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

Chao Ning
School of computer science and technology, Shandong University, Jinan, China
Email: ncnfl@mail.sdu.edu.cn

Qiuliang Xu
School of computer science and technology, Shandong University, Jinan, China
Email: xuqiuliang@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 also propose a generalization to bit-decomposition 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 $m \geq 2$. Obviously, when *m* is a power of 2, this (generalized) protocol is just the original bit-decomposition protocol.

Keywords: Multiparty Computation, Constant Rounds, Secret Sharing, Bitwise Sharing, Digit-wise Sharing, 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 desired outputs but nothing else, and the privacy of their inputs are guaranteed. Although generic solutions for MPC already exist [BGW88, GMW87], the efficiency of these generic protocols tend to be low. So we focus on constructing efficient protocols for specific functions. More precisely, we are interested in integer arithmetic in the information theory setting [NO07].

A proper choice of the representation of the inputs can have great influence on the efficiency of the computation [DFK+06,Tof09]. 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 filed Z_p and perform the computation 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 desired result, then we will get a highly inefficient protocol as the bitwise addition and the bitwise multiplication are very expensive [CFL83a,CFL83b]. On the other hand, if we want to *compare* some (private) integer values, then the binary representation will be of great advantage for comparison is a *bit-oriented* operation. In this case, the arithmetic circuit over Z_p will be a bad choice.

To bridge the gap between the arithmetic circuits and the Boolean circuits, Damgård *et al.* [DFK+06] proposed a novel technique, called bit-decomposition, to convert a sharing of secret a into the sharings of the bits of a. This is a very useful tool in MPC because it gives us the best of the two worlds. For example, for a protocol built in the prime filed Z_p, if a series of bit-oriented operations (such as comparisons, computations of Hamming weight) are needed in the future process, we can, using bit-decomposition, transform the sharings of the integers into the sharings of the bits of the integers. Then, the future process can be handled easily. On the other hand, in a Boolean circuit, if we need a series of additions and multiplications of the integers (which are represented as bits), then we can (freely) transform the binary representation of these integers into elements of a prime field (e.g. Z_p), and perform all the additions and multiplications in the field. When the desired results are obtained (in the field), the bit-decomposition can be involved and the (aimed) binary representation of the results can be finally obtained.

Thus, bit-decomposition is useful both in theory and application. However, the bit-decomposition protocol is relatively expensive in terms of round and communication complexities [NO07]. So the work for constructing (constant rounds) protocols for MPC problems without relying on bit-decomposition is not only interesting but also meaningful. Recently, in [NO07], 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, in MPC, it remains an open problem whether the *modulo reduction* problem can be solved in constant rounds without bit-decomposition [Tof07]. 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 [DFK+06] 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 need 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, we propose a generalization to bit-decomposition, which we call *Base-m Digit-Bit-Decomposition*, and which can de-composite the sharing of secret *a* into the sharings of the base-*m* digits of *a*, along with the sharings of the bits of every digit (if desired).

Our Results. First we introduce some necessary notation. We focus mainly on the multi-party computation based on linear secret sharing schemes. Assume that the underlying secret sharing scheme is built on prime field Z_p where p is a prime with bit-length l (i.e. $l = \lceil \log p \rceil$). For secret $a \in Z_p$, 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. $[a]_B = \left([a_{l-1}]_p, ..., [a_1]_p, [a_0]_p\right)$.

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

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

i.e. given a sharing of secret x, i.e. $[x]_p$, and a public modulus $m \in \{2, 3, ..., p-1\}$, the parties compute the sharing of $x \bmod m$, i.e. $[x \bmod m]_p$.

In existing (public) modulo reduction protocols [DFK+06,Tof07], the bit-decomposition is involved, incurring (at least) $O(l \log l)$ communication complexity. What's more, in the worst case, the communication complexity of this protocol may goes up to $O(l^2)$. Specifically, the existing modulo reduction protocol uses the bit-decomposition protocol to reduce the "size" of the problem, and then uses as many as l *comparisons*, which is non-trivial, to determine the final result. If the bit-length of the inputs to the comparison protocol is relatively long, e.g. $\theta(l)$ which is often the case, then the overall complexity will go up to $O(l^2)$. So, the efficiency of the protocol may be very low. To solve this problem, in this paper, we propose a protocol, which achieves constant round complexity and linear communication complexity, for (public) modulo reduction without relying on bit-decomposition. What's more, in this paper, we not only propose a protocol for the original (public) modulo reduction problem (which outputs $[x \bmod m]_p$), but also proposed an enhanced protocol that can output the sharings of the bits of $x \bmod m$, i.e. $[x \bmod m]_B$.

Some primitives used in bit-decomposition are generalized to meet the requirements of our modulo reduction protocol. Using these generalized primitives and some other techniques, we also propose a generalization to bit-decomposition which can, in constant rounds, convert a 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*. Obviously, when *m* is a power of 2, our Base-m Digit-Bit-Decomposition protocol degenerates to the bit-decomposition protocol.

For visualization, we will give out an example here. Pick binary number

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

If the sharing of *a*, i.e. $[a]_p$, 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 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

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

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

We also propose a simplified version of the protocol, i.e. the *Base-m Digit-Decomposition Protocol* that, e.g. when given $[a]_p$ and $m = 10$ as inputs, outputs $\left([2]_p, [4]_p, [9]_p\right)$, i.e. the sharings of the base-10 digits of a.

Finally, we'd like to stress that all the protocols proposed in our paper are constant rounds and unconditionally secure, and our techniques can also be used to construct non-constant-rounds variations which may be preferable in practice.

Related Work. The problem of bit-decomposition is a basic problem in MPC and was partially solved by Algesheimer *et al.* in [ACS02]. 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 for the problem of bit-decomposition in [DFK+06]. This ice-break work is based on linear secret sharing schemes [BGW88,GRR98]. Independently, Shoenmakers and Tuyls [ST06] solved the problem of bit-decomposition for multiparty computation based on (Paillier) threshold homomorphic cryptosystems [DB03,CDN01]. In order to improve the efficiency, Nishide and Ohta [NO07] gave out a simplification to the bit-decomposition protocol of [DFK+06] by throwing off one unnecessary invocation of bitwise addition, which is the most expensive primitive in bit-decomposition. Moreover, they proposed solutions for comparison, interval test and equality test of shared secrets without relying on the expensive bit-decomposition protocol. Their techniques are novel, and enlightened us a lot. Recently, Toft showed a novel technique that can reduce the communication complexity of the bit-decomposition protocol to *almost linear* [Tof09]. Although we do not focus on the *almost linear* property of protocols, some techniques used in their paper are so inspiring and enlightening to us.

2 Preliminaries

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

2.1 Notation

We will now introduce all the important notation used in this paper.

The multiparty computation considered in this paper is based on linear secret sharing schemes, such as Shamir's. As mentioned above, we denote the underlying field as Z_p where *p* is a prime with binary length l (i.e. $l = \lceil \log p \rceil$).

As in previous works, such as [DFK+06] and [NO07], 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) $a \in Z_p$ 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 process the (secure) *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 involves another b multiplications in parallel, then we can say that the round complexity is 2 and the communication complexity is $a+b$ multiplications. Note that the complexity analysis made in this paper is somewhat rough for we focus mainly on the ideas of the solution, not on the details for implementation.

As in [NO07], 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 $L(m)=\lceil \log 10 \rceil = 4$, i.e. we must use 4 bits to represent a base-10 digit. Note that we have $2^{L(m)-1} < m \le 2^{L(m)}$ and $m=2^{L(m)}$ holds iff m is a power of 2.

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

For any $a \in Z_p$, the secret sharing of a is denoted by $[a]_p$. We use $[a]_B$ to denote the bitwise sharing of a, i.e. $[a]_B = \left([a_{l-1}]_p,...,[a_1]_p,[a_0]_p\right)$. When a is public, $[a]_B$ degenerates to the binary representation 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. Obviously, we have $0 \le a_i \le (m-1)$ for $i \in \{0,1,...,l^{(m)}-1\}$ (because a_i is a base-*m* digit).

The *digit-bit-wise sharing* of a, i.e. $[a]_{D,B}^m$, is defined as follows:

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

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

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

It's easy to see that if we have obtained the bitwise sharing of x, i.e. $[x]_B$, then the sharing of x, i.e. $[x]_p$, can be freely obtained by a linear combination of the sharings of the bits of x. I.e. given

$$[x]_B = \left([x_{l-1}]_p,...,[x_1]_p,[x_0]_p\right),$$

$[x]_p$ can be obtained as follows:

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

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 some protocols that can output both $[x]_B$ and $[x]_p$, which is often the case in this paper, we always output $[x]_B$ *only*, and $[x]_p$ is omitted for simplicity.

Sometimes, we 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.

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

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

$$\textit{if } b = 1, \textit{then } C \textit{ is set to } A; \textit{otherwise } (i.e.\, b = 0), C \textit{ is set to } B.$$

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 in this command are secret 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 process:

---

for $i = 0,1,...,l-1$ do ▷ Suppose that $|A| = |B| = |C| = l$

$[C_i]_p \leftarrow [b]_p([A_i]_p - [B_i]_p) + [B_i]_p$

end for

$[C]_B \leftarrow \left([C_{l-1}]_p, ..., [C_1]_p, [C_0]_p\right)$

Note that the above process costs 1 round, l invocations of multiplication.
Other cases, such as

$$"[C]_D^m \leftarrow [b]_p\,?[A]_D^m : [B]_D^m" \text{ and } "[C]_{D,B}^m \leftarrow [b]_p\,?[A]_{D,B}^m : [B]_{D,B}^m",$$

can be realized similarly. We will often 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.

2.2.1 Random-Bit

The *Random-Bit* protocol is the most basic primitive which can generate a sharing of a uniformly random bit unknown to all parties. In the linear secret sharing setting, which is the case in this paper, it costs only 2 rounds and 2 multiplications [DFK+06]. We denote this sub-protocol as $\text{Random-Bit}(\cdot)$.

2.2.2 Bitwise-LessThan

Given two bitwise shared inputs, $[x]_B$ and $[y]_B$, the *Bitwise-LessThan* protocol can compute a secret shared bit $[x \overset{?}{<} y]_p$ where $(x \overset{?}{<} y) = 1$ iff $x < y$ holds. The main part of this protocol is a prefix-OR, which costs linear (in l) number of multiplications. We note that using the method of [Tof09], this protocol can be realized in 6 rounds and $13l + 6\sqrt{l}$ multiplications. Note that $13l + 6\sqrt{l} \le 14l$ holds for $l \ge 36$ which is often the case in practice. So, for simplicity, we refer to the complexity of this protocol as 6 rounds and $14l$ multiplications. We denote this sub-protocol as $\text{Bitwise-LessThan}(\cdot)$.

2.2.3 Bitwise-Addition Protocol

Given two bitwise shared inputs, $[x]_B$ and $[y]_B$, the *Bitwise-Addition* protocol outputs $[d]_B = [x+y]_B$. One important point of this protocol is that $d = x+y$ holds over the integers, not (only) $\bmod p$. This protocol, which costs 15 rounds and $47l\log l$ multiplications, is the most expensive part of the bit-decomposition protocol of [DFK+06]. We will not use this protocol in this paper, but use the *Bitwise-Subtraction* protocol instead. However, the asymptotic complexity of our Bitwise-Subtraction protocol is the same with that of the Bitwise-Addition for they both involves a *generic prefix protocol*, which involves $O(l\log l)$ multiplications. We will introduce our Bitwise-Subtraction protocol later.

3 A Simple Introduction to Our New Primitives

In this section, we will simply introduce all 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 [Tof09] 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 that $x \ge y$. A run of this restricted protocol is denoted by

$$[d]_B \leftarrow \text{Bitwise-Subtraction}^*([x]_B,[y]_B)$$

where $d = x-y$. It costs 15 rounds and $47l\log l$ multiplications [NO07].

- BORROWS. This protocol is used in the *Bitwise-Subtraction* protocol above (as well as in the *Bitwise-Subtraction** protocol) to compute the borrow bits. Although this protocol is an important sub-protocol in the Bit-Subtraction protocol and some other protocols of our paper, it will be only sketched in Section 6 because it is highly similar to the *CARRIES* protocol proposed in [DFK+06]. We will only describe the differences between them in Section 6. Given two bitwise sharings $[x]_B$ and $[y]_B$, this protocol outputs

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

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

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

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

where $d \in \{0,1,\ldots,m-1\}$ represents a base-m digit. Note that $[d]_p^m$ is implicitly obtained. This protocol costs 8 rounds and $16L(m)$ multiplications.

- Digit-Bit-wise-LessThan. The *Digit-Bit-wise-LessThan* protocol accepts two digit-bit-wise shared values

$$[x]_{D,B}^m = \left([x_{l^{(m)}-1}]_B^m,\ldots,[x_1]_B^m,[x_0]_B^m\right) \text{ and } [y]_{D,B}^m = \left([y_{l^{(m)}-1}]_B^m,\ldots,[y_1]_B^m,[y_0]_B^m\right)$$

and outputs

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

where $(x \overset{?}{<} y) = 1$ iff $x < y$. The complexity of this protocol is 6 rounds and $14l$ multiplications.

- Random-Solved-Digits-Bits. Using the above two primitives, i.e. *Random-Digit-Bit* and *Digit-Bit-wise-LessThan*, as sub-protocols, we can construct the *Random-Solved-Digits-Bits* protocol which, when given

$$m \in \{2,3,\ldots,p-1\}$$

as input, outputs a digit-bit-wise shared random value

$$[r]^m_{D,B} = ([r_{l^{(m)}-1}]^m_B, ..., [r_1]^m_B, [r_0]^m_B)$$

satisfying $r < p$. We denote a run of this protocol by

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

This protocol costs 14 rounds and $78l$ multiplications.

• Digit-Bit-wise-Subtraction. This protocol is novel generalization to the bitwise subtraction protocol. It accepts two digit-bit-wise shared values

$$[x]^m_{D,B} = \left([x_{l^{(m)}-1}]^m_B, ..., [x_1]^m_B, [x_0]^m_B\right) \text{ and } [y]^m_{D,B} = \left([y_{l^{(m)}-1}]^m_B, ..., [y_1]^m_B, [y_0]^m_B\right)$$

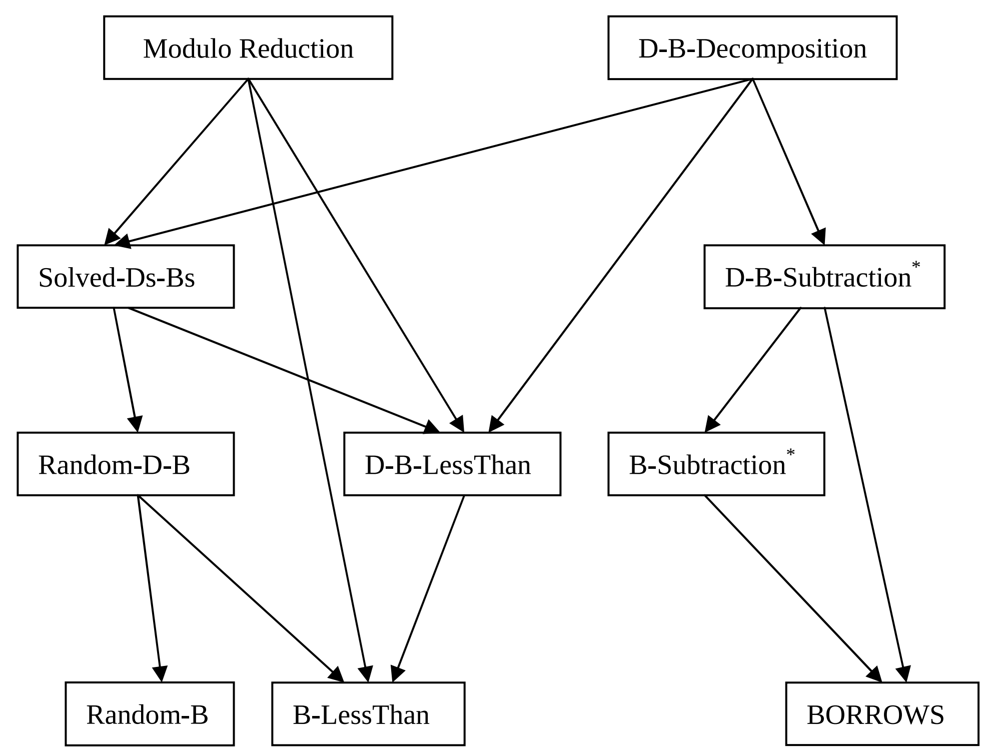
and outputs $[x-y]^m_{D,B}$. As is the case in *Bitwise-Subtraction*, we need only the restricted version, i.e. the *Digit-Bit-wise-Subtraction** protocol which requires $x \geq y$. A run of this (restricted) protocol is denoted by

$$[d]^m_{D,B} \leftarrow \text{Digit-Bit-wise-Subtraction}^*([x]^m_{D,B}, [y]^m_{D,B}),$$

where $d = x - y$. This restricted protocol costs 30 rounds and $47l\log l + 47l\log\left(L(m)\right)$ multiplications. What's more, if we don't need $[x-y]^m_{D,B}$ but need (only) $[x-y]^m_D$ instead (i.e. we do not need the bitwise sharing of the digits of the difference), then the *Digit-Bit-wise-Subtraction** protocol can be (greatly) simplified by dropping the expensive *Bitwise-Subtraction** protocol used in it. We denote a run of this (further) restricted protocol by

$$[d]^m_D \leftarrow \text{Digit-Bit-wise-Subtraction}^{*-}([x]^m_{D,B}, [y]^m_{D,B})$$

where $d = x - y$. The complexity of this protocol goes down to 15 rounds and $47l\log l$ multiplications.



Notes:
1. In this figure, "B" means "Bit" or "Bitwise"; "D" means "Digit".
2. For two protocols *A* and *B*, $A \rightarrow B$ means "*A* uses *B* as a sub-protocol".

Figure 1. *Protocol hierarchy*

Using the above primitives, we can construct our *Modulo Reduction* protocol and *Base-m Digit-Bit-Decomposition* protocol. These two protocols will be described in detail *separately* in Section 4 and Section 5. The dependency between the building blocks is shown in Figure 1.

4 Multiparty Computation for Modulo Reduction without Bit-Decomposition

In this section, we will 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 in fact the *Least Significant Digit Protocol*. 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, $\text{Modulo-Reduction}(\cdot)$, for computing the residue of a secret shared value modulo a public modulus.

Input: A secret shared value $[x]_p$ with $x \in Z_p$ and a public modulus $m \in \{2,3,...,p-1\}$.

Output: $[x \bmod m]_p$.

Process:

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

$c \leftarrow \mathit{reveal}([x]_p + [r]_{D,B}^m)$ ▷ Note that $[r]_{D,B}^m$ implies $[r]_p^m$. (1.a)

$[X_1]_p^m \leftarrow [c_0]_p^m - [r_0]_B^m$ ▷ Note that $[r_0]_B^m$ implies $[r_0]_p^m$.

$[X_2]_p^m \leftarrow [c_0]_p^m - [r_0]_B^m + m$

$[s]_p \leftarrow \text{Bitwise-LessThan}([c_0]_B^m, [r_0]_B^m)$ (1.b)

$[X]_p^m \leftarrow [s]_p \, ?[X_2]_p^m : [X_1]_p^m$ ▷ Recall that this is a *conditional selection command*.

$c' \leftarrow c + p$ ▷ Addition over the integers.

$[X_1']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m$

$[X_2']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m + m$

$[s']_p \leftarrow \text{Bitwise-LessThan}([c_0']_B^m, [r_0]_B^m)$ (1.c)

$[X']_p^m \leftarrow [s']_p \, ?[X_2']_p^m : [X_1']_p^m$

$[t]_p \leftarrow \text{Digit-Bit-wise-LessThan}(c, [r]_{D,B}^m)$ (1.d)

$[x \bmod m]_p = [x_0]_p^m \leftarrow [t]_p \, ?[X']_p^m : [X]_p^m$

return $[x \bmod m]_p$

Correctness: By simulating a *base-m addition* process, the protocol extracts (the sharing of) *the least significant base-m digit* of x, i.e. x_0 which is just $x \bmod m$. Specifically, we use an *addition* in line (1.a) to randomized the secret input $[x]_p$. Note that the digit-bit-wise representation of c, i.e. $[c]_{D,B}^m$, can be obtained without communication because c is public. The addition in line (1.a) can be viewed as a base-*m* addition. Using the *Bitwise-LessThan protocol* in line (1.b) and (1.c), we can decide whether a carry is set at the least significant digit when performing the base-*m* addition. What's more, we use the *Digit-Bit-wise-LessThan* protocol in line (1.d) to decide whether a wrap-around modulo p occurs when performing the (whole) base-*m* addition. Using these "knowledge", we can *select* the correct value for the least significant base-*m* digit of x, i.e. $[x \bmod m]_p$. So the correctness can be convinced.

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 tells no information

about the secret x. So the privacy can be convinced.
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 proceed in parallel. In all it will cost 22 rounds and

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

multiplications. Note that $L(m) \le l$ as $2 \le m \le p-1$, 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 $\text{Modulo-Reduction}^+(\cdot)$. The construction is seen as Protocol 2.

Protocol 2. The Modulo Reduction$^+$ protocol, $\text{Modulo-Reduction}^+(\cdot)$, for computing the *bitwise sharing* of the residue of a secret shared value modulo a public modulus.

Input: A secret shared value $[x]_p$ with $x \in Z_p$ and a public modulus $m \in \{2, 3, ..., p-1\}$.
Output: $[x \bmod m]_B$.
Process:

$[r]^m_{D,B} \leftarrow \text{Random-Solved-Digits-Bits}(m)$
$c \leftarrow reveal([x]_p + [r]^m_{D,B})$
$[\bar{M}_1]^m_B \leftarrow [c_0]^m_B \quad [\bar{S}_1]^m_B \leftarrow [r_0]^m_B$
$[\bar{M}_2]^m_B \leftarrow [c_0 + m]^m_B \quad [\bar{S}_2]^m_B \leftarrow [r_0]^m_B$ ▷ Note that the addition is over the integers.
$[s]_p \leftarrow \text{Bitwise-LessThan}([c_0]^m_B, [r_0]^m_B)$
$[\bar{M}]^m_B \leftarrow [s]_p \,?[\bar{M}_2]^m_B : [\bar{M}_1]^m_B$ ▷ Note that this command involves $L(m)$ multiplications.
$[\bar{S}]^m_B \leftarrow [s]_p \,?[\bar{S}_2]^m_B : [\bar{S}_1]^m_B$

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

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

$[x \bmod m]_B = [x_0]^m_B \leftarrow \text{Bitwise-Subtraction}^*([M]^m_B, [S]^m_B)$ ▷ Subtraction over the integers.
return $[x \bmod m]_B$

The correctness and the privacy of this protocol can be convinced similarly to that of the *Modulo Reduction* protocol above. By carefully selecting the *Minuend* and the *Subtrahend* (for the *Bitwise-Subtraction** protocol), we can get the desired result by using just one invocation of the *Bitwise-Subtraction** protocol. As for complexity, note that every *conditional selection command* in this protocol involves $L(m)$ multiplications, whereas in the previous protocol (i.e. the Modulo Reduction protocol) it involves only 1 multiplication. The overall complexity of this protocol is 37 rounds and

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

multiplications. Obviously, when $L(m)$ is large enough, e.g. $L(m) = l$, the asymptotic (communication) complexity of this protocol may goes up to $O(l\log l)$, which is the asymptotic (communication) complexity of the bit-decomposition protocol. We argue that this is *inevitable* because when m is large enough, this protocol becomes an enhanced bit-decomposition protocol. We will describe this in detail in Section 7.

5 The Base-*m* Digit-Bit-Decomposition Protocol — 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. All the details of this protocol are presented in Protocol 3. The main framework of this protocol is similar to that of the bit-decomposition protocol in [Tof09].

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

Input: A secret shared value $[x]_p$ with $x \in Z_p$ and the base *m*.

Output: $[x]_{D,B}^m = ([x_{l^{(m)}-1}]_B^m, ..., [x_1]_B^m, [x_0]_B^m)$, in which $[x_i]_B^m = ([x_i^{L(m)-1}]_p, ..., [x_i^1]_p, [x_i^0]_p)$ for $i \in \{0,1,...,l^{(m)}-1\}$.

Process:

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

$c \leftarrow \mathit{reveal}([x]_p + [r]_{D,B}^m)$ (3.a)

$c' \leftarrow c + p$

$[t]_p \leftarrow \text{Digit-Bit-wise-LessThan}(c, [r]_{D,B}^m)$ (3.b)

for $i = 0,1,...,l^{(m)}-1$ do ▹ To get $[\tilde{c}]_{D,B}^m = [t]_p\, ?[c']_{D,B}^m : [c]_{D,B}^m$.

$[\tilde{c}_i]_B^m \leftarrow [t]_p\, ?[c_i']_B^m : [c_i]_B^m$

end for

$[\tilde{c}]_{D,B}^m \leftarrow ([\tilde{c}_{l^{(m)}-1}]_B^m, ..., [\tilde{c}_1]_B^m, [\tilde{c}_0]_B^m)$ ▹ *Note that* $\tilde{c} = x + r$

$[x]_{D,B}^m \leftarrow \text{Digit-Bit-wise-Subtraction}^*([\tilde{c}]_{D,B}^m, [r]_{D,B}^m)$ (3.c)

return $[x]_{D,B}^m$

Correctness: Using the *Digit-Bit-wise-LessThan* protocol in line **(3.b),** we can decide whether a wrap-around modulo *p* occurs when performing the addition in line **(**3.a**)**. Basing on this “knowledge” we can *select* the correct value for $\tilde{c} = x + r$. Then, using the *Digit-Bit-wise-Subtraction** protocol in line **(**3.c**)**, with the digit-bit-wise sharing of $\tilde{c} = x + r$ and r as inputs, we can get the desired output, i.e. $[x]_{D,B}^m$. In fact, the major problem to overcome in constructing this protocol is how to realize the *Digit-Bit-wise-Subtraction** *protocol*, which will be described in detail in Section 6.

Privacy: Privacy can be convinced for we only call private sub-protocols.

Complexity: There are only three sub-protocols that count for complexity, i.e. the *Random-Solved-Digits-Bits* protocol, the *Digit-Bit-wise-LessThan* protocol, and the *Digit-Bit-wise-Subtraction** protocol. So, the overall complexity of this protocol is $14 + 6 + 30 = 50$ rounds and

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

multiplications.

Similar to the case in the *Digit-Bit-wise-Subtraction** *protocol*, 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, which is used at the end of the protocol, with the *Digit-Bit-wise-Subtraction**$^{-}$ protocol. We call this simplified protocol the *Base-m Digit-Decomposition Protocol*. A run of this protocol is denoted by $\text{Digit-Decomposition}(\cdot, m)$. The correctness and the privacy of this protocol can be similarly convinced. 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 $l^{(m)}=\lceil \log_m p \rceil \le l$. So the communication complexity is upper bounded by $342l+47l\log l$.

6 Realizing the Primitives

In this section, we will describe in detail the 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 [DFK+06] from base-2 to base-*m* for any $m \ge 2$. Note that, when m is a power of 2, some of our primitives degenerate to the existing primitives in [DFK+06]. So, in the complexity analysis, we focus on the case where m is not a power of 2, i.e. the case where $m < 2^{L(m)}$.

6.1 Bitwise-Subtraction

We describe the *Bitwise-Subtraction* protocol here. In fact, this protocol is already proposed in [Tof09]. However, they realized the protocol in a widely different manner to ours. They reduced the problem of *Bitwise-Subtraction* to the *Post-fix Comparison* problem and solved it in $O(l\log l)$ (communication) complexity. Here, we re-consider the problem of Bitwise-Subtraction and solve it in a manner which is highly similar to that of the *Bitwise-Addition* protocol of [DFK+06].

As is mentioned in Section 3, we will first propose a restricted (bitwise-subtraction) protocol which requires that *the minuend is not less than the subtrahend*. We will use this restricted version in all the protocols proposed in this paper. We call it the *Bitwise-Subtraction** protocol. The general version of bitwise-subtraction, which does not have the above restriction, can be realized with the help of the *Bitwise-LessThan* protocol. We will introduce this in detail later.

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 Bitwise-Subtraction* protocol, $\text{Bitwise-Subtraction}^*(\cdot)$, for computing the (bitwise sharing of the) difference of two bitwise shared values. This protocol requires that the minuend is not less than the subtrahend.

Input: Two bitwise shared values

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

satisfying $x \ge y$.

Output: $[x-y]_B=[d]_B=([d_{l-1}]_p,\ldots,[d_1]_p,[d_0]_p)$, i.e. the bitwise shared difference of the two inputs.

Process:

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

$[d_0]_p \leftarrow [x_0]_p-[y_0]_p+2[b_0]_p$

for $i=1,2,\ldots,l-1$ do

$\quad [d_i]_p \leftarrow [x_i]_p-[y_i]_p+2[b_i]_p-[b_{i-1}]_p$

end for

$[x-y]_B=[d]_B \leftarrow ([d_{l-1}]_p,\ldots,[d_1]_p,[d_0]_p)$ $\triangleright$ Note that $[d]_B$ has only l bits as $x \ge y$

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 $x \geq y$ and thus we do not need a sign bit.

Correctness is straightforward. Privacy follows readily from the fact that we only call private sub-protocols. The complexity of this protocol is the same with that of the *Bitwise-Addition* protocol in [DFK+06], i.e. 15 rounds and $47l\log l$ multiplications [NO07].

Although in all the protocols proposed in this paper we will only use the *Bitwise-Subtraction** protocol above, we also give out the general version, i.e. the *Bitwise-Subtraction* protocol. For arbitrary

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

the *Bitwise-Subtraction* protocol computes

$$[x-y]_B = [d]_B = ([d_l]_p,[d_{l-1}]_p,...,[d_1]_p,[d_0]_p),$$

where $[d_l]_p$ is the sharing of the sign bit, as follows.

First compute $[(x \overset{?}{<} y)]_p$ using the *Bitwise-LessThan* protocol. Then set

$$[X]_B \leftarrow [(x \overset{?}{<} y)]_p \,?[y]_B : [x]_B \text{ and } [Y]_B \leftarrow [(x \overset{?}{<} y)]_p \,?[x]_B : [y]_B .$$

It can be verified that $X \geq Y$ always holds. Finally set

$$([d_{l-1}]_p,...,[d_1]_p,[d_0]_p) = \text{Bitwise-Subtraction}^*([X]_B,[Y]_B)$$

and set $[d_l]_p = [(x \overset{?}{<} y)]_p$ as the sign bit. Then

$$[x-y]_B = [d]_B = ([d_l]_p,[d_{l-1}]_p,...,[d_1]_p,[d_0]_p)$$

is obtained.

Correctness and privacy can be easily verified. Comparing to the *Bitwise-Subtraction** protocol above, this protocol costs 7 additional rounds and $16l$ additional multiplications.

6.2 Computing the Borrow Bits

We now describe the *BORROWS* protocol which can compute the borrow bits. In fact our *BORROWS* protocol is highly similar to the *CARRIES* protocol in [DFK+06]. So the difference is only sketched here. As in [DFK+06], 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 [DFK+06], 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 $i = 0,1,\ldots,l-1$, 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 that 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 the value of e_i for every bit-position $i \in \{0,1,...,l-1\}$ is obtained, the rest of the process of the *BORROWS* protocol will be (completely) the same with that of the *CARRIES* protocol. So, the only difference between our *BORROWS* protocol and the *CARRIES* protocol lies in the process of computing e_i for every bit-position $i \in \{0,1,...,l-1\}$, which will be sketched in the following.

As in [DFK+06], we represent *S*, *P* and *K* with bit vectors

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

Then, given two inputs (to the *BORROWS* protocol)

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

the $[e_i]_B = ([s_i]_p, [p_i]_p, [k_i]_p)$ for $i \in \{0,1,...,l-1\}$ 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 of shared variables (i.e. $[x_i]_p[y_i]_p$).

Privacy is straightforward because nothing is revealed in the protocol. Correctness follows readily from the above arguments. The complexity of the protocol is the same with that of the *Bitwise-Subtraction** protocol above, i.e. 15 rounds and $47l\log l$ multiplications [NO07,DFK+06].

6.3 Random-Digit-Bit

We now introduce the *Random-Digit-Bit* protocol for generating a random bitwise shared base-*m* digit, which is denoted by d here, for any $m \geq 2$. Obviously, d is in fact a random integer satisfying $0 \leq d \leq m-1$. Note that the output of this protocol is not the sharing of d, but the sharings of the bits of d. The knowledge of (the sharings of) the bits of d helps us a lot in constructing other primitives. The protocol is presented in Protocol 5.

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

Input: The base *m* satisfying $2 \leq m \leq p-1$.

Output: $[d]_B^m = ([d^{L(m)-1}]_p, ..., [d^1]_p, [d^0]_p)$, i.e. the bitwise sharing of a base-*m* digit d, with $0 \leq d \leq m-1$.

Process:

for $i = 0,1,\ldots,L(m)-1$ do
  $[d^i]_p \leftarrow \text{Random-Bit}()$
end for
$[d]_B^m \leftarrow ([d^{L(m)-1}]_p, ..., [d^1]_p, [d^0]_p)$
if $m = 2^{L(m)}$ then ▹ Note that $m = 2^{L(m)}$ means m is a power of 2.
  return $[d]_B^m$ (5.a)
end if
$[r]_p \leftarrow \text{Bitwise-LessThan}([d]_B^m, m)$
$r \leftarrow reveal([r]_p)$ (5.b)
if $r = 0$ then
  protocol fails, abort
else
  return $[d]_B^m$
end if

Correctness: To generate a base-*m* digit, the protocol generates $L(m)$ random secret shared bits first (recall that $L(m) = \lceil \log m \rceil$ is the binary length of a base-*m* digit). Note that in line (5.a), a *return* command is involved. This means if $m = 2^{L(m)}$ holds, then all the commands after line (5.a) will not be run. When $m = 2^{L(m)}$ does not hold, using the *Bitwise-LessThan* protocol, the protocol checks whether the generated random integer lies in set $\{0,1,...,m-1\}$, which is a basic requirement for a base-*m* digit.

Privacy: The only information leakage takes place in line (5.b), where a *reveal* is involved. However, the revealed message, i.e. r, can only tell the parties that the digit d lies in $\{0,1,...,m-1\}$, which is already known to everyone.

Complexity: The protocol needs $L(m)$ invocations of *Random-Bit* (in parallel), and one invocation of *Bitwise-LessThan*. So, 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 [DFK+06], 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, $\text{Digit-Bit-wise-LessThan}(\cdot)$, for comparing two digit-bit-wise shared values.

Input: Two digit-bit-wise shared values

$$[x]_{D,B}^{m} = ([x_{l^{(m)}-1}]_B^m, \ldots, [x_1]_B^m, [x_0]_B^m) \quad \text{and} \quad [y]_{D,B}^{m} = ([y_{l^{(m)}-1}]_B^m, \ldots, [y_1]_B^m, [y_0]_B^m).$$

Output: $[(x \overset{?}{<} y)]_p$, i.e. the sharing of the bit $(x \overset{?}{<} y) \in \{0,1\}$, where $(x \overset{?}{<} y) = 1$ iff $x < y$ holds.

Process:

$$\begin{aligned}[X]_B \leftarrow (&[x_{l^{(m)}-1}^{L(m)-1}]_p, \ldots, [x_{l^{(m)}-1}^{1}]_p, [x_{l^{(m)}-1}^{0}]_p,\\ &\ldots\\ &\ldots\\ &\ldots,\\ &[x_1^{L(m)-1}]_p, \ldots, [x_1^1]_p, [x_1^0]_p,\\ &[x_0^{L(m)-1}]_p, \ldots, [x_0^1]_p, [x_0^0]_p).\end{aligned} \qquad \triangleright \text{ Viewing } [x]_{D,B}^m \text{ as binary number } [X]_B.$$

$$\begin{aligned}[Y]_B \leftarrow (&[y_{l^{(m)}-1}^{L(m)-1}]_p, \ldots, [y_{l^{(m)}-1}^{1}]_p, [y_{l^{(m)}-1}^{0}]_p,\\ &\ldots\\ &\ldots\\ &\ldots,\\ &[y_1^{L(m)-1}]_p, \ldots, [y_1^1]_p, [y_1^0]_p,\\ &[y_0^{L(m)-1}]_p, \ldots, [y_0^1]_p, [y_0^0]_p).\end{aligned} \qquad \triangleright \text{ Viewing } [y]_{D,B}^m \text{ as binary number } [Y]_B.$$

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

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

Correctness: In the protocol, we view the *Digit-Bit-wise representation* of x and y as two binary numbers (i.e. X and Y defined in the protocol). When $m < 2^{L(m)}$, the two binary numbers (i.e. X and Y) are of course not equal to the original numbers (i.e. x and y). But, when comparing x and y this is allowed because, both in the *digit-bit-wise representation* case and in the *binary* case, the relationship between the size of two numbers is determined by the left-most differing bits of them. So, we can say that $x < y \Leftrightarrow X < Y$ and the correctness can be convinced.

Privacy: The privacy follows from only using private sub-protocols.

Complexity: The complexity of the protocol is the same with that of the *Bitwise-LessThan* protocol. Note that the length of the inputs to the *Bitwise-LessThan* protocol, i.e. $[X]_B$ and $[Y]_B$, is

$$L(m)\cdot l^{(m)}=\lceil \log m\rceil\cdot\lceil \log_m p\rceil=\lceil \log m\rceil\cdot\left\lceil \frac{\log p}{\log m}\right\rceil\approx \log p=l\,.$$

So, the overall complexity of this protocol is 6 rounds and (about) $14l$ multiplications.

6.5 Random-Solved-Digits-Bits Protocol

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 [DFK+06]. The details are presented in Protocol 7.

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

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

for $i=0,1,...,l^{(m)}-1$ do
  $[r_i]_B^m \leftarrow \mathrm{Random\text{-}Digit\text{-}Bit}(m)$
end for
$[r]_{D,B}^m \leftarrow ([r_{l^{(m)}-1}]_B^m,...,[r_1]_B^m,[r_0]_B^m)$
$[c]_p \leftarrow \mathrm{Digit\text{-}Bit\text{-}wise\text{-}LessThan}([r]_{D,B}^m,[p]_{D,B}^m)$
$c \leftarrow \mathit{reveal}([c]_p)$ (7.a)
if $c=0$ then
  protocol fails, abort
else
  return $[r]_{D,B}^m$
end if

Note that for an integer r satisfying $r<p$, the (bit) length of the digit-bit-wise representation of r should be the same with (the (bit) length of the digit-bit-wise representation of) p. In this protocol, for simplicity, we assume that $[p_{l^{(m)}-1}^{L(m)-1}]_p^m=1$. Under this assumption, we can generate $[r_{l^{(m)}-1}]_B^m$ using the *Random-Digit-Bit* protocol. If $[p_{l^{(m)}-1}^{L(m)-1}]_p^m=0$, then $[r_{l^{(m)}-1}]_B^m$ can be generated using the *Random-Bit* protocol directly.

The correctness and the privacy is straightforward. The protocol uses $l^{(m)}$ invocations of *Random-Digit-Bit* and one invocation of *Digit-Bit-wise-LessThan*. So, the total complexity of one run of this protocol is 8+6=14 rounds and $l^{(m)}\cdot 64L(m)+14l=78l$ multiplications. Similar to the *Random-Digit-Bit* protocol above, if this protocol has to be repeated in parallel to get a lower abort probability, then the round complexity is still 14, and the amortized communication complexity goes up to $4\times 78l=312l$ multiplications**.**

6.6 Digit-Bit-wise-Subtraction Protocol

We will describe in detail a restricted version of the *Digit-Bit-wise-Subtraction* protocol, i.e. the *Digit-Bit-wise-Subtraction** protocol which requires that the minuend is not less than the subtrahend. The general version, i.e. the *Digit-Bit-wise-Subtraction* protocol, which can be realized using the techniques in Section 6.1, and which is not used in the paper, is omitted for simplicity.

6.6.1 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. In the process of this protocol, similar to the case in the *Digit-Bit-wise-LessThan* protocol, we will sometimes view the *digit-bit-wise representation* of an integer as a *binary number* directly. We will explain this in detail later. The process of the protocol is presented in Protocol 8.

Protocol 8. The Digit-Bit-wise-Subtraction* protocol, $\text{Digit-Bit-wise-Subtraction}^*(\cdot)$, for computing the digit-bit-wise sharing of the difference of two digit-bit-wise shared values with the minuend no less than the subtrahend.

Input: Two digit-bit-wise shared values

$$[x]_{D,B}^m = ([x_{l^{(m)}-1}]_B^m, ..., [x_1]_B^m, [x_0]_B^m) \text{ and } [y]_{D,B}^m = ([y_{l^{(m)}-1}]_B^m, ..., [y_1]_B^m, [y_0]_B^m).$$

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:

$$\begin{aligned}
[X]_B \leftarrow ([x_{l^{(m)}-1}^{L(m)-1}]_p, ..., [x_{l^{(m)}-1}^1]_p, [x_{l^{(m)}-1}^0]_p,& \\
&...\\
&...\\
&...,\\
&[x_1^{L(m)-1}]_p, ..., [x_1^1]_p, [x_1^0]_p,\\
&[x_0^{L(m)-1}]_p, ..., [x_0^1]_p, [x_0^0]_p).
\end{aligned}$$

$$\begin{aligned}
[Y]_B \leftarrow ([y_{l^{(m)}-1}^{L(m)-1}]_p, ..., [y_{l^{(m)}-1}^1]_p, [y_{l^{(m)}-1}^0]_p,& \\
&...\\
&...\\
&...,\\
&[y_1^{L(m)-1}]_p, ..., [y_1^1]_p, [y_1^0]_p,\\
&[y_0^{L(m)-1}]_p, ..., [y_0^1]_p, [y_0^0]_p).
\end{aligned}$$

$$\begin{aligned}
&([b_{l^{(m)}-1}^{L(m)-1}]_p, ..., [b_{l^{(m)}-1}^1]_p, [b_{l^{(m)}-1}^0]_p,\\
&...\\
&...\\
&...,\\
&[b_1^{L(m)-1}]_p, ..., [b_1^1]_p, [b_1^0]_p,\\
&[b_0^{L(m)-1}]_p, ..., [b_0^1]_p, [b_0^0]_p) \leftarrow \text{BORROWS}([X]_B, [Y]_B).
\end{aligned} \tag{8.a}$$

$$[t_0^0]_p = [x_0^0]_p - [y_0^0]_p + 2[b_0^0]_p \tag{8.b}$$

for $j = 1, ..., L(m)-1$ do

$[t_0^j]_p = [x_0^j]_p - [y_0^j]_p + 2[b_0^j]_p - [b_0^{j-1}]_p$

end for

for $i = 1, ..., l^{(m)}-1$ do

$[t_i^0]_p = [x_i^0]_p - [y_i^0]_p + 2[b_i^0]_p - [b_{i-1}^{L(m)-1}]_p$

for $j = 1, ..., L(m)-1$ do

$[t_i^j]_p = [x_i^j]_p - [y_i^j]_p + 2[b_i^j]_p - [b_i^{j-1}]_p$

end for

end for (8.c)

$$C \leftarrow 2^{L(m)} - m \qquad \triangleright \textit{Note that } C \textit{ is public.} \tag{8.d}$$

for $i = 0,1,...,l^{(m)}-1$ do

$[t_i]_B^m \leftarrow ([t_i^{L(m)-1}]_p,...,[t_i^1]_p,[t_i^0]_p)$

if $m < 2^{L(m)}$ then ▷ Recall that $m < 2^{L(m)}$ means m is not a power of 2.

$[d_i]_B^m \leftarrow \text{Bitwise-Subtraction}^*\left([t_i]_B^m, \left([b_i^{L(m)-1}]_p ? C : 0\right)\right)$ (8.e)

else

$[d_i]_B^m \leftarrow [t_i]_B^m$

end if

end for (8.f)

$[x-y]_{D,B}^m = [d]_{D,B}^m \leftarrow ([d_{l^{(m)}-1}]_B^m,...,[d_1]_B^m,[d_0]_B^m)$

return $[x-y]_{D,B}^m$

Correctness: When calling the *BORROWS* protocol, we view the *digit-bit-wise representation* of x and y as two *binary numbers* (i.e. X and Y). This is sensible because of the following.

For any two binary numbers

$$[S]_B = \left([S_{l-1}]_p,...,[S_1]_p,[S_0]_p\right) \text{ and } [T]_B = \left([T_{l-1}]_p,...,[T_1]_p,[T_0]_p\right)$$

and any bit-position i, the fact that "A borrow is set at position i" is equivalent to the fact that "$[S]_{(i,...,1,0)} = \left([S_i]_p,...,[S_1]_p,[S_0]_p\right)$ is less than $[T]_{(i,...,1,0)} = \left([T_i]_p,...,[T_1]_p,[T_0]_p\right)$". What's more, as is mentioned in Section 6.4, both in the *digit-bit-wise representation* case and in the *binary* case, the relationship between the size of two numbers is determined by the left-most differing bits of them. So, concluding the above, we can say that the fact that "A borrow is set at (bit) position i *in the digit-bit-wise representation case*" is equivalent to the fact that "A borrow is set at (bit) position i *in the binary case*". So, we can get the correct borrow bits by calling the *BORROWS* protocol with $[X]_B$ and $[Y]_B$ as inputs. Note that for $i \in \{0,1,...,l^{(m)}-1\}$, $[b_i^{L(m)-1}]_p$ is in fact the bit borrowed by the $i'th$ digit (from the $(i+1)'th$ digit).

From line (8.b) to line (8.c), we calculate every bit (of the difference) *as in the binary case*. This is of course not right when $m < 2^{L(m)}$ (i.e. m is not a power of 2) because for a base-*m* number, a "1" in the $(i+1)'th$ digit corresponds to "m" in the $i'th$ digit, not "$2^{L(m)}$". An example is as follows.

When $m = 10$, we have $L(m) = \lceil \log 10 \rceil = 4$, i.e. we should use 4 bits to represent a base-10 digit. In this case $2^{L(m)} = 2^4 = 16$. If the least significant digit d_0 borrows a "1" from d_1, then d_0 should view this "1" as "10" (which is the base *m*), not "16" (which is $2^{L(m)}$).

From line (8.b) to line (8.c), we (temporarily) ignore the above problem and calculate every bit as in the binary case. Then, to get the final result, we use the commands from line (8.d) to line (8.f) to "revise" the result. Specifically, if $m < 2^{L(m)}$, then for every digit-position $i \in \{0,1,...,l^{(m)}-1\}$, when $[b_i^{L(m)-1}]_p = 1$, which means the $i'th$ digit borrows a "1" from the $(i+1)'th$ digit, we set $[d_i]_B^m = [t_i]_B^m - \left(2^{L(m)} - m\right)$.

Privacy: Privacy follows readily from the fact that we only call private sub-protocols.

Complexity: There are only two commands that count for complexity. One is the *BORROWS* protocol in line (8.a), the other is the *Bitwise-Subtraction** *protocol* in line (8.e). The length of the inputs to the *BORROWS* protocol is $L(m) \cdot l^{(m)} \approx l$, so this sub-protocol costs 15 rounds and $47l\log l$ multiplications; when $m < 2^{L(m)}$, the *Bitwise-Subtraction** *protocol* is involved $l^{(m)}$ times (with inputs of length $L(m)$) and costs 15 rounds and

$$l^{(m)} \times 47 \times L(m)\log\left(L(m)\right) \approx 47l\log\left(L(m)\right)$$

multiplications. The total complexity of this protocol is 30 rounds and $47l\log l + 47l\log\left(L(m)\right)$ multiplications. Note that $L(m) \le l$ as $2 \le m \le p-1$, so the communication complexity is upper bounded by $94l\log l$ multiplications.

6.6.2 A Simplified Version—the Digit-Bit-wise-Subtraction^{*-} Protocol

If we do not need $[x-y]^m_{D,B}$ but only need $[x-y]^m_D$ instead, then a simplified version of the above protocol (i.e. Protocol 8), which we denote by $\text{Digit-Bit-wise-Subtraction}^{*-}(\cdot)$, can be obtained by simply replacing all the commands after line (8.a) (in Protocol 8) with the following commands.

$[d_0]^m_p = [x_0]^m_p - [y_0]^m_p + m[b_0^{L(m)-1}]_p$

for $i = 1,...,l^{(m)}-1$ do

$\quad [d_i]^m_p = [x_i]^m_p - [y_i]^m_p + m[b_i^{L(m)-1}]_p - [b_{i-1}^{L(m)-1}]_p$

end for

$[x-y]^m_D = [d]^m_D \leftarrow ([d_{l^{(m)}-1}]^m_p,...,[d_1]^m_p,[d_0]^m_p)$

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 for the expensive *Bitwise-Subtraction** protocol is omitted.

If this 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 in 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

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

So, using the *Bitwise-LessThan* protocol *in both ways*, which will cost $l+\sqrt{l}$ more multiplications and no more rounds than one single invocation of Bitwise-LessThan [Tof09], we can get $e_i \in \{S,P,K\}$ for every digit-position $i \in \{0,1,...,l^{(m)}-1\}$. Then as in the *BORROWS* protocol (or the *CARRIES* protocol), the aimed 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

In this section, we will make some comments on the protocols of this paper.

As in [NO07], although we describe all our protocols in the secret sharing setting, our techniques are also applicable to the threshold homomorphic setting. All the primitives in *Figure* 1 can be re-constructed 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-Bit-Decomposition* protocol (or the *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 is obviously much more efficient.

Obviously, we can say that the bit-decomposition protocol (of [DFK+06]) is a special case of our *Base-m Digit-Bit-Decomposition* protocol where m is a power of 2. In fact, we can also view the bit-decomposition protocol as a special case of our *Modulo Reduction*$^+$ *protocol* where the modulus *m* is just *p*, i.e. we have

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

for any $x \in Z_p$. We can say that our *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, our *Modulo Reduction*$^+$ protocol can also be viewed as a generalization to bit-decomposition.

We note that, in [Tof09], a novel technique is proposed which can reduce the communication complexity of the bit-decomposition protocol to *almost linear*. We can argue that their technique can also be used in our *Base-m Digit-Bit-Decomposition* protocol (and our *Base-m Digit-Decomposition* protocol) to reduce the (communication) complexity to almost linear, because their technique is in fact applicable to any $PreFix\text{-}\circ$ (or $PostFix\text{-}\circ$) protocol (which is a dominant factor of the communication complexity) assuming a linear protocol for computing the $Unbounded\ Fan\text{-}In\ \circ$ exists, which is just the case in our protocols.

8 Applications

We will introduce some applications of our protocols in this section. All the protocols propose in this section are constant rounds and unconditionally secure. Recall that in this paper we focus on integer arithmetic in the information theory setting. The underlying linear secret sharing scheme is built in prime field Z_p where *p* is a prime with bit-length *l* (i.e. $l = \lceil \log p \rceil$).

8.1 Efficient Integer Division Protocol

Given a secret shared value $[x]_p$ and a public modulus m, the integer division protocol

$$int\_div : Z_p \to Z_p,\ x \mapsto \left\lfloor \frac{x}{m} \right\rfloor$$

can be realized efficiently using our *Modulo Reduction* protocol. Denote $t = \left\lfloor \frac{x}{m} \right\rfloor$, then we have $x = tm + (x \bmod m)$. So we can see that, if $x \bmod m$ can be obtained in linear communication complexity, which is just the case in our *Modulo Reduction* protocol, then $t = \left\lfloor \frac{x}{m} \right\rfloor$ can also be obtained in linear complexity by setting $t = \left(x - (x \bmod m)\right)\left(m^{-1} \bmod p\right) \bmod p$.

8.2 Efficient Divisibility Test Protocol

The divisibility test problem can be formalized as follows:

$$[m \overset{?}{|} x]_p \leftarrow \text{Divisibility-Test}([x]_p, m),$$

where $x \in Z_p$, $m \in \{2,3,...,p-1\}$ and $(m \overset{?}{|} x) = 1$ iff m is a factor of x.

Obviously, $(m \overset{?}{|} x) = 1 \iff (x \bmod m) = 0$. So, in a divisibility test protocol, the parties need only to obtain the residue $x \bmod m$ first and then decide whether the residue is 0. We provide two options for this task in the following.

Option 1: First the parties get $[x \bmod m]_p$ using our *Modulo Reduction* protocol. Then using the *Equality Test* protocol or the *Probabilistic Equality Test* protocol in [NO07], which is realized without bit-decomposition and achieves constant rounds and linear communication complexity, the parties can determine whether $(x \bmod m) = 0$. When the *Equality Test* protocol of [NO07] is used, the total complexity of the above process is $8 + 22 = 30$ rounds and

$$81l + (326l + 28L(m) + 3) \approx 407l + 28L(m) \text{ multiplications.}$$

Option 2: Using our *Modulo Reduction*$^+$ protocol, the parties can get

$$[x \bmod m]_B = ([t^{L(m)-1}]_p, ..., [t^1]_p, [t^0]_p) .$$

Then the parties can compute $[x \bmod m]_B \stackrel{?}{=} 0$ as $\prod_{i=0}^{L(m)-1} (1-[t^i]_p)$ by using an unbounded fan-in *And* [DFK+06]. The overall complexity of "Option 2" is $5+37=42$ rounds and

$$5l + \left(326l + 34L(m) + 47L(m)\log\left(L(m)\right)\right) = 331l + 34L(m) + 47L(m)\log\left(L(m)\right)$$

multiplications.

Recall that $L(m) \le l$. Thus the communication complexity of "Option 1" is always linear (in l). However this is not the case in "Option 2". In fact, when m is large enough, e.g. $L(m) = l$, the asymptotic communication complexity (of "Option 2") goes up to $O(l \log l)$. However, when m is relatively small, e.g. $m = 10$ which is often the case in practice, "Option 2" can be a better choice.

8.3 Conversion of Integer Representation between Number Systems

In multiparty computation, being able to converting integer representation between different number systems is meaningful both in theory and application. This can be done using our *Base-m Digit-Decomposition* protocol. For example, given the sharings of the *base-M digits* of integer x , i.e. $[x]_D^M$, the parties can obtain the sharings of the *base-N digits* of x , i.e. $[x]_D^N$, as follows. (Note that $M, N \in \{2,3,..., p-1\}$). First get the sharing of x , i.e. $[x]_p$. This can be easily done by a linear combination which is free. Then by running the protocol $\text{Digit-Decomposition}([x]_p, N)$, the parties can get the desired result, i.e. $[x]_D^N$.

8.4 Base-10 Applications

Given a secret shared value $[x]_p$ and $m = 10$ as inputs, our *Base-m Digit-Decomposition* protocol (or our *Base-m Digit-Bit-Decomposition* protocol) can output the sharings of the base-10 digits of x . This is meaningful because in real life, integers are (almost always) encoded base-10. We believe that, in multiparty computation for practical use, being able to de-composite a secret shared integer into (the sharings of) its base-10 digits will provide us with a lot of convenience.

9 Conclusion and Future Work

In this paper, we have solved the open problem whether the public modulo reduction problem can be realized without relying on the bit-decomposition protocol. We propose an efficient protocol (i.e. the *Modulo Reduction* protocol) that can solve this problem in constant rounds and linear communication complexity. What's more, we generalize the bit-decomposition protocol, which is a powerful tool for multiparty computation, to the *Base-m Digit-Bit-Decomposition* protocol, which can convert the sharing of secret a into the sharings of the base-*m* digits of a , along with the bitwise sharing of every digit. We believe that our *Modulo Reduction* protocol and *Base-m Digit-Bit-Decomposition* protocol will be useful both in theory and application.

Although we are successful in providing an (efficient) solution for 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 will try to propose an efficient protocol for the private modulo reduction problem in the future.

Acknowledgements

The authors wish to thank anonymous reviewers for giving helpful suggestions. This work is supported by the National Nature Science Foundation of China under Grant No. 60873232, and the National Nature Science Foundation of Shandong Province under Grant No. Y2007G37.

References

[ACS02] 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)

[BGW88] 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)

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen, "Multiparty computation from threshold homomorphic encryption," EUROCRYPT'01, LNCS 2045, pp.280–300, Springer Verlag, 2001.

[CFL83a] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Lower bounds for constant depth circuits for prefix problems. In Proceedings of ICALP 1983, pages 109 – 117. Springer-Verlag, 1983. Lecture Notes in Computer Science Volume 154.

[CFL83b] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded fanin circuits and associative functions. In 15th Annual ACM Symposium on Theory of Computing, pages 52 – 60, Boston, Massachusetts, USA, April 25 – 27, 1983. ACM Press.

[DB03] I. Damgård and J.B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," CRYPTO'03, LNCS 2729, pp.247–264, Springer Verlag, 2003.

[DFK+06] 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)

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a complete theorem for protocols with honest majority," Proc. 19th STOC, pp.218–229, 1987.

[GRR98] R. Gennaro, M.O. Rabin, and T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," Proc. 17th ACM Symposium on Principles of Distributed Computing, pp.101–110, 1998.

[NO07] 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) [Sha79] Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)

[ST06] 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)

[Tof07] Toft, T.: Primitives and Applications for Multi-party Computation. PhD thesis, University of Aarhus (2007), http://www.daimi.au.dk/~ttoft/publications/dissertation.pdf

[Tof09] 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)