_p^m \leftarrow [c_0]_p^m - [r_0]_B^m + m$
$[s]_p \leftarrow Bitwise - LessThan([c_0]_B^m, [r_0]_B^m)$    (1.b)
$[X]_p^m \leftarrow [s]_p ? [X_2]_p^m : [X_1]_p^m$    $\triangleright$A *conditional selection command.*

$c' \leftarrow c + p$    $\triangleright$Addition over the integers.
$[X_1']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m$
$[X_2']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m + m$
$[s']_p \leftarrow Bitwise - LessThan([c_0']_B^m, [r_0]_B^m)$    (1.c)
$[X']_p^m \leftarrow [s']_p ? [X_2']_p^m : [X_1']_p^m$

$[t]_p \leftarrow Digit - Bit - wise - LessThan([c]_{D,B}^m, [r]_{D,B}^m)$    (1.d)
$[x \mod m]_p = [x_0]_p^m \leftarrow [t]_p ? [X']_p^m : [X]_p^m$
**Return** $[x \mod m]_p$
```

Correctness: By simulating a *base-m addition* process, the protocol extracts $[x_0]_p^m$ which is just $[x \mod m]_p$. See [13] for the details.

Privacy: The only possible information leakage takes place in line (1.a), where a *reveal* command is involved. However, the revealed value, i.e. c, is uniformly random, so it leaks no information about the secret x. So the privacy is guaranteed.

Complexity: Complexity comes mainly from the invocations of sub-protocols. Note that the two invocations of *Bitwise-LessThan* and the invocation of *Digit-Bit-wise-LessThan* can be scheduled in parallel. In all it will cost 22 rounds and

$$312l + 14L(m) + 1 + 14L(m) + 1 + 14l + 1 = 326l + 28L(m) + 3$$

multiplications. Recall that $L(m) \leq l$, so the communication complexity is upper bounded by $354l + 3$ multiplications.

The original modulo reduction problem does not require the sharings of the bits of the residue, i.e. $[x \bmod m]_B$. So in the above protocol, $[x \bmod m]_B$ is not computed. However, if we want, we can get $[x \bmod m]_B$ using an enhanced version of the above *Modulo-Reduction* protocol. This enhanced protocol will be denoted by $Modulo-Reduction^{+}(\cdot)$. The construction is seen as **Protocol 2**.

Protocol 2. The *enhanced* modulo reduction protocol, $Modulo-Reduction^{+}(\cdot)$, for computing the *bitwise shared* residue of a shared integer modulo a public integer.

```
**Input**: $[x]_p$ with $x \in \mathbb{Z}_p$ and $m \in \{2, 3, ..., p-1\}$.
**Output**: $[x \mod m]_B$.
**Process**:
$[r]^m_{D,B} \leftarrow Random-Solved-Digits-Bits(m)$
$c \leftarrow reveal([x]_p + [r]^m_{D,B})$
$[\bar{M}_1]^m_B \leftarrow [c_0]^m_B$          $[\bar{S}_1]^m_B \leftarrow [r_0]^m_B$
$[\bar{M}_2]^m_B \leftarrow [c_0 + m]^m_B$      $[\bar{S}_2]^m_B \leftarrow [r_0]^m_B$   $\triangleright$Addition over the integers.
$[s]_p \leftarrow Bitwise-LessThan([c_0]^m_B, [r_0]^m_B)$
$[\bar{M}]^m_B \leftarrow [s]_p?[\bar{M}_2]^m_B : [\bar{M}_1]^m_B$     $\triangleright$Involving $L(m)$ multiplications.
$[\bar{S}]^m_B \leftarrow [s]_p?[\bar{S}_2]^m_B : [\bar{S}_1]^m_B$

$c' \leftarrow c + p$
$[\bar{M}'_1]^m_B \leftarrow [c'_0]^m_B$          $[\bar{S}'_1]^m_B \leftarrow [r_0]^m_B$
$[\bar{M}'_2]^m_B \leftarrow [c'_0 + m]^m_B$      $[\bar{S}'_2]^m_B \leftarrow [r_0]^m_B$
$[s']_p \leftarrow Bitwise-LessThan([c'_0]^m_B, [r_0]^m_B)$
$[\bar{M}']^m_B \leftarrow [s']_p?[\bar{M}'_2]^m_B : [\bar{M}'_1]^m_B$
$[\bar{S}']^m_B \leftarrow [s']_p?[\bar{S}'_2]^m_B : [\bar{S}'_1]^m_B$

$[t]_p \leftarrow Digit-Bit-wise-LessThan(c, [r]^m_{D,B})$
$[M]^m_B \leftarrow [t]_p?[\bar{M}']^m_B : [\bar{M}]^m_B$     $\triangleright$M is the minuend.
$[S]^m_B \leftarrow [t]_p?[\bar{S}']^m_B : [\bar{S}]^m_B$       $\triangleright$S is the subtrahend.

$[x \mod m]_B = [x_0]^m_B \leftarrow Bitwise-Subtraction^*([M]^m_B, [S]^m_B)$
**Return** $[x \mod m]_B$
```

The correctness and privacy of this protocol can be proved similarly to the *Modulo-Reduction* protocol above. By carefully selecting the *Minuend* and the *Subtrahend*, we can get the expected result by using *only one invocation* of the *Bitwise-Subtraction** protocol. The overall complexity of this protocol is 37 rounds and

$$326l+28L(m)+47L(m)\log(L(m))+6\ L(m) = 326l+34L(m)+47L(m)\log(L(m))$$

multiplications.

5 A Generalization to Bit-Decomposition

In this section, we will propose our generalization to bit-decomposition, i.e. the *Base-m Digit-Bit-Decomposition* protocol. The details of this protocol are presented in **Protocol 3**. The main framework of this protocol is similar to the bit-decomposition protocol of [18].

Protocol 3. The *Base-m Digit-Bit-Decomposition* protocol, $Digit - Bit - Decomposition(\cdot, m)$, for converting the sharing of secret x into the digit-bit-wise sharing of x.

```
Input: $[x]_p$ with $x \in \mathbb{Z}_p$ and the base $m \in \{2, 3, ..., p-1\}$.
Output: $[x]^m_{D,B}$
Process:
$[r]^m_{D,B} \leftarrow Random - Solved - Digits - Bits(m)$
$c \leftarrow reveal([x]_p + [r]^m_{D,B})$                                      (3.a)
$c' \leftarrow c + p$
$[t]_p \leftarrow Digit - Bit - wise - LessThan([c]^m_{D,B}, [r]^m_{D,B})$               (3.b)
$[\tilde{c}]^m_{D,B} = [t]_p ? [c']^m_{D,B} : [c]^m_{D,B}$    $\triangleright$Note that $\tilde{c} = x + r$
$[x]^m_{D,B} \leftarrow Digit - Bit - wise - Subtraction^*([\tilde{c}]^m_{D,B}, [r]^m_{D,B})$          (3.c)
Return $[x]^m_{D,B}$
```

Correctness is described in detail in [13]. Privacy is straightforward. The overall complexity of this protocol is $14 + 6 + 30 = 50$ rounds and

$$312l + 14l + (47l \log l + 47l \log(L(m))) = 326l + 47l \log l + 47l \log(L(m))$$

multiplications. The communication complexity is upper bounded by $326l + 94l \log l$ multiplications as $L(m) \leq l$.

If we do not need $[x]^m_{D,B}$ but (only) need $[x]^m_D$ instead, then the above protocol can be simplified. The method is to replace the *Digit-Bit-wise-Subtraction** protocol with the *Digit-Bit-wise-Subtraction*$^{*-}$ protocol. We call this simplified protocol the *Base-m Digit-Decomposition Protocol*, a run of which is denoted by $Digit - Decomposition(\cdot, m)$. The correctness and privacy of this protocol can be similarly proved. The complexity goes down to $14 + 6 + 21 = 41$ rounds and

$$312l + 14l + \left(16l + 47l^{(m)} \log\left(l^{(m)}\right)\right) = 342l + 47l^{(m)} \log\left(l^{(m)}\right)$$

multiplications. Recall that $l^{(m)} = \lceil \log_m p \rceil \leq l$, so the communication complexity is upper bounded by $342l + 47l \log l$ multiplications.

6 Realizing the Primitives

In this section, we will describe in detail the (new) primitives which are essential for the protocols of our paper. Informally, most of the protocols in this section are generalized version of the protocols of [7] from base-2 to base-m for any $m \geq 2$. It will be seen that, when m is a power of 2, some of our primitives degenerate to the existing primitives in [7]. So, in the complexity analysis, we focus on the case where m is not a power of 2, i.e. $m < 2^{L(m)}$.

6.1 Bitwise-Subtraction

We describe the *Bitwise-Subtraction* protocol here. In fact, this protocol is already proposed in [18]. They reduced the problem of bitwise-subtraction to the *Post-fix Comparison* problem. Here, we re-consider the problem of bitwise-subtraction and solve it in a (highly) similar manner to the *Bitwise-Addition* protocol of [7].

As is mentioned in Section 3, we will first propose a restricted (*bitwise-subtraction*) protocol, *Bitwise-Subtraction**, which requires that *the minuend is not less than the subtrahend.* We will only use this restricted version in this paper. The general version without the above restriction can be realized with the help of the *Bitwise-LessThan* protocol. See [13] for the details. Given a *BORROWS* protocol that can compute the sharings of the borrow bits, the *Bitwise-Subtraction** protocol can be realized as in **Protocol 4**.

Protocol 4. The *restricted* bitwise-subtraction protocol, $Bitwise - Subtraction^*(\cdot)$, for computing the bitwise sharing of the difference between two bitwise shared values. This protocol requires that the minuend is not less than the subtrahend.

```
Input: $[x]_B = ([x_{l-1}]_p, ..., [x_1]_p, [x_0]_p)$ and $[y]_B = ([y_{l-1}]_p, ..., [y_1]_p, [y_0]_p)$ satisfying $x \geq y$.
Output: $[x-y]_B = [d]_B = ([d_{l-1}]_p, ..., [d_1]_p, [d_0]_p)$.
Process:
$([b_{l-1}]_p, ..., [b_1]_p, [b_0]_p) \leftarrow BORROWS([x]_B, [y]_B)$
$[d_0]_p \leftarrow [x_0]_p - [y_0]_p + 2[b_0]_p$
For $i = 1, 2, ..., l-1$ in parallel: $[d_i]_p \leftarrow [x_i]_p - [y_i]_p + 2[b_i]_p - [b_{i-1}]_p$
$[x-y]_B = [d]_B \leftarrow ([d_{l-1}]_p, ..., [d_1]_p, [d_0]_p)$
Return $[x-y]_B$
```

Note that the output of this protocol, i.e. $[x-y]_B$, is of bit length l, not $l+1$. This is because $x \geq y$ holds and thus we do not need a sign bit. Correctness and privacy is straightforward. The complexity of this protocol is 15 rounds and $47l \log l$ multiplications.

6.2 Computing the Borrow Bits

We now describe the *BORROWS* protocol which can compute the sharings of the borrow bits. In fact our *BORROWS* protocol is highly similar to the *CARRIES* protocol in [7]. So only the difference is sketched here. As in [7], we use an operator $\circ : \sum \times \sum \rightarrow \sum$, where $\sum = \{S, P, K\}$, which is defined by $S \circ x = S$ for all $x \in \sum$, $K \circ x = K$ for all $x \in \sum$, $P \circ x = x$ for all $x \in \sum$. Here, $\circ$ represents the *borrow-propagation* operator, whereas in [7] it represents the *carry-propagation* operator. When computing $[x-y]_B$ (where $x \geq y$ holds) with two bitwise shared inputs

$$[x]_B = ([x_{l-1}]_p, ..., [x_1]_p, [x_0]_p) \;\; and \;\; [y]_B = ([y_{l-1}]_p, ..., [y_1]_p, [y_0]_p),$$

for bit-position $i \in \{0, 1, \ldots, l-1\}$, let $e_i = S$ iff a borrow is set at position i (i.e. $x_i < y_i$); $e_i = P$ iff a borrow would be propagated at position i (i.e. $x_i = y_i$); $e_i = K$ iff a borrow would be killed at position i (i.e. $x_i > y_i$). It can be easily verified that $b_i = 1$ (i.e. the $i'th$ borrow bit is set, which means the $i'th$ bit needs to borrow a "1" from the $(i+1)'th$ bit) iff $e_i \circ e_{i-1} \circ \cdots \circ e_0 = S$. It can be seen that in the case where $\circ$ represents the borrow-propagation operator and in the case where $\circ$ represents the carry-propagation operator, the rules for $\circ$ (i.e. $S \circ x = S$, $K \circ x = K$ and $P \circ x = x$ for all $x \in \sum$) are *completely the same.* This means that when computing the borrow bits, once all the $e_i's$ are obtained, the residue procedure of our *BORROWS* protocol will be (completely) the same with that of the CARRIES protocol (in [7]). So, the only difference lies in the procedure of computing the $e_i's$, which will be sketched below.

As in [7], we represent S, P and K with bit vectors

$$(1,0,0),(0,1,0),(0,0,1) \in \{0,1\}^3.$$

Then, for every bit-position $i \in \{0,1,...,l-1\}$, $[e_i]_B = ([s_i]_p, [p_i]_p, [k_i]_p)$ can be obtained as follows: $[s_i]_p = [y_i]_p - [x_i]_p[y_i]_p$; $[p_i]_p = 1 - [x_i]_p - [y_i]_p + 2[x_i]_p[y_i]_p$; $[k_i]_p = [x_i]_p - [x_i]_p[y_i]_p$, which in fact need only one multiplication (i.e. $[x_i]_p[y_i]_p$). Correctness follows readily from the above arguments. Privacy is straightforward. The complexity of the protocol is 15 rounds and $47l \log l$ multiplications.

6.3 Random-Digit-Bit

We will now introduce the *Random-Digit-Bit* protocol for generating a random bitwise shared base-m digit, which is denoted by d here. In fact, d is a random integer satisfying $0 \le d \le m-1$. The details are presented in **Protocol 5**.

Protocol 5. The *Random-Digit-Bit* protocol, $Random-Digit-Bit(\cdot)$, for generating the bitwise sharing of a random digit. The digit is base-m for any $m \ge 2$.

Input: The base m satisfying $2 \le m \le p-1$.
Output: $[d]_B^m = ([d^{L(m)-1}]_p, ..., [d^1]_p, [d^0]_p)$ with $0 \le d \le m-1$.
Process:

```
For $i = 0, 1, \ldots, L(m) - 1$ in parallel: $[d^i]_p \leftarrow Random - Bit()$.
$[d]_B^m \leftarrow ([d^{L(m)-1}]_p, ..., [d^1]_p, [d^0]_p)$
If $m = 2^{L(m)}$, then **Return** $[d]_B^m$. Otherwise proceed as below.
$[r]_p \leftarrow Bitwise - LessThan([d]_B^m, m)$
$r \leftarrow reveal([r]_p)$
If $r = 0$, then abort. Otherwise **Return** $[d]_B^m$.
```

See [13] for the correctness. As for the privacy, when this protocol does not abort, the only information leaked is that $d < m$, which is an *a priori* fact. As for the complexity, when m is not a power of 2, the total complexity of one run of this protocol is 8 rounds and $16L(m)$ multiplications. As in [7], using a Chernoff bound, it can be seen that if this protocol has to be repeated in parallel to get a lower abort probability, then the round complexity is still 8, and the amortized communication complexity goes up to $4 \times 16L(m) = 64L(m)$ multiplications.

6.4 Digit-Bit-Wise-LessThan

The *Digit-Bit-wise-LessThan* protocol proposed here is a natural generalization to the *Bitwise-LessThan* protocol. Recall that 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. The details of the protocol are presented in **Protocol 6**.

Protocol 6. The *Digit-Bit-wise-LessThan* protocol, $Digit{-}Bit{-}wise{-}LessThan(\cdot)$, for comparing two digit-bit-wise shared values.

Input: Two digit-bit-wise shared values $[x]^m_{D,B} = ([x_{l^{(m)}-1}]^m_B, ..., [x_1]^m_B, [x_0]^m_B)$ and $[y]^m_{D,B} = ([y_{l^{(m)}-1}]^m_B, ..., [y_1]^m_B, [y_0]^m_B)$.

Output: $[(x \overset{?}{<} y)]_p$, where $(x \overset{?}{<} y) = 1$ iff $x < y$ holds.

Process:

$$
\begin{aligned}
[X]_B \leftarrow (&[x^{L(m)-1}_{l^{(m)}-1}]_p, ..., [x^1_{l^{(m)}-1}]_p, [x^0_{l^{(m)}-1}]_p,\\
&...\\
&...\\
&...\\
&[x^{L(m)-1}_1]_p, ..., [x^1_1]_p, [x^0_1]_p,\\
&[x^{L(m)-1}_0]_p, ..., [x^1_0]_p, [x^0_0]_p)
\end{aligned}
$$

$$
\begin{aligned}
[Y]_B \leftarrow (&[y^{L(m)-1}_{l^{(m)}-1}]_p, ..., [y^1_{l^{(m)}-1}]_p, [y^0_{l^{(m)}-1}]_p,\\
&...\\
&...\\
&...\\
&[y^{L(m)-1}_1]_p, ..., [y^1_1]_p, [y^0_1]_p,\\
&[y^{L(m)-1}_0]_p, ..., [y^1_0]_p, [y^0_0]_p)
\end{aligned}
$$

$[(x \overset{?}{<} y)]_p = [(X \overset{?}{<} Y)]_p \leftarrow Bitwise - LessThan([X]_B, [Y]_B)$

Return $[(x \overset{?}{<} y)]_p$

Correctness is presented in [13]. Privacy follows readily from only using private sub-protocols. The complexity of the protocol is 6 rounds and (about) $14l$ multiplications.

6.5 Random-Solved-Digits-Bits

The *Random-Solved-Digits-Bits* protocol is an important primitive which can generate a digit-bit-wise shared random value unknown to all parties. It is a natural generalization to the *Random-Solved-Bits* protocol in [7]. The details are presented in **Protocol 7**.

Recall that the bitwise representation of the most significant base-m digit of p is $[p_{l^{(m)}-1}]^m_B = \left(p^{L(m)-1}_{l^{(m)}-1}, ..., p^1_{l^{(m)}-1}, p^0_{l^{(m)}-1}\right)$. Suppose $p^j_{l^{(m)}-1}$ $(j \in \{0, 1, ...,$

$L(m)-1\}$) is the left-most "1" in $[p_{l^{(m)}-1}]_B^m$. Then, in order to get an acceptable abort probability, the bit-length of the most significant base-m digit of r should be $j+1$ because an acceptable r must be less than p. In this protocol, for simplicity, we assume that $p_{l^{(m)}-1}^{L(m)-1} = 1$. Under this assumption, we can generate $[r_{l^{(m)}-1}]_B^m$ using the *Random-Digit-Bit* protocol. If $p_{l^{(m)}-1}^{L(m)-1} = 0$, then $[r_{l^{(m)}-1}]_B^m$ can be generated by using the *Random-Bit* protocol directly.

Protocol 7. The *Random-Solved-Digits-Bits* protocol, $Random-Solved-Digits-Bits(\cdot)$, for jointly generating a digit-bit-wise shared value which is uniformly random from $\mathbb{Z}_p$.

Input: m, i.e. the expected base of the digits.
Output: $[r]_{D,B}^m$, in which r is a uniformly random value satisfying $r < p$.
Process:

```
For $i = 0, 1, ..., l^{(m)} - 1$ in parallel: $[r_i]_B^m \leftarrow Random - Digit - Bit(m)$.
$[r]_{D,B}^m \leftarrow ([r_{l^{(m)}-1}]_B^m, ..., [r_1]_B^m, [r_0]_B^m)$
$[c]_p \leftarrow Digit - Bit - wise - LessThan([r]_{D,B}^m, [p]_{D,B}^m)$
$c \leftarrow reveal([c]_p)$
If $c = 0$, then abort. Otherwise **Return** $[r]_{D,B}^m$.
```

The correctness and the privacy is straightforward. The amortized complexity of this protocol is 8+6=14 rounds and $\left(l^{(m)} \cdot 64L(m) + 14l\right) * 4 = 312l$ multiplications.

6.6 Digit-Bit-Wise-Subtraction

In this section, we will describe in detail the restricted version, *Digit-Bit-wise-Subtraction**, which requires that the minuend is not less than the subtrahend. The general version, which can be realized using the techniques in the *Bitwise-Subtraction* protocol and which is not used in the paper, is omitted for simplicity.

•The Restricted Digit-Bit-Wise-Subtraction. We will now describe in detail the *Digit-Bit-wise-Subtraction** protocol. This protocol is novel and is the most important primitive in our *Base-m Digit-Bit-Decomposition* protocol. The details are presented in **Protocol 8**.

Protocol 8. The *restricted Digit-Bit-wise-Subtraction* protocol, $Digit - Bit - wise - Subtraction^*(\cdot)$, for computing the digit-bit-wise sharing of the difference between two digit-bit-wise shared values with the minuend not less than the subtrahend.

Input: $[x]_{D,B}^m = ([x_{l^{(m)}-1}]_B^m, ..., [x_1]_B^m, [x_0]_B^m)$ and $[y]_{D,B}^m = ([y_{l^{(m)}-1}]_B^m, ..., [y_1]_B^m, [y_0]_B^m)$ satisfying $x \geq y$.
Output: $[x-y]_{D,B}^m = [d]_{D,B}^m = ([d_{l^{(m)}-1}]_B^m, ..., [d_1]_B^m, [d_0]_B^m)$.

```
**Process**:
$[X]_B \leftarrow ([x^{L(m)-1}_{l^{(m)}-1}]_p, ..., [x^1_{l^{(m)}-1}]_p, [x^0_{l^{(m)}-1}]_p,$
        ...
        ...
        ...
        $[x^{L(m)-1}_1]_p, ..., [x^1_1]_p, [x^0_1]_p,$
        $[x^{L(m)-1}_0]_p, ..., [x^1_0]_p, [x^0_0]_p)$
$[Y]_B \leftarrow ([y^{L(m)-1}_{l^{(m)}-1}]_p, ..., [y^1_{l^{(m)}-1}]_p, [y^0_{l^{(m)}-1}]_p,$
        ...
        ...
        ...
        $[y^{L(m)-1}_1]_p, ..., [y^1_1]_p, [y^0_1]_p,$
        $[y^{L(m)-1}_0]_p, ..., [y^1_0]_p, [y^0_0]_p)$
$([b^{L(m)-1}_{l^{(m)}-1}]_p, ..., [b^1_{l^{(m)}-1}]_p, [b^0_{l^{(m)}-1}]_p,$
...
...                                                        (8.a)
...
$[b^{L(m)-1}_1]_p, ..., [b^1_1]_p, [b^0_1]_p,$
$[b^{L(m)-1}_0]_p, ..., [b^1_0]_p, [b^0_0]_p) \leftarrow BORROWS([X]_B, [Y]_B)$
$[t^0_0]_p = [x^0_0]_p - [y^0_0]_p + 2[b^0_0]_p$                                 (8.b)
For $j = 1, ..., L(m) - 1$, in parallel: $[t^j_0]_p = [x^j_0]_p - [y^j_0]_p + 2[b^j_0]_p - [b^{j-1}_0]_p$.
For $i = 1, ..., l^{(m)} - 1$ do
    $[t^0_i]_p = [x^0_i]_p - [y^0_i]_p + 2[b^0_i]_p - [b^{L(m)-1}_{i-1}]_p$
    For $j = 1, ..., L(m) - 1$, in parallel: $[t^j_i]_p = [x^j_i]_p - [y^j_i]_p + 2[b^j_i]_p - [b^{j-1}_i]_p$.
End for                                                    (8.c)
$C \leftarrow 2^{L(m)} - m$        $\triangleright$ Note that $C$ is public.               (8.d)
For $i = 0, 1, ..., l^{(m)} - 1$ do
    $[t_i]^m_B \leftarrow ([t^{L(m)-1}_i]_p, ..., [t^1_i]_p, [t^0_i]_p)$
    If $m < 2^{L(m)}$ then    $\triangleright$Recall that $m < 2^{L(m)}$ means $m$ is not a power of 2.
        $[d_i]^m_B \leftarrow Bitwise - Subtraction^* \left([t_i]^m_B, \left([b^{L(m)-1}_i]_p ?C : 0\right)\right)$     (8.e)
    Else
        $[d_i]^m_B \leftarrow [t_i]^m_B$
    End if
End for                                                    (8.f)
$[x - y]^m_{D,B} = [d]^m_{D,B} \leftarrow ([d_{l^{(m)}-1}]^m_B, ..., [d_1]^m_B, [d_0]^m_B)$
**Return** $[x - y]^m_{D,B}$
```

Correctness is described in detail in [13]. Privacy follows readily from the fact that we only call private sub-protocols. The complexity of this protocol is 30 rounds and $47l\log l + 47l\log(L(m))$ multiplications. The communication complexity is upper bounded by $94l\log l$ multiplications since $L(m) \leq l$.

•A Simplified Version. If we do not need $[x-y]^m_{D,B}$ but (only) need $[x-y]^m_D$ instead, a simplified version of the above protocol, *Digit-Bit-wise-Subtraction*$^{*-}$, can be obtained by simply replacing all the statements after *statement* (8.a) with the following.

```
$[d_0]^m_p = [x_0]^m_p - [y_0]^m_p + m[b_0^{L(m)-1}]_p$
For $i = 1, ..., l^{(m)} - 1$ in parallel: $[d_i]^m_p = [x_i]^m_p - [y_i]^m_p + m[b_i^{L(m)-1}]_p - [b_{i-1}^{L(m)-1}]_p$
$[x-y]^m_D = [d]^m_D \leftarrow ([d_{l^{(m)}-1}]^m_p, ..., [d_1]^m_p, [d_0]^m_p)$
**Return** $[x-y]^m_D$
```

Note that the above process is free. Correctness and privacy is straightforward. The complexity of this protocol goes down to 15 rounds and $47l\log l$ multiplications as the expensive *Bitwise-Subtraction** protocol is omitted.

If this (simplified) protocol is constructed from scratch, then, for relatively large m, the borrow bits for every digit-position, i.e. $[b_i^{L(m)-1}]_p$ for $i \in \{0, 1, ..., l^{(m)} - 1\}$, can be obtained with a lower cost. For every digit-position $i \in \{0, 1, ..., l^{(m)} - 1\}$, $e_i \in \{S, P, K\}$ can be obtained by calling the linear primitive *Bitwise-LessThan*. Specifically, we have

$$e_i = S \Leftrightarrow [x_i]^m_B < [y_i]^m_B; \;\; e_i = P \Leftrightarrow [x_i]^m_B = [y_i]^m_B; \;\; e_i = K \Leftrightarrow [x_i]^m_B > [y_i]^m_B.$$

So, using the *Bitwise-LessThan* protocol *in both ways*, which costs $l + \sqrt{l}$ more multiplications and no more rounds than one single invocation [18], we can get all the $e_i's$. Then as in the *BORROWS* protocol (or the *CARRIES* protocol), the target borrow bits (for every digit-position) can be obtained by using a generic prefix protocol which costs 15 rounds and $47l^{(m)}\log l^{(m)}$ multiplications. So the *Digit-Bit-wise-Subtraction*$^{*-}$ protocol can be realized in 6+15=21 rounds and (less than) $16l + 47l^{(m)}\log\left(l^{(m)}\right)$ multiplications. Recall that $l^{(m)} = \lceil \log_m p \rceil$. Then for relatively large m, e.g. $m \approx p^{\frac{1}{10}}$ where $l^{(m)} = 10$, the communication complexity may be very low.

7 Comments

As in [12], although we describe all our protocols in the secret sharing setting, our techniques are also applicable to the threshold homomorphic setting. All the protocols in our paper can be similarly realized in this setting. However, some of the protocols in this setting may be less efficient than their counterpart in the secret sharing setting because the *Random-Bit* protocol, which is a basic building block, is more expensive in the threshold homomorphic setting.

It is easy to see that using our *Base-m Digit-Decomposition* protocol which extracts all the base-m digits of the shared input, we can also solve the modulo reduction problem (which requires only the least significant base-m digit). However, our *Modulo-Reduction* protocol is meaningful because it achieves linear communication complexity and thus is much more efficient.

Obviously, we can say that the bit-decomposition protocol (of [7]) is a special case of our *Base-m Digit-Bit-Decomposition* protocol when m is a power of 2. In fact, we can also view the bit-decomposition protocol as a special case of our *enhanced Modulo-Reduction protocol* when the modulus m is just p, i.e. we have

$$[x]_B = Bit - Decomposition([x]_p) = Modulo - Reduction^{+}([x]_p, p)$$

for any $x \in \mathbb{Z}_p$. Our *enhanced Modulo-Reduction* protocol can handle not only the special case where $m = p$ but also the general case where $m \in \{2, 3, ..., p-1\}$, so it can also be viewed as a generalization to bit-decomposition.

We note that, in [18], a novel technique is proposed which can reduce the communication complexity of the bit-decomposition protocol to *almost linear*. We argue that their technique can also be used in our *Base-m Digit-Bit-Decomposition protocol* (as well as our *Base-m Digit-Decomposition protocol*) to reduce the (communication) complexity to almost linear, because their technique is in fact applicable to any $PreFix-\circ$ (or $PostFix-\circ$) protocol (which is a dominant factor of the communication complexity) assuming a linear protocol for computing the $UnboundedFanIn - \circ$ exists, which is just the case in our protocols.

8 Applications and Future Work

In [13], we will show some applications of our new protocols, such as efficient *Integer Division* protocol, *Divisibility Test* protocol, *Conversion of Integer Representation between Number Systems*, etc.

Although we are successful in providing an (efficient) solution to the *public* modulo reduction problem, we fail in solving the *private* modulo reduction problem where the modulus is (also) secret shared. The absence of the knowledge of the exact value of m makes our techniques useless. We leave it an open problem to construct efficient protocols for private modulo reduction without relying on bit-decomposition.

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. 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)
3. 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)
4. Chandra, A.K., Fortune, S., Lipton, R.J.: Lower Bounds for Constant Depth Circuits for Prefix Problems. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 109–117. Springer, Heidelberg (1983)

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. Catrina, O., Saxena, A.: Secure Computation with Fixed-Point Numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (2010)
7. 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)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. Ning, C., Xu, Q.: Multiparty Computation for Modulo Reduction without Bit-Decomposition and A Generalization to Bit-Decomposition. Cryptology ePrint Archive, Report 2010/266, `http://eprint.iacr.org/2010/266`
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)