Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol

Takashi Nishide[1,2] and Kazuo Ohta[1]

[1] Department of Information and Communication Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka Chofu-shi, Tokyo 182-8585 Japan
{t-nishide,ota}@ice.uec.ac.jp

[2] Hitachi Software Engineering Co., Ltd.; 4-12-7 Higashi-Shinagawa Shinagawa-ku, Tokyo, 140-0002 Japan

Abstract. Damgård *et al.* [11] showed a novel technique to convert a polynomial sharing of secret a into the sharings of the bits of a in constant rounds, which is called the bit-decomposition protocol. The bit-decomposition protocol is a very powerful tool because it enables bit-oriented operations even if shared secrets are given as elements in the field. However, the bit-decomposition protocol is relatively expensive.
In this paper, we present a simplified bit-decomposition protocol by analyzing the original protocol. Moreover, we construct more efficient protocols for a comparison, interval test and equality test of shared secrets without relying on the bit-decomposition protocol though it seems essential to such bit-oriented operations. The key idea is that we do computation on secret a with c and r where $c = a + r$, c is a revealed value, and r is a random bitwise-shared secret. The outputs of these protocols are also shared without being revealed.
The realized protocols as well as the original protocol are constant-round and run with less communication rounds and less data communication than those of [11]. For example, the round complexities are reduced by a factor of approximately 3 to 10.

Key words: Multiparty Computation, Secret Sharing, Bitwise Sharing

1 Introduction

Secure *multiparty computation* (MPC) allows a set of mutually distrustful parties to jointly compute an agreed function of their inputs in such a way that the correctness of the output and the privacy of the parties' inputs are guaranteed. That is, when a function is represented as $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$, each party with its private input x_i obtains only the output y_i but nothing else.

A great deal of work (e.g., [3, 17, 7, 25]) has been done in this research field. By using generic circuit based protocols, it is shown that any function can be computed securely [3, 17]. However, the general protocols tend to be inefficient; hence the main aim of our research is to construct efficient protocols for specific functions.

When we are interested in integer arithmetic, there are two choices to represent a function: an arithmetic circuit over a prime field $\mathbb{Z}_p$ and a Boolean circuit.

Inputs (and outputs) in the arithmetic circuit are represented as elements in $\mathbb{Z}_p$ (or a ring), while inputs in the Boolean circuit are represented as bits. The input encoding has an influence on the efficiency of computation. Addition and multiplication of shared secrets can be performed efficiently in the arithmetic circuit, whereas not in the Boolean circuit. On the other hand, bit-oriented operations like interval tests, equality tests, and comparisons of shared secrets are easy in the Boolean circuit, whereas they are non-trivial tasks in the arithmetic circuit.

To bridge the gap between arithmetic circuits and Boolean circuits, Damgård *et al.* [11] have proposed the MPC protocol called bit-decomposition in the secret sharing setting (e.g., [3, 16]). Also, Schoenmakers and Tuyls [21] have proposed a similar protocol for MPC [10, 13] based on threshold homomorphic cryptosystems (THC) [12, 14]. In the bit-decomposition protocol, a sharing of an element in the field (or an encryption of an element in the ring in the threshold homomorphic setting) is converted into sharings (encryptions) of bits.

The bit-decomposition protocol is very useful and has many applications because it enables bit-oriented operations to be performed in the arithmetic circuit without performing the entire computation bitwise. For example, when computing a^b by using the techniques in [1, 11], or the Hamming distance between a and b where shared secrets a and b are elements in $\mathbb{Z}_p$, the bit-decomposition protocol is essential because we need the bitwise sharings of the shared secrets. Other important applications are comparisons, interval tests and equality tests of shared secrets. For example, in the comparison protocol, a single shared bit is computed such that it indicates the result of a comparison between two shared secrets. In the Boolean circuit, it is relatively easy to compare two shared secrets because the bits of the secrets are shared. That is, in the comparison protocol based on the Boolean circuit (which we call the bitwise less-than protocol in Section 3.3 as in [11]), we can check the secrets *bit by bit* privately and compare the two shared secrets even without revealing the bit position that determines the comparison result. Therefore, even if inputs are given as sharings of elements in the field, the comparison can be performed easily with the bit-decomposition protocol.

Thus the bit-decomposition protocol is a very powerful tool because changing the representations of shared secrets enables us to gain the benefits of both Boolean circuits and arithmetic circuits. However, the bit-decomposition protocol involves expensive computation in terms of round and communication complexities.

In this paper, we present a simplified bit-decomposition protocol by analyzing the original protocol. Moreover, we construct more efficient protocols for the main applications of the bit-decomposition protocol, which are interval tests, equality tests, and comparisons, without relying on the bit-decomposition protocol though it seemed essential. For example, the equality test protocol is an important subprotocol in [8, 20], so it will be meaningful to construct efficient protocols for these applications without relying on the bit-decomposition protocol if possible. For the equality test, we present deterministic and probabilistic protocols.

In our constructions, the outputs of the protocols are also shared without being revealed, so they can be secret inputs for the subsequent computation.

Therefore, our protocols can be used as building blocks in the more complex computation.

Our Results. We construct constant-round protocols for bit-decomposition, interval test, comparison, and equality test, building on the subprotocols in [11]. The proposed bit-decomposition protocol runs with less communication rounds and less data communication than the original protocol [11]. Therefore, the interval test, comparison and equality test protocols are also improved inevitably by using the proposed bit-decomposition protocol. However, we present new protocols dedicated to them without relying on the bit-decomposition protocol. By using our protocols, given shared secrets as elements in $\mathbb{Z}_p$, we can perform the interval tests, equality tests, and comparisons of the shared secrets more efficiently than the bit-decomposition based protocols. For the equality test, we propose two kinds of protocols. One (Proposed1) is a deterministic protocol and the other (Proposed2) is a probabilistic protocol with a negligible error probability and a much smaller round complexity. The key idea is that we do computation on secret a with c and r where $c = a + r$, c is a revealed value, and r is a random bitwise-shared secret.

In Table 1, we summarize the results of the round and communication (comm.) complexities of each protocol where ℓ is the bit length of prime p of the underlying field for linear secret sharing schemes and k must be chosen such that the error probability $\left(\frac{1}{2}\right)^k$ is negligible. Here "BD-based" means that the protocol is based on the proposed bit-decomposition protocol. As shown in Table 1, we can see that these bit-oritented operations can be realized with smaller complexities than those of the bit-decomposition based protocols by constructing them without the bit-decomposition protocol. For example, the round complexities are reduced by a factor of approximately 3 to 10.

Our protocols (except the probabilistic equality test protocol which is only applicable to the secret sharing setting) are applicable to both the secret sharing setting [11] and the threshold homomorphic setting [21] though we describe our constructions based on the secret sharing setting.

Related Work. Damgård *et al.* [11] have shown a novel technique to convert a polynomial sharing of an element in $\mathbb{Z}_p$ into sharings of bits in constant rounds. Also Shoenmakers and Tuyls [21] have shown a similar conversion technique for multiparty computation based on threshold homomorphic cryptosystems [10, 13]. These protocols are the first to bridge the gap between arithmetic circuits and Boolean circuits.

Toft [24] has proposed another version of a probabilistic equality test protocol independently of and concurrently with our probabilistic equality test protocol. Both the protocols use the property of quadratic residues in a probabilistic way.

Recently, as a practical approach (rather than theoretical constant-round protocols), in [4, 15, 23], the implementation for multiparty integer computation, including the bit-decomposition and comparison, is described with non-constant-round protocols where shared secrets are assumed to be sufficiently small compared with prime p of the underlying secret sharing scheme, whereas we do not assume that shared secrets are upper bounded by a certain value as in [11]. We mention this aspect in Section 7.

Table 1. Comparison of Round / Communication Complexities

Protocol		Round	Comm.
Bit-Decomposition	[11]	38	$93\ell + 94\ell \log_2 \ell$
	Proposed	25	$93\ell + 47\ell \log_2 \ell$
Interval Test	[11]	44	$127\ell + 94\ell \log_2 \ell + 1$
	BD-based	31	$127\ell + 47\ell \log_2 \ell + 1$
	Proposed	13	$110\ell + 1$
Comparison	[11]	44	$205\ell + 188\ell \log_2 \ell$
	BD-based	31	$205\ell + 94\ell \log_2 \ell$
	Proposed	15	$279\ell + 5$
Equality Test	[11]	39	$98\ell + 94\ell \log_2 \ell$
	BD-based	26	$98\ell + 47\ell \log_2 \ell$
	Proposed1	8	81ℓ
	Proposed2	4	$12k$

2 Preliminaries

We assume that n parties $P_1, \ldots, P_n$ are mutually connected by secure and authenticated channels in a synchronous network and the index i for each P_i is public among the parties. Let p be an odd prime and ℓ be the bit length of p. $\mathbb{Z}_p$ is a prime field. When we write $a \in \mathbb{Z}_p$, it means that $a \in \{0, 1, \ldots, p-1\}$. We use $[a]_p$ to denote a polynomial sharing [22] of secret $a \in \mathbb{Z}_p$. That is, $[a]_p$ means that a is shared among the parties where f_a is a random polynomial $f_a(x) = a + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod p$ with randomly chosen $a_i \in \mathbb{Z}_p$ for $1 \leq i \leq t$, $t < \frac{n}{2}$, and $f_a(i)$ is the P_i's share of a. An adversary can corrupt up to t parties. We describe our protocols in the so-called "honest-but-curious" model, but standard techniques will be applicable to make our protocols robust.

Let C be a Boolean test. When we write $[C]_p$, it means that $C \in \{0, 1\}$ and $C = 1$ iff C is true. For example, we use $[a < b]_p$ to denote the output of the comparison protocol.

Because the multiplication protocol is a dominant factor of the complexity, as in [11], we measure the round complexity of a protocol by the number of rounds of parallel invocations of the multiplication protocol [16] and we also measure the communication complexity by the number of invocations of the multiplication protocol. The round complexity relates to the time required for a protocol to be completed and the communication complexity relates to the amount of data communicated among the parties during a protocol run. Though our measurement of complexities basically follows that of [11], the complexity analysis in [11] is rough. In this paper, we reevaluate the round and communication complexities of the protocols in [11] to compare our protocols with those of [11] based on the same measurement.

3 Building Blocks

3.1 Distributed Computation with Shared Secrets for Addition and Multiplication

Let's assume now that n parties have two shared secrets a and b as $[a]_p = \{f_a(1), \ldots, f_a(n)\}$ and $[b]_p = \{f_b(1), \ldots, f_b(n)\}$. Then the parties can obtain $[c+a \bmod p]_p, [ca \bmod p]_p$, and $[a+b \bmod p]_p$ easily where c is a public constant as follows: To compute $[c+a \bmod p]_p, [ca \bmod p]_p$, and $[a+b \bmod p]_p$, each P_i has only to locally compute $c + f_a(i) \bmod p, cf_a(i) \bmod p$, and $f_a(i) + f_b(i) \bmod p$ respectively. Therefore, these can be done efficiently without communication among n parties. When we write $[c+a]_p = c + [a]_p, [ca]_p = c[a]_p$, and $[a+b]_p = [a]_p + [b]_p$, these mean that the parties perform these operations. We also use $\sum$, for example, like $\sum_{i=1}^{3}[a_i]_p$ to denote $[a_1]_p + [a_2]_p + [a_3]_p$.

Multiplication to obtain $[ab \bmod p]_p$ is a bit more complex and it requires the parties to communicate with each other. We assume that the parties perform the multiplication protocol in [16]. When we write $[ab]_p = [a]_p \times [b]_p$, it means that the parties perform the multiplication protocol to compute $[ab \bmod p]_p$.

We will evaluate the round complexity of a protocol by performing the multiplication protocol in parallel as much as possible.

3.2 Bitwise Sharing

The concept of bitwise sharing is to share $a \in \mathbb{Z}_p(= \{0, 1, \ldots, p-1\})$ in the form of $\{[a_{\ell-1}]_p, \ldots, [a_0]_p\}$ such that $a = \sum_{i=0}^{\ell-1} 2^i a_i$ where $a_i \in \{0, 1\}$. We use $[a]_B$ to denote $\{[a_{\ell-1}]_p, \ldots, [a_1]_p, [a_0]_p\}$.

3.3 Subprotocols

We describe several subprotocols in [2, 11] necessary for our constructions. All these subprotocols run in a constant number of rounds. By combining these subprotocols, we will construct our interval test, equality test, comparison, and bit-decomposition protocols that also run in a constant number of rounds.

Joint Random Number Sharing. The parties can share a uniformly random, unknown number r [2] as follows: Each P_i picks up $r_{\_i} \in \mathbb{Z}_p$ at random and shares it by a sharing $[r_{\_i}]_p = \{f_i(1), \ldots, f_i(n)\}$ where $f_i(0) = r_{\_i}$ and f_i is a random polynomial. That is, P_i distributes $f_i(j)$'s to other P_j's. From each $[r_{\_i}]_p$, the parties compute $[r]_p = \sum_{i=1}^{n}[r_{\_i}]_p$. We assume that the complexity for this is almost the same as the complexity of 1 invocation of the multiplication protocol. We denote this subprotocol as $[r \in_R \mathbb{Z}_p]_p$.

Joint Random Bit Sharing. The parties can share a uniformly random $a \in \{0, 1\}$ as follows: The parties compute $[r \in_R \mathbb{Z}_p]_p$, perform the multiplication protocol to obtain $[r^2]_p$ and reveal r^2. If $r^2 = 0$, the parties retry. If $r^2 \neq 0$, the parties compute $r' = \sqrt{r^2}$ such that $0 < r' < \frac{p}{2}$. This can be done in polynomial time because p is an odd prime. Then the parties set $[a]_p = 2^{-1}(r'^{-1}[r]_p + 1)$. It is clear that $r'^{-1}r \in \{-1, 1\}$; hence $a \in \{0, 1\}$. The total complexity is 2 rounds and 2 invocations. We denote this subprotocol as $[a \in_R \{0,1\}]_p$. In the setting [10, 13], this can be computed as $a = \oplus_{i=1}^{n} b_i$ where $b_i \in_R \{0,1\}$ is generated by P_i (see [21] for the details).

Unbounded Fan-In Or. Given $[a_{\ell-1}]_p, \ldots, [a_0]_p$ where $a_i \in \{0,1\}$, the parties can compute $[\vee_{i=0}^{\ell-1} a_i]_p$ in a constant number of rounds. For this, as in [11], we can use the same technique to evaluate symmetric Boolean functions as follows:

The parties compute $[A]_p = 1 + \sum_{i=0}^{\ell-1}[a_i]_p$. Note that $1 \leq A \leq \ell + 1$. Next, the parties define a ℓ-degree polynomial $f_\ell(x)$ such that $f_\ell(1) = 0$ and $f_\ell(2) = f_\ell(3) = \cdots = f_\ell(\ell+1) = 1$. $f_\ell(x)$ can be determined by using Lagrange interpolation. Note that $f_\ell(A) = \vee_{i=0}^{\ell-1} a_i$. Then the parties try to obtain $[\vee_{i=0}^{\ell-1} a_i]_p$ by computing $[f_\ell(A)]_p$ from $[A]_p$ and $f_\ell(x)$. This can be done in a constant number of rounds by using an unbounded fan-in multiplication and the inversion protocol [2] as follows:

Let's assume that $f_\ell(x)$ is represented as $f_\ell(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_\ell x^\ell \bmod p$. To obtain $[f_\ell(A)]_p$, the parties compute $[A]_p, [A^2]_p, \ldots, [A^\ell]_p$ because $[f_\ell(A)]_p = \alpha_0 + \sum_{i=1}^{\ell} \alpha_i [A^i]_p$.

For $1 \leq i \leq \ell$, the parties generate $[b_i \in_R \mathbb{Z}_p]_p$ and $[b'_i \in_R \mathbb{Z}_p]_p$ in parallel, compute $[B_i]_p = [b_i]_p \times [b'_i]_p$, and reveal B_i. Note that $[b_i^{-1}]_p$ can be computed as $[b_i^{-1}]_p = B_i^{-1}[b'_i]_p$ at the same time (inversion protocol).

Next, the parties compute in parallel

$$\begin{aligned}
[c_1]_p &= [A]_p \times [b_1^{-1}]_p \\
[c_2]_p &= [A]_p \times [b_1]_p \times [b_2^{-1}]_p \\
&\vdots \\
[c_{\ell-1}]_p &= [A]_p \times [b_{\ell-2}]_p \times [b_{\ell-1}^{-1}]_p \\
[c_\ell]_p &= [A]_p \times [b_{\ell-1}]_p \times [b_\ell^{-1}]_p
\end{aligned}$$

and reveal all c_i's.

Then the parties can compute $[A^i]_p = (\prod_{k=1}^{i} c_k)[b_i]_p$.

If $A = 0$, information about A is leaked. That is why we used $[A]_p = 1 + \sum_{i=0}^{\ell-1}[a_i]_p$ to guarantee that A is not zero.

The complexity of computing each component is as follows: 2 rounds and 3ℓ invocations for $[b_i]_p$'s, $[b'_i]_p$'s, and B_i's and 2 rounds and 2ℓ invocations for c_i's. $[b_i]_p \times [b_{i+1}^{-1}]_p$ for $1 \leq i \leq \ell - 1$ can be precomputed as $[b_i]_p \times [b'_{i+1}]_p$ in the second round in parallel with $[b_i]_p \times [b'_i]_p$. Therefore, the total complexity is 3 rounds (including 2 rounds for random value generation) and 5ℓ invocations.

Note that we can compute unbounded fan-in And and Xor similarly because a symmetric Boolean function depends only on the number of 1's in its inputs. Also note that the random values necessary for this protocol can be generated in advance rather than on demand when this subprotocol is used as a building block in the larger protocol, thus reducing the round complexity. Actually all the random value generation (for bits and numbers) can be done in the first 2 rounds (3 rounds in the setting [21] by using an unbounded fan-in Xor).

Prefix-Or. Given $[a_1]_p, \ldots, [a_\ell]_p$ where $a_i \in \{0,1\}$, the parties can compute the Prefix-Or $[b_1]_p, \ldots, [b_\ell]_p$ such that $b_i = \vee_{j=1}^{i} a_j$ in a constand number of rounds. As in [11], this can be done by using the technique from [5] as follows:

For notational convenience, let's assume that $\ell = \lambda^2$ for an integer λ and index the bits a_k as $a_{i,j} = a_{\lambda(i-1)+j}$ for $i, j = 1, \ldots, \lambda$. Other cases can be adapted quite straightforwardly.

First the parties compute $[x_i]_p = \vee_{j=1}^{\lambda} [a_{i,j}]_p$ for $i = 1, \ldots, \lambda$ in parallel by using unbounded fan-in Or where the size of problems is λ instead of ℓ. Then the parties compute similarly $[y_i]_p = \vee_{k=1}^{i} [x_k]_p$ for $i = 1, \ldots, \lambda$ in parallel. Now we can notice that $y_i = 1$ iff some block $\{a_{i',1}, \ldots, a_{i',\lambda}\}$ with $i' \le i$ contains a $a_{i',j} = 1$.

Next, the parties set $[f_1]_p = [x_1]_p$, and for $i = 2, \ldots, \lambda$, set $[f_i]_p = [y_i]_p - [y_{i-1}]_p$. Now we can notice that $f_i = 1$ iff $\{a_{i,1}, \ldots, a_{i,\lambda}\}$ is the first block containing a $a_{i,j} = 1$. Let i_0 be such that $f_{i_0} = 1$. The parties can compute $\{[a_{i_0,1}]_p, \ldots, [a_{i_0,\lambda}]_p\}$ by $[a_{i_0,j}]_p = \sum_{i=1}^{\lambda} [f_i]_p \times [a_{i,j}]_p$ in parallel without revealing i_0.

Next, the parties compute $\{[b_{i_0,1}]_p, \ldots, [b_{i_0,\lambda}]_p\}$ where $b_{i_0,j} = \vee_{k=1}^{j} a_{i_0,k}$ by using unbounded fan-in Or in parallel.

Finally, the parties set $[s_i]_p = [y_i]_p - [f_i]_p$. Then $s_i = 1$ iff $i > i_0$. If we index the bits of Prefix-Or b_k as $b_{i,j} = b_{\lambda(i-1)+j}$ as we did for a_k, the Prefix-Or can be computed as $[b_k]_p = [b_{\lambda(i-1)+j}]_p = [b_{i,j}]_p = [f_i]_p \times [b_{i_0,j}]_p + [s_i]_p$ in the end.

When we use several invocations of unbounded fan-in Or, all the necessary random values in unbounded fan-in Or can be generated in the first 2 rounds. Therefore, the total complexity is 7 rounds (including 2 rounds for random value generation) and 17ℓ invocations. [3] Similarly the Prefix-And can also be computed by using the same technique.

Bitwise Less-Than. Given two bitwise sharings $[a]_B$ and $[b]_B$, the parties can compute $[a < b]_p$ without revealing $(a < b)$ itself. The basic idea is the same as the circuit for the millionaire's problem. We will give an outline of this subprotocol based on the description in [11].

For $0 \le i \le \ell - 1$, the parties compute $[c_i]_p = [a_i \oplus b_i]_p = [a_i] + [b_i]_p - 2[a_i b_i]_p$ in parallel and then compute $[d_i]_p = \vee_{j=i}^{\ell-1} [c_j]_p$ by using Prefix-Or, and set $[e_i]_p = [d_i - d_{i+1}]_p$ where $[e_{\ell-1}]_p = [d_{\ell-1}]_p$. Finally, the parties compute $[a < b]_p = \sum_{i=0}^{\ell-1} ([e_i]_p \times [b_i]_p)$ in parallel.

The complexity of computing each component is as follows: 1 round and ℓ invocations for c_i's, 7 rounds and 17ℓ invocations for the Prefix-Or, and 1 round and ℓ invocations for $\sum_{i=0}^{\ell-1} ([e_i]_p \times [b_i]_p)$. Because c_i's can be computed in parallel with random value generation in the Prefix-Or, the total complexity is 8 rounds (including 2 rounds for random value generation) and 19ℓ invocations. We use $[a <_B b]_p$ in order to stress that a and b are bitwise-shared. Note that if b is known, the complexity is 7 rounds (including 2 rounds for random value generation) and 17ℓ invocations by saving the invocations for c_i's and $\sum_{i=0}^{\ell-1} ([e_i]_p \times [b_i]_p)$.

Joint Random Number Bitwise-Sharing. The parties can bitwise-share a

[3] The evaluation in [11] is 17 rounds and 20ℓ invocations by generating random values on demand.

uniformly random, unknown number r such that $0 \leq r = \sum_{i=0}^{\ell-1} 2^i r_i < p$ as follows: The parties generate each bit, $[r_i \in_R \{0,1\}]_p$ for $0 \leq i \leq \ell - 1$ in parallel, compute $[r <_B p]_p$ by using the bitwise less-than protocol and reveal $(r < p)$. If $r \geq p$, the parties retry.

The complexity of computing each component is as follows: 2 rounds and 2ℓ invocations for r_i's and 7 rounds and 17ℓ invocations for the bitwise less-than protocol (note that p is known). Because r_i's can be generated in parallel with random value generation in the Prefix-Or of the bitwise less-than protocol, the complexity is 7 rounds and 19ℓ invocations. As in [11], we assume that at least one of four generated candidates is less than p and the amortized complexity is 7 rounds (including 2 rounds for random value generation) and 76ℓ invocations. We denote this subprotocol as $[r \in_R \mathbb{Z}_p]_B$.

Bitwise Sum. Given two bitwise sharings $[a]_B = \{[a_{\ell-1}]_p, \ldots, [a_0]_p\}$ and $[b]_B = \{[b_{\ell-1}]_p, \ldots, [b_0]_p\}$, the parties can compute the bitwise sharing $[d]_B = \{[d_\ell]_p, \ldots, [d_0]_p\}$ such that $d = a + b$ over the integers (not mod p). By using the method of [6], the bitwise sum protocol can be performed in constant rounds (see [11] for the details). The total complexity based on the complexity analysis in this paper is 15 rounds (including 2 rounds for random value generation) and $47\ell \log_2 \ell$ invocations. [4] See Appendix A for the details. We denote this subprotocol as $[d]_B = [a]_B + [b]_B$.

4 Existing Protocols [11, 21]

Damgård *et al.* [11] have shown a novel technique to convert $[a]_p$ into $[a]_B$. This technique is called the bit-decomposition protocol (Fig. 1). Note that we can obtain $[a]_p$ from $[a]_B$ easily by computing $[a]_p = \sum_{i=0}^{\ell-1} 2^i [a_i]_p \bmod p$. Also, Schoenmakers and Tuyls [21] have proposed a similar bit-decomposition protocol (called **BITREP** gate) in the context of multiparty computation [10, 13] based on threshold additively-homomorphic cryptosystems.

The complexity of computing each component in [11] is as follows: 7 rounds (including 2 rounds for random value generation) and 76ℓ invocations for $[r \in_R \mathbb{Z}_p]_B$, 13 rounds and $47\ell \log_2 \ell$ invocations for $[d]_B$ (bitwise sum), 5 rounds and 17ℓ invocations for $[q]_p$, that is, $[d <_B p]_p$, and 13 rounds and $47\ell \log_2 \ell$ invocations for $[h]_B$ (bitwise sum). The total complexity is 38 rounds (including 2 rounds for random value generation) and $93\ell + 94\ell \log_2 \ell$ invocations.

By using the bit-decomposition protocol, any bit-oriented operation can be performed in arithmetic circuits where inputs are given as polynomial sharings (rather than bitwise sharings) of elements in $\mathbb{Z}_p$.

However, the bit-decomposition protocol is not cheap, so we try to construct a simplified bit-decomposition protocol and construct more efficient protocols for interval tests, equality tests, and comparisons without relying on the bit-decomposition protocol.

[4] The evaluation in [11] is 37 rounds and $55\ell \log_2 \ell$ invocations by generating random values on demand.

The parties convert $[a]_p$ into $[a]_B$.

1. The parties generate $[r]_B$ and obtain $[r]_p$ eventually.
2. The parties compute $[c]_p = [a]_p - [r]_p$ and reveal $c = a - r \bmod p \in \{0, 1, \ldots, p-1\}$.
3. The parties compute $[d]_B = [r]_B + [c]_B = \{[d_\ell]_p, \ldots, [d_0]_p\}$.
4. Note that d can be represented as $d = a + qp$ where $q \in \{0, 1\}$. The parties can compute the bit q as $[q]_p = [p \leq d]_p = 1 - [d <_B p]_p$.
5. Consider $g = (2^\ell - qp) \bmod 2^\ell$ and its bitwise sharing $[g]_B = \{[g_{\ell-1}]_p, \ldots, [g_0]_p\}$. Let $(f_{\ell-1}, \ldots, f_0)_2$ be the bit representation of $2^\ell - p$ such that $2^\ell - p = \sum_{i=0}^{\ell-1} 2^i f_i$ and $f_i \in \{0, 1\}$. Then the parties can compute $[g]_B$ by $[g_i]_p = f_i[q]_p$ for $0 \leq i \leq \ell-1$ because $g = 0$ if $q = 0$ and $g = 2^\ell - p$ if $q = 1$.
6. The parties now have the two following bitwise sharings, $[d]_B = [a + qp]_B$ and $[g]_B = [(2^\ell - qp) \bmod 2^\ell]_B$. Therefore, the parties can compute $[h]_B = [d]_B + [g]_B$ where $h = a + q2^\ell$.
7. By discarding the sharing $[h_\ell]_p$ from $[h]_B$, they can obtain $[a]_B$.

Fig. 1. Bit-Decomposition [11]

5 Simplified Bit-Decomposition Protocol

In the original bit-decomposition protocol, we need 2 invocations of the bitwise sum protocol (in Steps 3 and 6 in Fig. 1). We can notice that the first invocation for $[d]_B$ can be eliminated by changing the way in which we compute $[q]_p$ based on the following observation.

In Step 4 of the original protocol, the parties compute $[q]_p = 1 - [d <_B p]_p$ where $d = r + c$, c is public, and r is bitwise-shared. Therefore, the condition, $(d < p)$ can be changed into $(r < p - c)$. The parties have $[r]_B$ and $p - c$ is public, so $(r < p - c)$ can be computed by using the bitwise less-than protocol without computing $[d]_B = [r]_B + [c]_B$, thus eliminating one invocation of the bitwise sum protocol.

Since we have eliminated $[d]_B$, we need to specify how to compute $[a]_B$ in the rest of the protocol. Fortunately, we can use $[r]_B$ itself to compute $[a]_B$ by using the bitwise sum protocol. The simplified bit-decomposition protocol is given in Fig. 2.

Complexity of Bit-Decomposition Protocol. The complexity of computing each component is as follows: 7 rounds (including 2 rounds for random value generation) and 76ℓ invocations for $[r \in_R \mathbb{Z}_p]_B$, 5 rounds and 17ℓ invocations for $[q]_p$, that is, $[r <_B p - c]_p$, and 13 rounds and $47\ell \log_2 \ell$ invocations for $[h]_B$. The total complexity is 25 rounds (including 2 rounds for random value generation) and $93\ell + 47\ell \log_2 \ell$ invocations.

The parties convert $[a]_p$ into $[a]_B$.

1. The parties generate $[r]_B$ and obtain $[r]_p$ eventually.
2. The parties compute $[c]_p = [a]_p - [r]_p$ and reveal $c = a - r \bmod p \in \{0, 1, \ldots, p-1\}$. If $c = 0$, the parties are successfully done because $[r]_B$ is equal to $[a]_B$ by a coincidence.
3. If $c \neq 0$, next, the parties compute the bit q, $[q]_p = [p \leq r + c]_p = 1 - [r <_B p - c]_p$ by using the bitwise less-than protocol.
4. Note that a can be represented as $a = c + r - qp$ over the integers where $q \in \{0, 1\}$. Therefore, we also have $2^\ell + a = 2^\ell + c - qp + r$ over the integers. Consider $\widetilde{g} = (2^\ell + c - qp) \bmod 2^\ell$ and its bitwise sharing $[\widetilde{g}]_B = \{[\widetilde{g}_{\ell-1}]_p, \ldots, [\widetilde{g}_0]_p\}$. Let $(\widetilde{f}_{\ell-1}, \ldots, \widetilde{f}_0)_2$ be the bit representation of $2^\ell + c - p$ such that $2^\ell + c - p = \sum_{i=0}^{\ell-1} 2^i \widetilde{f}_i$ and $\widetilde{f}_i \in \{0, 1\}$. Also, let $(\widetilde{f}'_{\ell-1}, \ldots, \widetilde{f}'_0)_2$ be the bit representation of c such that $c = \sum_{i=0}^{\ell-1} 2^i \widetilde{f}'_i$ and $\widetilde{f}'_i \in \{0, 1\}$. Then the parties can compute $[\widetilde{g}]_B$ by $[\widetilde{g}_i]_p = (\widetilde{f}_i - \widetilde{f}'_i)[q]_p + \widetilde{f}'_i$ for $0 \leq i \leq \ell - 1$ because $\widetilde{g} = c$ if $q = 0$ and $\widetilde{g} = 2^\ell + c - p$ if $q = 1$.
5. The parties now have the two following bitwise sharings, $[r]_B$ and $[\widetilde{g}]_B = [(2^\ell + c - qp) \bmod 2^\ell]_B$. Therefore, the parties can compute $[h]_B = [r]_B + [\widetilde{g}]_B$ where $h = a + q2^\ell$.
6. By discarding the sharing $[h_\ell]_p$ from $[h]_B$, they can obtain $[a]_B$.

Fig. 2. Simplified Bit-Decomposition

6 Proposed Protocols Without Bit-Decomposition

6.1 Interval Test Protocol

In the interval test protocol, given public constants $c_1, c_2 \in \mathbb{Z}_p$ (where $c_1 < c_2$) and shared secret $a \in \mathbb{Z}_p$, the parties compute $[c_1 < a < c_2]_p$ without revealing $(c_1 < a < c_2)$ itself.

If the parties use the bit-decomposition protocol, the parties compute $[a]_B$ from $[a]_p$ and compute $[c_1 < a < c_2]_p = [c_1 <_B a]_p \times [a <_B c_2]_p$.

The basic idea of our construction is as follows: We randomize a by $c = a + r$ and reveal c where r is a bitwise-shared random secret. We obtain an appropriate interval $[r_{\text{low}}, r_{\text{high}}]$ from c, c_1, and c_2. Then computing $[c_1 < a < c_2]_p$ is reduced to checking whether r exists in the appropriate interval $r_{\text{low}} < r < r_{\text{high}}$ (for example, see Fig. 3) by the bitwise less-than protocol.

Procedure. The parties generate $[r \in_R \mathbb{Z}_p]_B$ and obtain $[r]_p$ eventually. Next, the parties compute $[c]_p = [a]_p + [r]_p$ and reveal $c = a + r \bmod p \in \{0, 1, \ldots, p-1\}$. At this point, no information about a is leaked from c because r is uniformly random and unknown to the parties. Now we can think that $a \in \{-(p - c - 1), \ldots, -1, 0, 1, \ldots, c - 1, c\}$ because $r \in \{0, 1, \ldots, p - 1\}$.

First, we consider the case where $c_1 < c < c_2$ does not hold. When $c_2 \leq c$ (see Fig. 3), obviously, we have $(c_1 < a < c_2) = 1$ if $(r_{\text{low}} =) c - c_2 < r < c - c_1 (= r_{\text{high}})$. Similarly, when $c \leq c_1$ (see Fig. 4), if $(r_{\text{low}} =) c + p - c_2 < r < c + p - c_1 (= r_{\text{high}})$, we have $-(p - c_1) < a < -(p - c_2)$. This means that $(c_1 < a < c_2) = 1$. Therefore, the parties compute, by using the bitwise less-than protocol,

$$[c_1 < a < c_2]_p = [r_{\text{low}} <_B r]_p \times [r <_B r_{\text{high}}]_p.$$

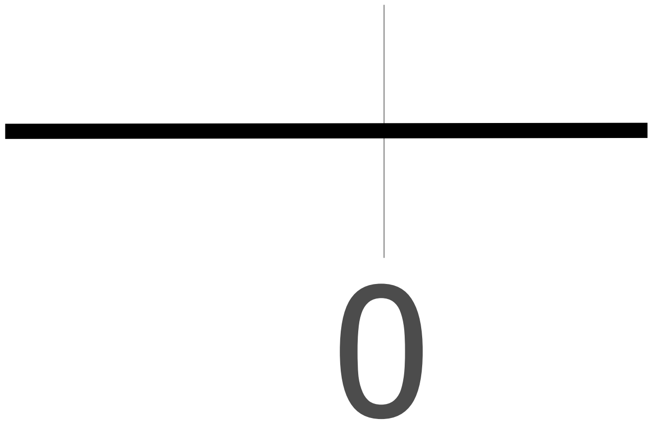

Fig. 3. Case of $c_2 \leq c$

Next, we consider the case where $c_1 < c < c_2$ holds (see Fig. 5). In this case, if $(r_{\text{low}} =)c - c_1 - 1 < r < c + p - c_2 + 1(= r_{\text{high}})$, we have $-(p - c_2) \leq a \leq c_1$. This means that $(c_1 < a < c_2) = 0$. Therefore, the parties compute

$$[r_{\text{low}} < r < r_{\text{high}}]_p = [c - c_1 - 1 <_B r]_p \times [r <_B c + p - c_2 + 1]_p$$

by using the bitwise less-than protocol and set

$$[c_1 < a < c_2]_p = 1 - [r_{\text{low}} < r < r_{\text{high}}]_p.$$

Complexity of Interval Test Protocol. If we use the bit-decomposition protocol straightforwardly, the complexity of computing each component is as follows: 38 rounds (including 2 rounds for random value generation) and $93\ell + 94\ell \log_2 \ell$ invocations for $[a]_B$, 5 rounds and $(17\ell \times 2)$ invocations for $[c_1 <_B a]_p$ and $[a <_B c_2]_p$, and 1 round and 1 invocation for $[c_1 <_B a]_p \times [a <_B c_2]_p$. The total complexity is 44 rounds (including 2 rounds for random value generation) and $127\ell + 94\ell \log_2 \ell + 1$ invocations.

On the other hand, in our construction, the complexity of computing each component is as follows: 7 rounds (including 2 rounds for random value generation) and 76ℓ invocations for $[r \in_R \mathbb{Z}_p]_B$, 5 rounds and $(17\ell \times 2)$ invocations for $[r_{\text{low}} <_B r]_p$ and $[r <_B r_{\text{high}}]_p$, and 1 round and 1 invocation for $[r_{\text{low}} <_B r]_p \times [r <_B r_{\text{high}}]_p$. The total complexity is 13 rounds (including 2 rounds for random value generation) and $110\ell + 1$ invocations.

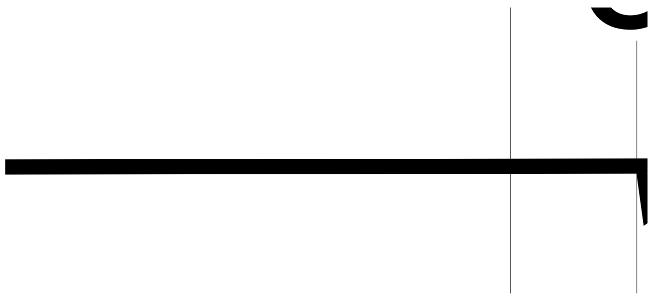

Fig. 4. Case of $c \leq c_1$

6.2 LSB Protocol for Special Case of Interval Test Protocol

In order to construct our comparison protocol later, we consider computing $[a < \frac{p}{2}]_p$. Though it is possible for us to use the technique in Section 6.1, we compute $[a < \frac{p}{2}]_p$ more efficiently by using special properties of $\frac{p}{2}$ and apply this subprotocol (called the LSB protocol here) to our comparison protocol. By a simple observation, we can notice that $a \in \{0, 1, \ldots, \frac{p-1}{2}\} \Leftrightarrow (2a \bmod p)_0 = 0$, and that $a \in \{\frac{p-1}{2} + 1, \ldots, p-1\} \Leftrightarrow (2a \bmod p)_0 = 1$ where $(x)_0$ is the least significant bit (LSB) of $x \in \{0, 1, \ldots, p-1\}$. That is, if $a < \frac{p}{2}$, *no* wrap-around modulo p occurs when $2a \bmod p$ is computed and $2a \bmod p$ is even. On the other hand, if $a > \frac{p}{2}$, a wrap-around modulo p occurs when $2a \bmod p$ is computed and $2a \bmod p$ is odd. Therefore, if we can compute $[(x)_0]_p$ from $[x]_p$, we can use it to compute $[a < \frac{p}{2}]_p$.

To compute $[(x)_0]_p$ from $[x]_p$, we randomize x by $c = x+r$ and reveal c where r is a bitwise-shared random secret. Then we can obtain $[(x)_0]_p$ from $(c)_0$ and $[(r)_0]_p$.

Procedure. The parties want to compute $[(x)_0]_p$ from $[x]_p$. The parties generate $[r \in_R \mathbb{Z}_p]_B$ and obtain $[r]_p$ eventually. Next, the parties compute $[c]_p = [x]_p + [r]_p$ and reveal $c = x+r \bmod p \in \{0, 1, \ldots, p-1\}$. If *no* wrap-around modulo p occurs when c is computed, we have $(x)_0 = (c)_0 \oplus (r)_0$ and if a wrap-around modulo p occurs when c is computed, we have $(x)_0 = 1 - \{(c)_0 \oplus (r)_0\}$. Furthermore, we

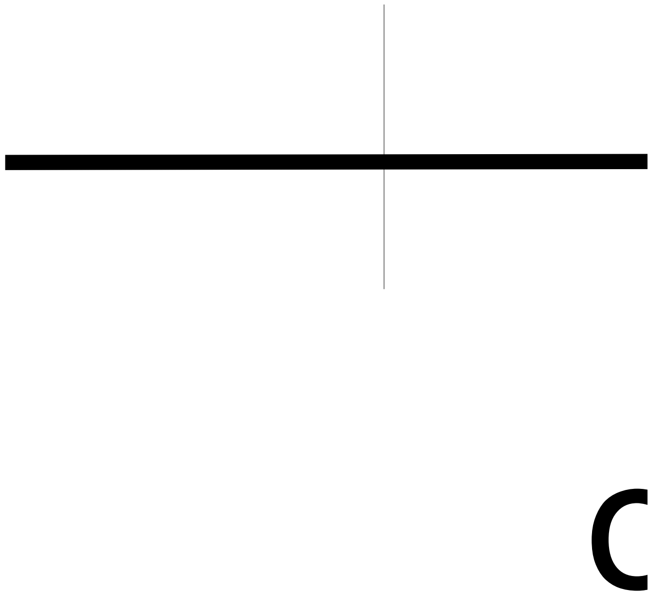

Fig. 5. Case of $c_1 < c < c_2$

can use $(c < r)$ to know whether or not a wrap-around modulo p occurred when c was computed. That is, if $(c < r) = 0$, it means that no wrap-around modulo p occurred, and if $(c < r) = 1$, it means that a wrap-around modulo p occurred because $r \in \{0, 1, \ldots, p-1\}$.

From these facts, the parties can compute $[(x)_0]_p$ as

$$\begin{aligned} [(x)_0]_p &= [c <_B r]_p \times (1 - \{(c)_0 \oplus [(r)_0]_p\}) + (1 - [c <_B r]_p) \times \{(c)_0 \oplus [(r)_0]_p\} \\ &= [c <_B r]_p + \{(c)_0 \oplus [(r)_0]_p\} - 2[c <_B r]_p \times \{(c)_0 \oplus [(r)_0]_p\}. \qquad (1) \end{aligned}$$

The interpretation of Eq. (1) is that if $(c <_B r) = 1$, we have $(1 - \{(c)_0 \oplus [(r)_0]_p\})$ and otherwise we have $\{(c)_0 \oplus [(r)_0]_p\}$. Because c is public, note that $(c)_0 \oplus [(r)_0]_p$ can be computed as

$$(c)_0 \oplus [(r)_0]_p = \begin{cases} [(r)_0]_p & \text{if } (c)_0 = 0 \\ 1 - [(r)_0]_p & \text{if } (c)_0 = 1. \end{cases}$$

Also note that the parties already have $[(r)_0]_p$ because r is generated by $[r \in_R \mathbb{Z}_p]_B$.

By using the LSB protocol, the parties can compute $[a < \frac{p}{2}]_p$ from $[a]_p$ as $[a < \frac{p}{2}]_p = 1 - [(2a)_0]_p$.

Complexity of LSB Protocol. The complexity of computing each component is as follows: 7 rounds (including 2 rounds for random value generation) and 76ℓ

Table 2. Truth Table for $(a < b)$

$w = (a < p/2)$	$x = (b < p/2)$	$y = (a - b \bmod p < p/2)$	$z = (a < b)$
1	0	*	1
0	1	*	0
0	0	0	1
0	0	1	0
1	1	0	1
1	1	1	0

invocations for $[r \in_R \mathbb{Z}_p]_B$, 5 rounds and 17ℓ invocations for $[c <_B r]_p$, and 1 round and 1 invocation for $[c <_B r]_p \times [(r)_0]_p$. The total complexity is 13 rounds (including 2 rounds for random value generation) and $93\ell + 1$ invocations.

6.3 Comparison Protocol

In the comparison protocol, given two shared secrets $a, b \in \mathbb{Z}_p$, the parties compute $[a < b]_p$ without revealing $(a < b)$ itself. For example, we can compute $[\max(a, b)]_p = [a]_p + [a < b]_p \times [b - a]_p$ by using the comparison protocol.

If the parties use the bit-decomposition protocol, the parties compute $[a]_B$ and $[b]_B$ from $[a]_p$ and $[b]_p$ and compute $[a <_B b]_p$ as in [11].

It seems difficult for us to compare a and b directly without using the bit-decomposition protocol. Therefore, we compare a and b indirectly via the value of $\frac{p}{2}$ by computing $[a < \frac{p}{2}]_p$, $[b < \frac{p}{2}]_p$, and $[a - b \bmod p < \frac{p}{2}]_p$.

Procedure. By a simple observation, we can notice that $(a < b)$ is determined from $(a < \frac{p}{2})$, $(b < \frac{p}{2})$, and $(a - b \bmod p < \frac{p}{2})$. This observation can be confirmed by the truth table (Table 2).

When we denote $(a < \frac{p}{2})$, $(b < \frac{p}{2})$, $(a - b \bmod p < \frac{p}{2})$, and $(a < b)$ as w, x, y, and z respectively, then z is represented as

$$\begin{aligned} z &= w\bar{x} \vee \bar{w}\bar{x}\bar{y} \vee wx\bar{y} \\ &= w(1 - x) + (1 - w)(1 - x)(1 - y) + wx(1 - y) \\ &= w(x + y - 2xy) + 1 - y - x + xy. \end{aligned} \tag{2}$$

Therefore, if the parties can compute $[a < \frac{p}{2}]_p$, $[b < \frac{p}{2}]_p$, and $[a - b \bmod p < \frac{p}{2}]_p$, they can compute $[a < b]_p$ from Eq. (2) by using addition and the multiplication protocol. We can use the LSB protocol to compute all three of these values.

Complexity of Comparison Protocol. If we use the bit-decomposition protocol straightforwardly, the complexity of computing each component is as follows: 38 rounds (including 2 rounds for random value generation) and $2 \times (93\ell + 94\ell \log_2 \ell)$ invocations for $[a]_B$ and $[b]_B$ and 6 rounds and 19ℓ invocations for $[a <_B b]_p$. The total complexity is 44 rounds (including 2 rounds for random value generation) and $205\ell + 188\ell \log_2 \ell$ invocations.

On the other hand, in our construction, the complexity of computing each component is as follows: 13 rounds (including 2 rounds for random value generation) and $3 \times (93\ell + 1)$ invocations for $[a < \frac{p}{2}]_p$, $[b < \frac{p}{2}]_p$, and $[a - b \bmod p < \frac{p}{2}]_p$ and 2 rounds and 2 invocations for Eq. (2). The total complexity is 15 rounds (including 2 rounds for random value generation) and $279\ell + 5$ invocations.

6.4 Equality Test Protocol

In the equality test protocol, given two shared secrets $a, b \in \mathbb{Z}_p$, the parties compute $[a = b]_p$ without revealing $(a = b)$ itself.

Because $[a = b]_p$ can be computed by $[a - b = 0]_p$, we focus on computing $[a = 0]_p$.

If the parties use the bit-decomposition protocol, the parties compute $[d]_B$ from $[d]_p = [a - b]_p$ and compute $[\wedge_{i=0}^{\ell-1}(1 - d_i)]_p$ by using an unbounded fan-in And as in [11].

In our construction, we use a very simple observation that the randomization $c(= d + r)$ of d is equal to r if d is zero.

Procedure. First the parties generate $[r \in_R \mathbb{Z}_p]_B$ and obtain $[r]_p$ eventually. Next, the parties compute $[c]_p = [a]_p + [r]_p$ and reveal $c = a + r \bmod p \in \{0, 1, \ldots, p - 1\}$. We can note that $c = r$ iff $a = 0$. Therefore, the parties compute whether all bits of c are the same as $[r]_B$. Let $(c_{\ell-1}, \ldots, c_0)_2$ be the bit representation of c. Then the parties compute $[c'_i]_p$ for $0 \le i \le \ell - 1$ as

$$[c'_i]_p = \begin{cases} [r_i]_p & \text{if } c_i = 1 \\ 1 - [r_i]_p & \text{if } c_i = 0. \end{cases}$$

We can note that $c'_i \in \{0, 1\}$ and that $c'_i = 1$ iff $c_i = r_i$. Finally, the parties compute $[a = 0]_p$ as $[\wedge_{i=0}^{\ell-1} c'_i]_p$ by using an unbounded fan-in And.

Complexity of Equality Test Protocol. If we use the bit-decomposition protocol straightforwardly, the complexity of computing each component is as follows: 38 rounds (including 2 rounds for random value generation) and $93\ell + 94\ell \log_2 \ell$ invocations for $[d]_B$ and 1 rounds and 5ℓ invocations for $[\wedge_{i=0}^{\ell-1}(1 - d_i)]_p$. The total complexity is 39 rounds (including 2 rounds for random value generation) and $98\ell + 94\ell \log_2 \ell$ invocations.

On the other hand, in our construction, the complexity of computing each component is as follows: 7 rounds (including 2 rounds for random value generation) and 76ℓ invocations for $[r]_B$ and 1 rounds and 5ℓ invocations for $[\wedge_{i=0}^{\ell-1} c'_i]_p$. The total complexity is 8 rounds (including 2 rounds for random value generation) and 81ℓ invocations.

6.5 Probabilistic Equality Test Protocol

We consider another version of the equality test protocol with a very small round complexity. We focus on computing $[a = 0]_p$ again. In our construction, we assume that $p = 3 \bmod 4$ or $p = 5 \bmod 8$. These imply that Legendre symbol

$\left(\frac{-1}{p}\right) = -1$ if $p = 3 \bmod 4$ and that $\left(\frac{2}{p}\right) = -1$ if $p = 5 \bmod 8$. The basic idea is based on the property of quadratic residues as follows: If a is a zero, we always have $\left(\frac{c}{p}\right) = \left(\frac{r}{p}\right)$ where $c = a + r$, r is a random secret and c is a revealed value. If a is not a zero, we have $\left(\frac{c}{p}\right) \neq \left(\frac{r}{p}\right)$ with non-negligible probability. By checking whether $\left(\frac{c}{p}\right) = \left(\frac{r}{p}\right)$ secretly with sufficiently many trials, we can perform the equality test on a in a probabilistic way. Here note that we need to generate random secret r in a special way to compute $\left(\frac{r}{p}\right)$ secretly.

Procedure. First we describe the case of $p = 3 \bmod 4$. The case of $p = 5 \bmod 8$ can be obtained quite straightforwardly as we mention later.

The parties generate $[b_j \in_R \{-1, 1\}]_p$, $[r_j \in_R \mathbb{Z}_p]_p$, and $[r'_j \in_R \mathbb{Z}_p]_p$ for $1 \leq j \leq k$ in parallel where k is chosen such that the error probability $\left(\frac{1}{2}\right)^k$ is negligible. The value b_j can be generated by a joint random bit sharing. Next, the parties compute for $1 \leq j \leq k$ in parallel,

$$[c_j]_p = [a]_p \times [r_j]_p + [b_j]_p \times [r'_j]_p \times [r'_j]_p$$

and reveal all the c_j's. Note that $b_j r'^2_j$ is uniformly random and unknown to the parties, so no information about a is leaked from c_j. Actually we can confirm that $\Pr[b_j r'^2_j = 0] = \Pr[r'_j = 0] = \frac{1}{p}$, that $\Pr[b_j r'^2_j = y] = \Pr[b_j = 1] \times \Pr[r'_j = \pm\sqrt{y}] = \frac{1}{2} \times \frac{2}{p} = \frac{1}{p}$ if y is a quadratic residue , and that $\Pr[b_j r'^2_j = y] = \Pr[b_j = -1] \times \Pr[r'_j = \pm\sqrt{-y}] = \frac{1}{2} \times \frac{2}{p} = \frac{1}{p}$ if y is a quadratic nonresidue.

Also note that if $a = 0$, ar_j is always a zero and that if $a \neq 0$, ar_j is uniformly random.

If c_j is a zero, the parties discard the c_j and retry. The probability that c_j happens to be a zero is $\frac{1}{p}$ and negligible in the practical setting (e.g., $p > 2^{32}$).

Assuming that c_j is not a zero, we can notice that $a = 0 \Rightarrow \left(\frac{c_j}{p}\right) = \left(\frac{b_j r'^2_j}{p}\right) = b_j$ with prob. 1, and that $a \neq 0 \Rightarrow \left(\frac{c_j}{p}\right) = b_j$ with prob. $\frac{1}{2}$. The case of $a = 0$ is obvious. When $a \neq 0$, c_j is uniformly random whether b_j is -1 or 1 because ar_j is uniformly random, so the probability that $\left(\frac{c_j}{p}\right) = b_j$ is $\frac{1}{2}$.

Then the parties compute for $1 \leq j \leq k$,

$$[x_j]_p = \begin{cases} 2^{-1}([b_j]_p + 1) & \text{if } \left(\frac{c_j}{p}\right) = 1 \\ -2^{-1}([b_j]_p - 1) & \text{if } \left(\frac{c_j}{p}\right) = -1. \end{cases}$$

Note that $x_j \in \{0, 1\}$ and that $x_j = 1$ iff $\left(\frac{c_j}{p}\right) = b_j$. Finally, the parties compute $[a = 0]_p = [\wedge_{j=1}^k x_j]_p$ by using an unbounded fan-in And, assuming that at least one of x_j's is 0 if $a \neq 0$ with sufficiently large k.

The error probability that $(a = 0) = 1$ when $a \neq 0$ is $\left(\frac{1}{2}\right)^k$ and it can be negligible if we use sufficiently large k.

Similarly, when $p = 5 \bmod 8$, the parties compute and reveal for $1 \le j \le k$

$$c_j = ar_j + b'_j r'^2_j \bmod p$$

instead of $c_j = ar_j + b_j r'^2_j \bmod p$ where $b'_j = -2^{-1}(b_j - 3)$.

Note that $b'_j \in_R \{2, 1\}$ because $b_j \in_R \{-1, 1\}$. Therefore, noting that $\left(\frac{2}{p}\right) = -1$, we can notice that $a = 0 \Rightarrow \left(\frac{c_j}{p}\right) = \left(\frac{b'_j r'^2_j}{p}\right) = b_j$ with prob. 1, and that $a \neq 0 \Rightarrow \left(\frac{c_j}{p}\right) = b_j$ with prob. $\frac{1}{2}$. The rest of computation can be done as we did for $p = 3 \bmod 4$.

Though we assumed, for simplicity, that $p = 3 \bmod 4$ or that $p = 5 \bmod 8$, actually we can extend the idea to arbitrary primes if we generate $b_j \in_R \{y, 1\}$ such that $\left(\frac{y}{p}\right) = -1$.

Quadratic Residuosity Test Protocol. Incidentally, by using the random secret $b_j r_j^2$ in Section 6.5, we can also construct a quadratic residuosity test protocol where, given $[a \in \mathbb{Z}_p^*]_p$, the parties can compute $[\left(\frac{a}{p}\right)]_p$ as follows:

Here we assume that $p = 3 \bmod 4$ for simplicity. The parties generate $[br^2]_p$ in the same way as $b_j r_j^2$ is generated in Sect. 6.5, and reveal $c = br^2 a$. If c is a zero, the parties retry. The parties can compute $[\left(\frac{a}{p}\right)]_p$ as $\left(\frac{c}{p}\right)[b]_p$ since $\left(\frac{c}{p}\right) = \left(\frac{b}{p}\right)\left(\frac{a}{p}\right) = b\left(\frac{a}{p}\right)$.

Complexity of Probabilistic Equality Test Protocol. The complexity of computing each component is as follows: 3 rounds (including 2 rounds for random value generation) and $7k$ invocations for $[c_j]_p$'s and 1 rounds and $5k$ invocations for $[\wedge_{j=1}^k x_j]_p$. The total complexity is 4 rounds (including 2 rounds for random value generation) and $12k$ invocations.

7 Implementation

In the real implementation, we can use (odd-even) parallel prefix computation [19, 18] based on carry propagation and generation for the bitwise less-than and bitwise sum protocols as in [4, 15, 23] where the complexity of bitwise less-than is roughly $2 + \log_2(\ell)$ rounds and $3\ell - 1$ invocations ($2\ell - 1$ invocations if one of the two operands is known) and the complexity of bitwise sum is roughly $2\log_2(\ell) - 1$ rounds and $5\ell - 2\log_2(\ell) - 4$ invocations ($4\ell - 2\log_2(\ell) - 4$ invocations if one of the two operands is known). Also, instead of joint random number sharing, we can use non-interactive pseudo-random secret sharing by Cramer, Damgård and Ishai [9] in the secret sharing setting in order to reduce the round and communication complexities. In Table 3, we summarize the number of invocations of main subprotocols in each protocol. Whether we use constant-round subprotocols or non-constant-round subprotocols as building blocks, our constructions are more efficient according to Table 3. Though, in the comparison protocol, we need 3 invocations of joint random number bitwise-sharing compared with 2 in [11], this can be done in advance and our protocol seems more advantageous.

Table 3. Number of Invocations of Subprotocols

Protocol		Random Bitwise-Sharing	Bitwise Less-Than	Bitwise Sum
Bit-Decomposition	[11]	1	1	2
	Proposed	1	1	1
Interval Test	[11]	1	3	2
	Proposed	1	2	0
Comparison	[11]	2	3	4
	Proposed	3	3	0
Equality Test	[11]	1	1	2
	Proposed1	1	0	0

Acknowledgements.

We would like to thank Tomas Toft for giving us the idea in Section 6.2 that led to an efficient comparison protocol and the information on [24]. We also thank Prof. Ivan Damgård for suggesting the possibility of using quadratic residues to construct an efficient equality test protocol. We are also grateful to the anonymous reviewers and Prof. C.Pandu Rangan for their helpful comments.

References

1. J. Algesheimer, J. Camenisch, and V. Shoup, “Efficient computation modulo a shared secret with application to the generation of shared safe-prime products,” CRYPTO’02, LNCS 2442, pp.417–432, Springer Verlag, 2002.
2. J. Bar-Ilan and D. Beaver, “Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction,” Proc. ACM Symposium on Principles of Distributed Computing, pp.201–209, 1989.
3. M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorem for non-cryptographic fault-tolerant distributed computation,” 20th Annual ACM Symposium on Theory of Computing, pp.1–10, 1988.
4. P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft, “A practical implementation of secure auctions based on multiparty integer computation,” Financial Cryptography 2006, LNCS 4107, pp.142–147, Springer Verlag, 2006.
5. A.K. Chandra, S. Fortune, and R.J. Lipton, “Lower bounds for constant depth circuits for prefix problems,” ICALP, LNCS 154, pp.109–117, Springer Verlag, 1983.
6. A.K. Chandra, S. Fortune, and R.J. Lipton, “Unbounded fan-in circuits and associative functions,” Proc. 15th ACM Symposium on Theory of Computing, pp.52–60, 1983.
7. D. Chaum, C. Crêpeau, and I. Damgård, “Multi-party unconditionally secure protocols,” Proc. ACM STOC’88, pp.11–19, 1988.
8. R. Cramer and I. Damgård, “Secure distributed linear algebra in a constant number of rounds,” CRYPTO’01, LNCS 2139, pp.119–136, Springer Verlag, 2001.
9. R. Cramer, I. Damgård, and Y. Ishai, “Share conversion, pseudorandom secret sharing and applications to secure computation,” Proc. 2nd Theory of Cryptography Conference, LNCS 3378, pp.342–362, Springer Verlag, 2005.

10. R. Cramer, I. Damgård, and J.B. Nielsen, "Multiparty computation from threshold homomorphic encryption," EUROCRYPT'01, LNCS 2045, pp.280–300, Springer Verlag, 2001.
11. I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," Proc. 3rd Theory of Cryptography Conference, LNCS 3876, pp.285–304, Springer Verlag, 2006.
12. I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," PKC 2001, LNCS 1992, pp.119–136, Springer Verlag, 2001.
13. 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.
14. P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," Financial Cryptography 2000, LNCS 1962, pp.90–104, Springer Verlag, 2000.
15. S. L. From and T. Jakobsen, "Secure multi-party computation on integers," Master's Thesis, `http://www.daimi.au.dk/~mas/thesis/index.html`, 2006
16. 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.
17. 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.
18. H. Jordan and G. Alaghband "Fundamentals of parallel processing," Prentice Hall, 2003.
19. R. Ladner and M. Fischer, "Parallel prefix computation," Journal of the Association for Computing Machinery vol.27, pp.831–838, 1980.
20. E. Ong and J. Kubiatowicz, "Optimizing robustness while generating shared secret safe primes," PKC 2005, LNCS 3386, pp.120–137, Springer Verlag, 2005.
21. B. Schoenmakers and P. Tuyls, "Efficient binary conversion for Paillier encrypted values," EUROCRYPT'06, LNCS 4004, pp.522–537, Springer Verlag, 2006.
22. A. Shamir, "How to share a secret," Communications of ACM, vol.22, no.11, pp.612–613, 1979.
23. T. Toft, "Secure integer computation with applications in economy," `http://www.aicis.alexandra.dk/uk/projects/scet_demo.htm#Tof05`, Available from `http://www.daimi.au.dk/~tomas/publications/progress.pdf`
24. T. Toft, "An efficient, unconditionally secure equality test for secret shared values," Workshop on Models for Cryptographic Protocols (MCP 2006), Abstract available from `http://www.daimi.au.dk/~buus/mcp2006/talks/T.pdf`
25. A. Yao, "Protocols for secure computation," Proc. 23rd FOCS, pp.160–164, 1982.

A Complexity of Bitwise Sum Protocol

Based on [11] (see unbounded fan-in carry propagation in Section 6.4), the complexity of the bitwise sum protocol is evaluated as follows:

If the Prefix-And is computed with x rounds and $y \times \ell$ invocations, the complexity of the bitwise sum protocol is upper bounded by $2(x+1)+1$ rounds and $(2(y+6)+1)\ell \log_2 \ell$ invocations. Assuming that all the random values are generated in the first 2 rounds and that the complexity of the Prefix-And is 5 rounds (not including 2 rounds for random value generation) and 17ℓ invocations, the total complexity is 15 rounds (including 2 rounds for random value generation) and $47\ell \log_2 \ell$ invocations.