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 x, it allows the parties to compute the sharings of the bits of x 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 x into the sharings of the digits of x, 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, Modulo Reduction, Generalization to Bit-Decomposition.

This is the full version of the paper in Proc. AsiaCrypt'2010.

*Supported by the National Natural Science Foundation of China under Grant No. 60873232.
†Corresponding author.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Related Work . . . 5

2 Preliminaries **5**
- 2.1 Notations and Conventions . . . 5
- 2.2 Known Primitives . . . 7

3 A Simple Introduction to Our New Primitives **7**

4 Multiparty Computation for Modulo Reduction without Bit-Decomposition **9**

5 A Generalization to Bit-Decomposition **11**

6 Realizing the Primitives **12**
- 6.1 Bitwise-Subtraction . . . 12
- 6.2 Computing the Borrow Bits . . . 13
- 6.3 Random-Digit-Bit . . . 14
- 6.4 Digit-Bit-wise-LessThan . . . 15
- 6.5 Random-Solved-Digits-Bits . . . 16
- 6.6 Digit-Bit-wise-Subtraction . . . 16

7 Comments **19**

8 Applications **19**
- 8.1 Efficient Integer Division Protocol . . . 20
- 8.2 Efficient Divisibility Test Protocol . . . 20
- 8.3 Conversion of Integer Representation between Number Systems . . . 21
- 8.4 Base-10 Applications . . . 21

9 Conclusion and Future Work **21**

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 [BGW88, GMW87], the efficiency of these generic 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 [NO07].

A proper choice of 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 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 [CFL83a, CFL83b]. 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.* [DFK$^+$06] proposed a novel technique, called bit-decomposition, to convert a sharing of secret x into the sharings of the bits of x. 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 field $\mathbb{Z}_p$, if a series of bit-oriented operations (such as comparisons, computations of Hamming weight) are needed in the future procedure, we can, using bit-decomposition, transform the sharings of the integers into the sharings of the bits of the integers. Then, the future procedure 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. $\mathbb{Z}_p$) and perform all the additions and multiplications in this field. When the expected results are obtained (in the field), 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. 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 [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 x into the sharings of the *bits* of x. However, especially in practice, we may often need the sharings of the *digits* of x. 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 on field $\mathbb{Z}_p$ where p is a prime with bit-length l (i.e. $l = \lceil \log p \rceil$). For secret $x \in \mathbb{Z}_p$, we use $[x]_p$ to denote the secret sharing of x, and $[x]_B$ to denote the sharings of the bits of x, i.e. $[x]_B = \Big([x_{l-1}]_p, ..., [x_1]_p, [x_0]_p\Big)$.

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

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

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

In existing public modulo reduction protocols [DFK+06, Tof07], the bit-decomposition protocol 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. $\Theta(l)$, 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. $[x \text{ mod } m]_B$.

Some primitives involved in bit-decomposition are generalized to meet the requirements of our modulo reduction protocol. Using these generalized primitives and some other techniques, we also construct a generalized bit-decomposition protocol which can, in constant rounds, convert the sharing of secret x into the sharings of the digits of x, 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

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

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

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

if $[x]_p$ 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 $[x]_p$ 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*, which outputs $\Big([2]_p, [4]_p, [9]_p\Big)$ when given $[x]_p$ and $m = 10$ as inputs.

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.

1.2 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 to the problem of bit-decomposition in [DFK$^+$06]. This ice-break work is based on linear secret sharing schemes [BGW88, GRR98]. Independently, Shoenmakers and Tuyls [ST06] solved the problem of bit-decomposition for multiparty computation based on (Paillier) threshold homomorphic cryptosystems [CDN01, DN03]. In order to improve efficiency, Nishide and Ohta [NO07] proposed a simplification to the bit-decomposition protocol of [DFK$^+$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 have enlightened us a lot. Recently, Toft showed a novel technique that can reduce the communication complexity of bit-decomposition 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. In a followup work, Reistad and Toft proposed a linear bit-decomposition protocol [RT09]. 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 [GMS10]. In [CS10], 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 [Sha79]. 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 [DFK$^+$06] and [NO07], we assume that the underlying secret sharing scheme allows to compute $[x + y \bmod p]_p$ from $[x]_p$ and $[y]_p$ without communication, and that it allows to compute $[xy \bmod p]_p$ from (public) $x \in \mathbb{Z}_p$ and $[y]_p$ without communication. We also assume that the secret sharing scheme allows to compute $[xy \bmod p]_p$ from $[x]_p$ and $[y]_p$ through communication among the parties. We call this procedure the *secure multiplication* (or *multiplication* for simplicity). Obviously, for multiparty protocols, this 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 [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 $[x \stackrel{?}{<} y]_p$ to denote the output of the comparison protocol, i.e. $(x \stackrel{?}{<} y) = 1$ 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 $x \in \mathbb{Z}_p$, the secret sharing of x is denoted by $[x]_p$. We use $[x]_B$ to denote the bitwise sharing of x.

We use $[x]_D^m = \left([x_{l^{(m)}-1}]_p^m, ..., [x_1]_p^m, [x_0]_p^m\right)$ to denote the *digit-wise sharing* of x. For $i \in \{0, 1, ..., l^{(m)} - 1\}$, $[x_i]_p^m$ denotes the sharing of the $i'th$ base-m digit of x with $0 \leq x_i \leq (m-1)$.

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

$$[x]_{D,B}^m = \left([x_{l^{(m)}-1}]_B^m, ..., [x_1]_B^m, [x_0]_B^m\right),$$

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

Sometimes, if m can be inferred from the context, we may write $[x_i]_p^m$ (or $[x_i]_B^m$) as $[x_i]_p$ (or $[x_i]_B$) 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 $[x]_{D,B}^m = \left([x_{l^{(m)}-1}]_B^m, ..., [x_1]_B^m, [x_0]_B^m\right)$, then $[x]_D^m = \left([x_{l^{(m)}-1}]_p^m, ..., [x_1]_p^m, [x_0]_p^m\right)$ 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 $c \leftarrow \text{Reveal}([c]_p)$.

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* [Tof07]. 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: $[C_i]_p \leftarrow [b]_p([A_i]_p - [B_i]_p) + [B_i]_p$
$[C]_B \leftarrow \Big([C_{l-1}]_p, ..., [C_1]_p, [C_0]_p\Big)$ $\qquad \triangleright |A| = |B| = |C| = l$

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 [DFK+06].

- ***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 [Tof09], this protocol can be realized in 6 rounds and $13l + 6\sqrt{l}$ multiplications. Notice 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.

- ***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 [DFK+06]. 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 [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 $x \ge y$. A run of this *restricted* protocol is denoted by

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

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

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

where $[b_i]_p$ is the sharing of the borrow bit at bit-position $i \in \{0, 1, ..., l-1\}$. The complexity of this protocol is 15 rounds and $47l \log l$ multiplications.

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

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

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 $64L(m)$ 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

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

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

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

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

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

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

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

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 [NO07]. Recall that for any integer x, the sharing of the least significant base-m digit of x is denoted by $[x_0]_p^m$, and the bitwise sharing of the least significant base-m digit of x is denoted by $[x_0]_B^m$. The protocol is described in detail in Figure 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: $[x \bmod m]_p$.
Process:
$[r]_{D,B}^m \leftarrow$ Random-Solved-Digits-Bits(m)
$c \leftarrow$ Reveal$([x]_p + [r]_{D,B}^m)$ ▷Note that $[r]_{D,B}^m$ implies $[r]_p$. (1.a)
$[X_1]_p^m \leftarrow [c_0]_p^m - [r_0]_B^m$ ▷ $[r_0]_B^m$ 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)
$[X]_p^m \leftarrow [s]_p?[X_2]_p^m : [X_1]_p^m$ ▷A *conditional selection command.*

$c' \leftarrow c + p$ ▷Addition over the integers.
$[X_1']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m$
$[X_2']_p^m \leftarrow [c_0']_p^m - [r_0]_B^m + m$
$[s']_p \leftarrow$ 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)
$[x \bmod m]_p = [x_0]_p^m \leftarrow [t]_p?[X']_p^m : [X]_p^m$
Return $[x \bmod m]_p$

Figure 1: The Modulo Reduction Protocol *without* Bit-Decomposition

Correctness: By simulating a *base-m addition* procedure, the protocol extracts $[x_0]_p^m$ which is just $[x \bmod m]_p$. Specifically, we use an *addition* in line (1.a) to randomize the secret input $[x]_p$. This addition 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-position 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) addition. Using these "knowledge", we can "select" the correct value for $[x_0]_p^m$.

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 $L(m) \le l$, 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$^+(\cdot)$. Although we can get $[x \bmod m]_B$ by using the bit-decomposition protocol with $[x \mod m]_p$ as input, our enhanced *Modulo-Reduction* protocol, which is constructed from scratch, is meaningful because it is more efficient. The construction is seen as Figure 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$([x]_p + [r]^m_{D,B})$
$[\bar{M}_1]^m_B \leftarrow [c_0]^m_B$ $\quad$ $[\bar{S}_1]^m_B \leftarrow [r_0]^m_B$
$[\bar{M}_2]^m_B \leftarrow [c_0 + m]^m_B$ $\quad$ $[\bar{S}_2]^m_B \leftarrow [r_0]^m_B$ $\quad$ ▷Addition over the integers.
$[s]_p \leftarrow$ Bitwise-LessThan$([c_0]^m_B, [r_0]^m_B)$
$[\bar{M}]^m_B \leftarrow [s]_p?[\bar{M}_2]^m_B : [\bar{M}_1]^m_B$ $\quad$ ▷Involving $L(m)$ multiplications.
$[\bar{S}]^m_B \leftarrow [s]_p?[\bar{S}_2]^m_B : [\bar{S}_1]^m_B$

$c' \leftarrow c + p$
$[\bar{M}'_1]^m_B \leftarrow [c'_0]^m_B$ $\quad$ $[\bar{S}'_1]^m_B \leftarrow [r_0]^m_B$
$[\bar{M}'_2]^m_B \leftarrow [c'_0 + m]^m_B$ $\quad$ $[\bar{S}'_2]^m_B \leftarrow [r_0]^m_B$
$[s']_p \leftarrow$ 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$(c, [r]^m_{D,B})$
$[M]^m_B \leftarrow [t]_p?[\bar{M}']^m_B : [\bar{M}]^m_B$ $\quad$ ▷M is the minuend.
$[S]^m_B \leftarrow [t]_p?[\bar{S}']^m_B : [\bar{S}]^m_B$ $\quad$ ▷S is the subtrahend.

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

Figure 2: The *Enhanced* Modulo Reduction Protocol

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. As for the complexity, note that every *conditional selection command* in this protocol involves $L(m)$ multiplications, whereas in the *Modulo-Reduction* protocol it involves only 1 multiplication. The overall complexity of this protocol is 37 rounds and

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

multiplications. Obviously, when $L(m)$ is large enough (e.g. $L(m) = l$), the asymptotic (communication) complexity of this protocol goes up to $O(l \log l)$, which is the asymptotic 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 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 Figure 3. The main framework of this protocol is similar to the bit-decomposition protocol of [Tof09].

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)
$[\tilde{c}]^m_{D,B} = [t]_p?[c']^m_{D,B} : [c]^m_{D,B}$ $\quad \triangleright\ \tilde{c} = x + r$ holds over the integers
$[x]^m_{D,B} \leftarrow$ Digit-Bit-wise-Subtraction$^*([\tilde{c}]^m_{D,B}, [r]^m_{D,B})$ (3.c)
Return $[x]^m_{D,B}$

Figure 3: The Base-m Digit-Bit-Decomposition Protocol

As for the 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). Based 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 expected result, i.e. $[x]^m_{D,B}$. In fact, the major obstacle in constructing this protocol is how to realize the *Digit-Bit-wise-Subtraction** protocol, which is described in detail in Section 6.6.

Privacy is straightforward.

As for the complexity, it is easy to see that 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

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

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 $l^{(m)} = \lceil \log_m p \rceil \le l$, 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 [DFK$^+$06] from base-2 to base-m for any $m \ge 2$. It will be seen 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. $m < 2^{L(m)}$.

6.1 Bitwise-Subtraction

We describe the *Bitwise-Subtraction* protocol here. In fact, this protocol is already proposed in [Tof09]. 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 [DFK$^+$06].

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. We will describe 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 Figure 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 $x \ge y$.
Output: $[x-y]_B = [d]_B = ([d_{l-1}]_p, ..., [d_1]_p, [d_0]_p)$.
Process:
$([b_{l-1}]_p, ..., [b_1]_p, [b_0]_p) \leftarrow \text{BORROWS}([x]_B, [y]_B)$
$[d_0]_p \leftarrow [x_0]_p - [y_0]_p + 2[b_0]_p$
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 $[x-y]_B$

Figure 4: The Bitwise-Subtraction* Protocol

Note that the output of this protocol, i.e. $[x-y]_B$, is of bit length l, not $l+1$. This is because $x \ge y$ holds 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. With (arbitrary) inputs

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

the *Bitwise-Subtraction* protocol outputs

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

in which $[d_l]_p$ is the sharing of the sign bit. The procedure is presented below.

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

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

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

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

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

and set $[d_l]_p \leftarrow [(x \overset{?}{<} y)]_p$ 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 with the *Bitwise-Subtraction** protocol, this protocol costs 7 additional rounds and $16l$ additional multiplications. So the overall complexity of this protocol is 22 rounds and $47l \log l + 16l$ 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 [DFK$^+$06]. So only the difference is sketched here. As in [DFK$^+$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 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 [DFK$^+$06]). So, the only difference lies in the procedure of computing the $e_i's$, which will be sketched below.

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, 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 because nothing is revealed in the protocol. The complexity of the protocol is the same with the CARRIES protocol in [DFK$^+$06], i.e. 15 rounds and $47l\log l$ multiplications [NO07].

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 $0\le d\le m-1$. We would like to stress that the output of this protocol is not (only) 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 details are presented in Figure 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 $i=0,1,\dots,L(m)-1$ 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** $[d]_B^m$.

Figure 5: The Random-Digit-Bit Protocol

As for the correctness, to generate a base-m digit, the protocol generates $L(m)$ random shared bits first. Then using *Bitwise-LessThan*, the protocol checks whether the generated random integer is less than m, which is a basic requirement for a base-m digit. Note that when m is a power of 2 (i.e. $m=2^{L(m)}$), this check is unnecessary because an $L(m)$-bit binary number is at most $2^{L(m)}-1$ and thus is always less than m.

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, note that the protocol contains $L(m)$ 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 Figure 6.

> The *Digit-Bit-wise-LessThan* protocol, Digit-Bit-wise-LessThan$(\cdot)$, for comparing two digit-bit-wise shared values.
>
> ---
>
> **Input**: $[x]^m_{D,B} = ([x_{l^{(m)}-1}]^m_B, ..., [x_1]^m_B, [x_0]^m_B)$ and $[y]^m_{D,B} = ([y_{l^{(m)}-1}]^m_B, ..., [y_1]^m_B, [y_0]^m_B)$.
>
> **Output**: $[(x \stackrel{?}{<} y)]_p$, where $(x \stackrel{?}{<} y) = 1$ iff $x < y$ holds.
>
> **Process**:
>
> $$\begin{aligned}[X]_B \leftarrow (&[x^{L(m)-1}_{l^{(m)}-1}]_p, ..., [x^1_{l^{(m)}-1}]_p, [x^0_{l^{(m)}-1}]_p,\\ &...\\ &...\\ &...\\ &[x^{L(m)-1}_1]_p, ..., [x^1_1]_p, [x^0_1]_p,\\ &[x^{L(m)-1}_0]_p, ..., [x^1_0]_p, [x^0_0]_p)\end{aligned}$$
>
> $$\begin{aligned}[Y]_B \leftarrow (&[y^{L(m)-1}_{l^{(m)}-1}]_p, ..., [y^1_{l^{(m)}-1}]_p, [y^0_{l^{(m)}-1}]_p,\\ &...\\ &...\\ &...\\ &[y^{L(m)-1}_1]_p, ..., [y^1_1]_p, [y^0_1]_p,\\ &[y^{L(m)-1}_0]_p, ..., [y^1_0]_p, [y^0_0]_p)\end{aligned}$$
>
> $[(x \stackrel{?}{<} y)]_p = [(X \stackrel{?}{<} Y)]_p \leftarrow$ Bitwise-LessThan$([X]_B, [Y]_B)$
>
> **Return** $[(x \stackrel{?}{<} y)]_p$

Figure 6: The Digit-Bit-wise-LessThan Protocol

As for the correctness, we will simply explain why $x < y \iff X < Y$. In this protocol, we view the *Digit-Bit-wise representation* of x and y as two binary numbers (i.e. X and Y). 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). However, 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 $x < y \iff X < Y$ and the correctness is guaranteed.

Privacy follows readily from only using private sub-protocols.

The complexity of the protocol is *the same* with the *Bitwise-LessThan* protocol involved. The length of the inputs to the *Bitwise-LessThan* protocol (i.e. $[X]_B$ and $[Y]_B$) is

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

So, the overall complexity of this 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 [DFK$^+$06]. The details are presented in Figure 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**: $[r]^m_{D,B}$, in which r is a uniformly random value satisfying $r < p$.
> **Process**:
> For $i = 0, 1, ..., l^{(m)} - 1$ in parallel: $[r_i]^m_B \leftarrow$ Random-Digit-Bit(m).
> $[r]^m_{D,B} \leftarrow ([r_{l^{(m)}-1}]^m_B, ..., [r_1]^m_B, [r_0]^m_B)$
> $[c]_p \leftarrow$ Digit-Bit-wise-LessThan$([r]^m_{D,B}, [p]^m_{D,B})$
> $c \leftarrow$ Reveal$([c]_p)$
> If $c = 0$, then abort. Otherwise **Return** $[r]^m_{D,B}$.

Figure 7: The Random-Solved-Digits-Bits protocol Protocol

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 $[p_{l^{(m)}-1}]^m_B$. 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 $[r_{l^{(m)}-1}]^m_B$ using the *Random-Digit-Bit* protocol. If $p^{L(m)-1}_{l^{(m)}-1} = 0$, then $[r_{l^{(m)}-1}]^m_B$ can be generated by using the *Random-Bit* protocol directly.

The correctness and the privacy is straightforward. As for the complexity, 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

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. 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 why we can do this in detail later. The details of the protocol is presented in Figure 8.

Correctness: When calling the *BORROWS* protocol, we view the *digit-bit-wise representation* of x and y as two *binary numbers* X and Y. This does make sense because of the following. For any two binary numbers

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

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

$$
\begin{array}{l}
([b^{L(m)-1}_{l^{(m)}-1}]_p, ..., [b^1_{l^{(m)}-1}]_p, [b^0_{l^{(m)}-1}]_p, \\
... \\
... \\
... \\
[b^{L(m)-1}_1]_p, ..., [b^1_1]_p, [b^0_1]_p, \\
[b^{L(m)-1}_0]_p, ..., [b^1_0]_p, [b^0_0]_p) \leftarrow \text{BORROWS}([X]_B, [Y]_B)
\end{array}
\qquad (8.a)
$$

$[t^0_0]_p = [x^0_0]_p - [y^0_0]_p + 2[b^0_0]_p$ (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.
    $[d_i]^m_B \leftarrow \text{Bitwise-Subtraction}^*\left([t_i]^m_B, \left([b^{L(m)-1}_i]_p ? C : 0\right)\right)$ (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 $[x-y]^m_{D,B}$

Figure 8: The Digit-Bit-wise-Subtraction* Protocol

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

and any bit-position i, the fact that "A borrow is set at position i" is equivalent to the fact that

$$[S]_{(i,...,1,0)} = \Big([S_i]_p, ..., [S_1]_p, [S_0]_p\Big) \text{ is less than } [T]_{(i,...,1,0)} = \Big([T_i]_p, ..., [T_1]_p, [T_0]_p\Big).$$

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 $m < 2^{L(m)}$ because, in this case, a "1" in the $(i+1)'th$ digit corresponds to "m" in the $i'th$ digit, not "$2^{L(m)}$". Here is an example.

When $m = 10$, we have $L(m) = \lceil \log 10 \rceil = 4$, i.e. we 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 $[d_i]_B^m = [t_i]_B^m - \big(2^{L(m)} - m\big)$.

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

Complexity: There are only two sub-protocols that count for complexity. One is the *BORROWS* in *statement* (8.a), the other is the *Bitwise-Subtraction** 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 $l^{(m)} \times 47 \times L(m) \log(L(m)) \approx 47l \log(L(m))$ multiplications. Thus, the total 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 $L(m) \le l$.

•**A Simplified Version** If we do not need $[x-y]_{D,B}^m$ but (only) need $[x-y]_D^m$ 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.

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

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

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

So, using the *Bitwise-LessThan* protocol *in both ways*, which costs $l + \sqrt{l}$ more multiplications and no more rounds than one single invocation [Tof09], 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 [NO07], 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 [DFK$^+$06]) 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

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

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 [Tof09], 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

Here we show some applications of our protocols. All application protocols proposed here 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 field $\mathbb{Z}_p$ where p is a prime with bit-length l.

8.1 Efficient Integer Division Protocol

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

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

can be realized efficiently using our *Modulo-Reduction* protocol. Denote $t = \lfloor \frac{x}{m} \rfloor$, then we have $x = tm + (x \mod m)$. 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 $t = \lfloor \frac{x}{m} \rfloor$ can also be obtained in linear complexity by setting

$$t = (x - (x \mod m)) \left(m^{-1} \mod p\right) \mod p.$$

8.2 Efficient Divisibility Test Protocol

The divisibility test problem can be formalized as follows:

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

where $x \in \mathbb{Z}_p$, $m \in \{2, 3, ..., p-1\}$, $(m \stackrel{?}{\mid} x) \in \{0, 1\}$ and $(m \stackrel{?}{\mid} x) = 1$ iff m is a factor of x. Obviously,

$$(m \stackrel{?}{\mid} x) = 1 \quad \Leftrightarrow \quad (x \mod m) = 0.$$

So, in a divisibility test protocol, the parties need only to obtain (the sharing or the bitwise sharing of) $x \bmod m$ and then decide whether it is 0. We provide two options for this task below.

Option 1: First the parties get $[x \mod m]_p$ 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 round complexity and linear communication complexity, the parties can determine whether $(x \mod m) = 0$ holds. When the *Equality-Test* protocol of [NO07] is involved, the total complexity of the above procedure is $8+22 = 30$ rounds and $81l+(326l+28L(m)+3) \approx 407l + 28L(m)$ multiplications.

Option 2: Using our *enhanced Modulo-Reduction* protocol, the parties can get

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

Then the parties can compute $[x \mod m]_B \stackrel{?}{=} 0$ 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

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

multiplications.

Recall that $L(m) \leq l$. Thus the communication complexity of "Option 1" is always linear. 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 convert integer representation between different number systems is of both theoretical and practical interest. 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. 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 Digit-Decomposition($[x]_p, N$), the parties can get the expected result, i.e. $[x]_D^N$.

8.4 Base-10 Applications

Given a shared value $[x]_p$ and $m = 10$ as inputs, our *Base-m Digit-Decomposition* protocol (as well as 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 give an answer to the open problem whether the *public modulo reduction problem* can be solved without relying on bit-decomposition. We propose a linear, constant-rounds protocol (i.e. the *Modulo-Reduction* protocol) for the public modulo reduction problem without relying on bit-decomposition. 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 x into the sharings of the base-m digits of x, 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 (i.e. the base m) 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

[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] 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)

[CFL83a] 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)

[CFL83b] 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)

[CS10] 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)

[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)

[DN03] 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)

[GMS10] 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)

[GMW87] 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)

[GRR98] 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)

[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)

[RT09] 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)

[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)