Practical and Secure Solutions for Integer Comparison

Juan Garay[1], Berry Schoenmakers[2], and José Villegas[2]

[1] Bell Labs – Alcatel-Lucent, 600 Mountain Ave., Murray Hill, NJ 07974
garay@research.bell-labs.com
[2] Dept. of Mathematics and Computing Science, TU Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
berry@win.tue.nl, j.a.villegas@tue.nl

Abstract. Yao's classical *millionaires' problem* is about securely determining whether $x > y$, given two input values x, y, which are held as private inputs by two parties, respectively. The output $x > y$ becomes known to both parties.

In this paper, we consider a variant of Yao's problem in which the inputs x, y as well as the output bit $x > y$ are encrypted. Referring to the framework of secure n-party computation based on threshold homomorphic cryptosystems as put forth by Cramer, Damgård, and Nielsen at Eurocrypt 2001, we develop solutions for integer comparison, which take as input two lists of encrypted bits representing x and y, respectively, and produce an encrypted bit indicating whether $x > y$ as output. Secure integer comparison is an important building block for applications such as secure auctions.

In this paper, our focus is on the two-party case, although most of our results extend to the multi-party case. We propose new logarithmic-round and constant-round protocols for this setting, which achieve simultaneously very low communication and computational complexities. We analyze the protocols in detail and show that our solutions compare favorably to other known solutions.

Keywords: Millionaires' problem; secure multi-party computation; homomorphic encryption.

1 Introduction

The *millionaires' problem*, introduced by Yao [Yao82], involves two parties who want to compare their riches: they wish to know who is richer but do not want to disclose any other information about their riches to each other. More formally, the problem is to find a two-party protocol for the secure evaluation of the function $f(x, y) = [x > y]$ where the bracket notation $[B]$, for a condition B, is defined by $[B] = 1$ if B holds and $[B] = 0$ otherwise (this is called Iverson's convention; see [Knu97]).

T. Okamoto and X. Wang (Eds.): PKC 2007, LNCS 4450, pp. 330–342, 2007.

Rather than requiring that the inputs x and y are actually known as private inputs to the parties, we will work in the more general setting where the inputs are not necessarily known to the parties running the protocol. Instead, the inputs to the protocol may be given as encrypted values only, and the output will also be made available in encrypted form. Note that the inputs to our protocols will actually be encryptions of the individual bits, representing the integers to be compared. For these encryptions we will use a threshold homomorphic cryptosystem, as in the framework of secure n-party computation based on threshold homomorphic cryptosystems put forth by Cramer, Damgård, and Nielsen [CDN01]. In line with this, we consider the case of an *active*, static adversary[1], i.e., we consider the malicious case.

Requiring (i) that the inputs are given in encrypted form (without anyone knowing these inputs) and (ii) that the output bit $[x > y]$ also be encrypted (without anyone learning its value) sets our problem setting apart from the setting of Yao's paper [Yao82] and much of the follow-up literature. Indeed, consider computing $[x = y]$ in the case of encrypted inputs but *public* output, where the following well-known solution works. Let $[\![M]\!]$ denote a (probabilistic) encryption of a message M in a threshold homomorphic cryptosystem. Given encryptions $[\![x]\!]$ and $[\![y]\!]$, the encryption $[\![x - y]\!]$ is publicly computed. Furthermore, the parties jointly compute an encryption $[\![r]\!]$ for a (jointly) random r. Using one invocation of a secure multiplication protocol, the parties then produce encryption $[\![(x - y)r]\!]$, which is jointly decrypted. If the result is 0, then $x = y$; otherwise, $x \neq y$, and the result is a random number. In contrast, when the output is required in encrypted form, such simple solutions are not known and typically protocols (including ours) work over the encrypted values of the binary representation of the inputs x and y.

Furthermore, unlike many publications on the millionaires' problem, we consider the malicious case rather than the semi-honest (or honest-but-curious) case.

1.1 Our Contributions

The contributions of this paper are as follows:

- A logarithmic-round protocol for secure integer comparison, which is based on an elegant Boolean circuit for integer comparison of depth $\log_2 m$ for m-bit integers. In addition, the size of the circuit is only $3m$ (counting the number of secure multiplication gates). The circuit can be readily used as a drop-in replacement for the $O(1)$-depth circuit for integer comparison in [DFK$^+$06], which is only of theoretical interest as it uses 19 rounds and $22m$ secure multiplications. Note that the depth of our log-depth circuit exceeds their constant-depth circuit for integer comparison only if the inputs consist of integers of *bit length* $m = 2^{20}$ *or longer.*)

[1] In principle, the case of adaptive adversaries could be handled at the expense of additional tools (e.g., [DN00, DN03, GMY03]); in this paper we focus on the static (and stand-alone) case.

– A constant-round protocol for secure integer comparison for which the number of rounds is a small constant and the number of secure multiplications is a small multiple of m. Our constant-round solution is restricted to the case of two parties (or, rather, any constant number of parties). Our protocol builds on a protocol by Blake and Kolesnikov [BK04] for integer comparison for a different setting. In particular, we provide an efficient technique for securely returning the output bit in an encrypted form.

We like to stress that application of our log-depth circuit is not restricted to the framework of [CDN01]: the circuit can be used in any framework for secure n-party computation that assumes that the function to be computed is given as a circuit. In particular, the log-depth circuit can be used for secure computation based on verifiable secret sharing, thus yielding solutions which are unconditionally secure—rather than computationally secure, as described in this paper.

Furthermore, the proof of security of our constant-round protocol is interesting in its own right. Theorem 1, as explained below, essentially captures the security of the protocol in a modular way. Here, we have adopted the approach suggested recently in [ST06], and we show how the required simulator can be built even though our protocol is of a much different nature than the ones in [ST06].

1.2 Related Work

There appear to be only a few publications in the literature which consider encrypted inputs *and* outputs for integer comparison. Above we have already mentioned the work of Damgård *et al.* [DFK+06]. The main difference is that they work in an unconditional setting, reflected by the use of sharings for an underlying linear secret sharing scheme, while we work in the *cryptographic* model where we use encryptions for an underlying threshold homomorphic cryptosystem.

Together with a secure multiplication protocol for a homomorphic threshold ElGamal scheme, Schoenmakers and Tuyls [ST04] also present a solution for secure integer comparison for encrypted inputs and outputs. Their solution, however, requires a linear ($O(m)$) number of rounds and secure multiplication gates. With more relaxed requirements than ours, Brandt [Bra06] presents a solution where the inputs are encrypted but the output is in the clear for both participants, and furthermore, it is not 0 or 1 but instead 0 or 'random,' which limits its applicability.

A different approach to solve the integer comparison problem is when one of the parties acts as a server. In this setting, say, Alice knows the private keys to open encryptions and Bob works over his input bits and Alice's encrypted input bits to produce some information that allows Alice to know the output of the function being evaluated. Examples of these approaches to integer comparison are presented in [DiC00, Fis01, BK04, LT05]. In contrast to our solutions, these solutions do not provide encrypted output and the actual encrypted inputs are known to the parties running the protocols.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we introduce the main building blocks used by our protocols and we give some background on threshold homomorphic cryptosystems. In Section 3 we present our two new protocols for integer comparison, together with their proof of security (specifically, of the second protocol, as the proof of the first protocol follows directly from the security guarantees provided by the [CDN01] setting). We conclude in Section 4 with a brief performance analysis and comparison to existing results.

2 Preliminaries

Our results apply to any threshold homomorphic cryptosystem, such as those based on ElGamal or Paillier. It is assumed that a secure multiplication protocol is available, as in [CDN01, ST04]. Since we only need secure multiplication of binary values, we use the *conditional gate* of [ST04], which allows for an efficient implementation based on threshold homomorphic ElGamal—which in turn allows for the use of elliptic curves, hence yielding compact and efficient implementations.

We write $[\![x]\!]$ for a (probabilistic) encryption of the value x, using the public key of the underlying threshold homomorphic ElGamal cryptosystem. Further, let $\mathbb{Z}_q$ denote the message space, for a large prime q (of, say, size 160 bits). The cyclic group G used for ElGamal is also of order q, and we assume that elements of G are represented using $|q|$ bits only (which is the case for elliptic curves). Thus, an ElGamal encryption consisting of two group elements is of size $2|q|$.

In order to withstand active attacks, we use Σ-protocols [CDS94], a standard type of zero-knowledge proofs/arguments. Assuming the random oracle model, all proofs can be converted into non-interactive ones and can be simulated easily.

As mentioned above, we make use of *secure multiplication gates* which on input $[\![x]\!]$ and $[\![y]\!]$ allows two or more parties (who share the private key of the underlying threshold homomorphic cryptosystem) to jointly compute an encryption $[\![xy]\!]$. Secure multiplication gates can be implemented in a constant number of rounds [CDN01], using the Paillier cryptosystem. Using a number of rounds linear in the number of parties (which is constant in case of two-party computation), the *conditional gate* [ST04] can be used instead, in case one of the multiplicands is from a two-valued domain (e.g., if $x \in \{0, 1\}$).

Furthermore, in case one of inputs, say, x is private to one of the parties, a simplified multiplication protocol can be used with no interaction between the parties. The protocol consists in letting the party knowing the private value x broadcast a re-encryption of $[\![xy]\!] = [\![y]\!]^x$ using the homomorphic properties of the scheme, and generate a Σ-proof showing that $[\![xy]\!]$ was correctly computed with respect to $[\![x]\!]$ and $[\![y]\!]$. Following [ST04], we will refer to this protocol as the *private-multiplier* gate.

For the performance comparisons presented at the end of this paper, we will assume a setup using a (2,2)-threshold homomorphic ElGamal cryptosystem. We note that in this case a conditional gate requires about 50 exponentiations

and $34|q|$ bits of communication, per invocation. Similarly, a private-multiplier gate requires about 10 exponentiations and $6|q|$ bits of communication, per invocation. In the same setting, a threshold decryption requires 6 exponentiations and $6|q|$ bits of communication.

A final tool that we will use are *verifiable mixes* [SK95], a tool for verifiably mixing lists of ciphertexts. More formally, a verifiable mix takes as input a list of encryptions $[\![x_1]\!], \ldots, [\![x_m]\!]$, and produces another list of encryptions $[\![x'_1]\!], \ldots, [\![x'_m]\!]$ as output such that $[\![x'_{\pi(1)}]\!] = [\![x_1]\!] * [\![0]\!]$, $\ldots$, $[\![x'_{\pi(m)}]\!] = [\![x_m]\!] * [\![0]\!]$ for some random permutation π of $\{1, \ldots, m\}$. Here, each occurrence of $[\![0]\!]$ denotes a probabilistic encryption of 0.

A verifiable mix also outputs a non-interactive zero-knowledge proof (for which we assume the random-oracle model throughout). For concreteness, we assume Groth's efficient proof [Gro03], which for our setting requires about $14m$ exponentiations and is of size $6m|q|$ bits.

We are now ready to describe our protocols for integer comparison.

3 New Solutions to the Integer Comparison Problem

In this section we present two new protocols for integer comparison following different approaches. In both cases, the inputs x and y are given as sequences of encrypted bits, $[\![x_{m-1}]\!], \ldots, [\![x_0]\!]$ and $[\![y_{m-1}]\!], \ldots, [\![y_0]\!]$, with $x = \sum_{i=0}^{m-1} x_i 2^i$, $y = \sum_{i=0}^{m-1} y_i 2^i$. The output is $[\![x > y]\!]$. Hence, both inputs and output are available in encrypted form only.

As a starting point and for later comparison, we first the linear-depth circuit of [ST04] for computing $x > y$, using simple arithmetic gates only (addition, subtraction, conditional gates). The circuit (or, oblivious program) is fully described by the following recurrence:

$$t_0 = 0, \qquad t_{i+1} = (1 - (x_i - y_i)^2)t_i + x_i(1 - y_i),$$

where t_m is the output bit (hence $t_m = [x > y]$). Rather than starting from the most significant bit, this circuit computes $[x > y]$ starting from the least significant bit. Although somewhat counterintuitive, the advantage of this approach is that the circuit contains $2m - 1$ conditional gates only (compared to about $3m$ conditional gates when starting from the most significant bit, see [ST04]).

A disadvantage is that the depth of the circuit is m, hence inducing a critical path of m sequential secure multiplications (the terms $[\![x_1 y_1]\!], \ldots, [\![x_m y_m]\!]$ can be computed in parallel, but the computation of $t_1, \ldots, t_m$ must be done sequential). The computational complexity and communication complexity of a protocol for integer comparison based on this circuit is thus determined by the work required for the conditional gates. For later comparison, in the two-party case, we have about $100m$ exponentiations and $68m|q|$ bits of communication—and a linear number of rounds.

3.1 Logarithmic Round Complexity with Low Computational Complexity

The result in this section shows how to reduce the depth of the circuit to $O(\log m)$ without increasing its size beyond $O(m)$. The idea relies on the following simple but crucial property of integer comparison. Write $x = X_1X_0$ and $y = Y_1Y_0$ as bit strings, where $0 \le |X_1| = |Y_1| \le m$ and $0 \le |X_0| = |Y_0| \le m$. Then,

$$[x > y] = \begin{cases} [X_1 > Y_1], & X_1 \neq Y_1; \\ [X_0 > Y_0], & X_1 = Y_1, \end{cases}$$

which may be "arithmetized" as

$$[x > y] = [X_1 > Y_1] + [X_1 = Y_1][X_0 > Y_0].$$

This property suggests a protocol that would first split the bit strings x and y in about equally long parts, compare these parts recursively, and then combine these to produce the final output. To evaluate the expression for $[x > y]$ using simple arithmetic gates, we introduce the following auxiliary function:

$$z(x, y) = [x = y] = 1 - (x - y)^2$$

Let $t_{i,j}$ stand for the value of $>$ when applied to the substrings $x_{i+j-1}, \ldots, x_{i+1}, x_i$ and $y_{i+j-1}, \ldots, y_{i+1}, y_i$. Expressed explicitly in terms of the bits of x and y, a full solution for $[x > y]$ is obtained by evaluating $t_{0,m}$ from (using $l = \lfloor j/2 \rfloor$)[2]:

$$t_{i,j} = \begin{cases} x_i - x_i y_i, & j = 1; \\ t_{i+l,j-l} + z_{i+l,j-l} t_{i,l}, & j > 1. \end{cases}$$

$$z_{i,j} = \begin{cases} 1 - x_i + 2x_i y_i - y_i, & j = 1; \\ z_{i+l,j-l} z_{i,l}, & j > 1. \end{cases}$$

Correctness of the computation should be immediate, and its security follows from the security guarantees provided by the framework we are considering [CDN01], assuming secure arithmetic gates.

Regarding overhead, the number of conditional gates required for $z_{i,j}$ is $2j-1$. The number of conditional gates for $t_{i,j}$ is $j - 1$, not counting the conditional gates for z. Thus, the total number of conditional gates for $t_{0,m}$ is bounded above by $3m - 2$. About $\log_2 m$ conditional gates can be saved by observing that some z-values are not needed for the evaluation of t.

The computational and communication complexities are dominated by the number of conditional gates. In the worst case, $3m - 2$ conditional gates are required, resulting in about $150m$ exponentiations and $102m|q|$ broadcast bits.

[2] Any value l, $0 < l < j$, actually works, but only $l \approx j/2$ gives logarithmic depth. The msb-to-lsb and lsb-to-msb circuits in [ST04] are special cases, obtained respectively by setting $l = 1$ and $l = j - 1$.

The depth of the circuit is exactly $\lceil \log_2 m \rceil$, hence $O(\log m)$ with hidden constant equal to 1 for the base-2 logarithm.

As a further remark we note that this log-depth circuit allows for the computation of $\mathrm{sgn}(x - y)$ at virtually no extra cost. Here, $\mathrm{sgn}(z)$ is the signum function, which is equal to the sign of z (which is equal to -1 if $z < 0$, 0 if $z = 0$, and 1 if $z > 0$). This follows form the fact that the circuit also computes $[x = y]$, next to $[x > y]$, hence one obtains $\mathrm{sgn}(x - y) = 2[x > y] - 1 + [x = y]$ as well.

3.2 Constant Round Complexity with Low Computational Complexity

In this section we seek to reduce the round complexity to $O(1)$, adopting an approach quite different from the one above. We consider the problem of computing $[\![[x > y]]\!]$ in the two-party case, and we wish to achieve a low, constant-round complexity while keeping the size of the circuit small as well.

First, we note that the $O(1)$-depth and $O(m)$-size circuit for integer comparison of [DFK+06] is only of theoretical interest to us: the depth of the circuit is actually 19, and its size is $22m$ (only counting secure multiplication gates). For a result that possibly competes with our logarithmic solution we take the protocol for *conditional oblivious transfer* of Blake and Kolesnikov [BK04] (where the condition is also an integer comparison) as a starting point. The main idea in that protocol is to calculate the first position where the bits of x and y differ, starting from the most-significant bit. Let i^* be that position; then $x_{i^*} - y_{i^*} \in \{-1, 1\}$ indicates whether $x > y$ or not. Jumping ahead a little, the position i^* will be determined as the unique index satisfying $\gamma_{i^*} = 1$ (which is guaranteed to exist if we assume $x \neq y$; see below). Of course, the value of i^* must remain hidden, which is achieved by the parties randomly permuting (i.e., mixing) the relevant sequences.

The protocol is described in detail below. As said above, our starting point is the protocol in [BK04] for the passive adversary setting. New ingredients include the fact that we allow for encrypted inputs $[\![x]\!]$ and $[\![y]\!]$, rather than private inputs x and y. Accordingly, we use a (2,2)-threshold homomorphic cryptosystem instead of just a homomorphic cryptosystem, and we use secure multiplication (conditional gates). Furthermore, we use a specific kind of blinding at the end of the protocol in order to extract the outcome of the integer comparison in encrypted form. Finally, as an important difference, we can actually use other homomorphic cryptosystems, such as ElGamal, whereas [BK04] makes essential use of Paillier.

Constant-round protocol. The protocol consists of the following steps:

1. Using m conditional gates, parties A and B jointly compute $[\![f_i]\!] = [\![[x_i \neq y_i]]\!]$. Then they publicly compute the γ-sequence: $[\![\gamma_m]\!] = [\![0]\!]$; $[\![\gamma_i]\!] = [\![2\gamma_{i+1} + f_i]\!]$, for $i = m - 1, \ldots, 0$.
2. For $i = m - 1, \ldots, 0$, party A broadcasts $[\![r_i^A]\!]$ for random $r_i^A \in_R \mathbb{Z}_q$ and produces sequence $[\![u_i^A]\!] = [\![r_i^A(\gamma_i - 1)]\!]$ using a private-multiplier gate.

3. Party B does the same with $[\![r_i^B]\!]$ producing sequence $[\![u_i^B]\!] = [\![r_i^B(\gamma_i - 1)]\!]$, where $r_i^B \in_R \mathbb{Z}_q$. Now they publicly produce sequence $[\![u_i]\!] = [\![u_i^A]\!][\![u_i^B]\!]$ $[\![x_i - y_i]\!] = [\![(r_i^A + r_i^B)(\gamma_i - 1) + (x_i - y_i)]\!]$.
4. Party A verifiably mixes sequence $[\![u_i]\!]$ producing sequence $[\![u'_i]\!]$.
5. Party B verifiably mixes sequence $[\![u'_i]\!]$ producing sequence $[\![v_i]\!]$.
   Now, parties A and B take turns to multiply this last sequence by a randomly selected number in $\{-1, 1\}$:
6. Party A broadcasts $[\![s_A]\!]$, $s_A \in_R \{-1, 1\}$, and uses a private-multiplier gate to produce sequence $[\![v'_i]\!] = [\![s_A v_i]\!]$. A proof that $[\![s_A]\!]$ is an encryption of either -1 or 1 is also given.
7. Party B does the same, broadcasting $[\![s_B]\!]$, $s_B \in_R \{-1, 1\}$, and producing sequence $[\![w_i]\!] = [\![s_B v'_i]\!]$ along with the required proofs.
8. Finally, parties A and B proceed to decrypt the sequence $[\![w_i]\!]$ until they find the unique index i^* satisfying $w_{i^*} \in \{-1, 1\}$. The output is defined as $[\![(v_{i^*} + 1)/2]\!]$.

The value v_{i^*} is either -1 or 1, hence $(v_{i^*} + 1)/2$ is either 0 or 1. This linear transformation can be done for free because of homomorphic properties.

The above protocol assumes that $x \neq y$, in order that index i^* is well defined. If $x = y$, then no entry in the w-sequence will be equal to -1 or 1. One can put "sentinels" to resolve possible equality, by setting $f_{-1} = 1$ and $u_{-1} = (r_{-1}^A + r_{-1}^B)(\gamma - 1) + 1$. The rest of the protocol is adapted accordingly.

In case the output need not be encrypted, steps 6 and 7 are omitted, and the participants directly open the sequence v to find the position i^* where v_{i^*} is in $\{-1, 1\}$, where -1 means that x is less than or equal to y, and 1 means x is greater than y.

For the complexities, the number of rounds for the protocol is small: at most 9 rounds (two rounds for the conditional gates in step 1, and one round for each of the subsequent steps). For the number of exponentiations, we have $50m$ for the conditional gates (step 1), $40m$ for the multiplication gates (steps 2, 3, 6, and 7), $28m$ for the verifiable mixes, and $3m$ for the decryption ($m/2$ expected decryptions), which amounts to $124m$ exponentiations in total. Similarly, $77m|q|$ is the number of bits of communication. We have omitted further optimizations for clarity of exposition.

The protocol easily extends to the multiparty case, but since the mixing is done sequentially, constant round complexity is not achieved (note that secure multiplication gates can be constant-round even in the multi-party case if Paillier encryption is used, as in [CDN01]).

Proof of security. For the proof of security, we want to be able to simulate this protocol assuming that one of the participants is corrupted. The idea is to give the simulator the inputs $[\![x_i]\!]$ and $[\![y_i]\!]$ in such a way that a consistent view of the protocol can be constructed without making use of the private information of the honest participant.

We first review the simulation requirements for the building blocks. In order to simulate a conditional gate, encryptions $[\![x]\!]$ and $[\![y]\!]$ are required, as well

as one encryption of $[\![xy]\!]$ with the requirement that $x \in \{-1, 1\}$ (or, any other two-value domain) and the contents of the encryptions are consistent. The actual values x,y and xy need not be known. The same holds for the private multiplier gate, where in this case the proof of knowledge of, say, x is simulated. For a threshold decryption, we need to provide both $[\![x]\!]$ and x to the corresponding simulator.

We now turn to the overall simulation strategy. We note that one problem already arises at the first step of the protocol: in order to simulate the conditional gate invocations in Step 1, the simulator has to produce $[\![x_i y_i]\!]$ only given $[\![x_i]\!]$ and $[\![y_i]\!]$, which is impossible! We circumvent such problems by adopting the approach recently introduced in [ST06], in which it is explained that simulation for input/output pairs of a special form (see Theorem 1 below) suffice to ensure integration with the framework of [CDN01]. This is a consequence of the fact that the security proof in [CDN01] centers around the construction of a so-called *YAD^b distribution*, which is defined as a function of an encrypted bit $[\![b]\!]$.

The structure of the security proof [CDN01] follows an ideal-model/real-model approach. The YAD^0 distribution is identical to the distribution of the ideal case, whereas the YAD^1 distribution is statistically indistinguishable from the distribution in the real case. Therefore, if an adversary can distinguish between the ideal/real cases, it implies that the adversary can distinguish the YAD^0 distribution from the YAD^1 distribution. But as the choice between these two distributions is determined by the value of the encrypted bit b, it follows that the distinguisher for the ideal/real cases is a distinguisher for the underlying encryption scheme. And this is done in tight way, i.e., without loss in the success probability for the distinguisher. (See [CDN01, ST06] for more details.)

Thus, it is sufficient to show a simulation for inputs of a special form, namely, $[\![\tilde{x}]\!] = [\![(1-b)x^{(0)} + bx^{(1)}]\!]$, where $x^{(0)}$ and $x^{(1)}$ are given in the clear to the simulator, but b is only given in encrypted form $[\![b]\!]$. The values $x^{(0)}$ and $x^{(1)}$ correspond to the values arising in the YAD^0 and YAD^1 cases, respectively.

Theorem 1. *Given input values $x_i^{(0)}$, $y_i^{(0)}$, $x_i^{(1)}$ and $y_i^{(1)}$ and an encryption $[\![b]\!]$ with $b \in \{0, 1\}$ the above protocol can be simulated statistically indistinguishably for inputs $[\![\tilde{x_i}]\!] = [\![(1-b)x_i^{(0)} + bx_i^{(1)}]\!]$ and $[\![\tilde{y_i}]\!] = [\![(1-b)y_i^{(0)} + by_i^{(1)}]\!]$.*

Proof. Let $x_i^{(0)}$, $y_i^{(0)}$, $x_i^{(1)}$ and $y_i^{(1)}$ and encryption $[\![b]\!]$ with $b \in \{0, 1\}$ be given. Assuming that party A is corrupted, the simulation works as follows:

1. For Step 1, we rely on the simulator for the conditional gates, which we need to provide with the inputs $[\![\tilde{x_i}]\!]$ and $[\![\tilde{y_i}]\!]$ and the corresponding output $[\![\tilde{f_i}]\!] = [\![\tilde{x_i}\tilde{y_i}]\!]$. The latter values are computed as $[\![(1-b)x_i^{(0)}y_i^{(0)} + bx_i^{(1)}y_i^{(1)}]\!]$, using $[\![b]\!]$ and the homomorphic properties of the cryptosystem.

   Similarly, the simulator also computes $[\![\tilde{\gamma_i}]\!] = [\![(1-b)\gamma_i^{(0)} + b\gamma_i^{(1)}]\!]$. Let i_0 and i_1 denote the indices such that $\gamma_{i_0}^{(0)} = \gamma_{i_1}^{(1)} = 1$ as these values are known to the simulator.
2. Next, we let party A do her work. She will broadcast $[\![\tilde{r}_i^A]\!]$ and $[\![\tilde{u}_i^A]\!]$, for all i. The values $\tilde{r}_i^A$ can be extracted by rewinding the proof of knowledge of the private-multiplier invocation.

3. The idea of this step is to generate values $r_i^{B(j)}$ such that the simulator may put equal values (up to sign) in the u-sequences, which will later decrypt to the same value independently of b. For this the simulator does the following. First, he selects $s_B^{(0)} \in_R \{-1, 1\}$. The value of $s_B^{(1)}$ depends on the result of the comparison of $x^{(0)}$ against $y^{(0)}$, and $x^{(1)}$ against $y^{(1)}$. If both comparisons have the same result, then $s_B^{(1)} = s_B^{(0)}$, otherwise $s_B^{(1)} = -s_B^{(0)}$.

   Now the simulator selects $r_i^{B(0)}, r_i^{B(1)}$ in such a way that $u_i^{(0)}$ and $u_i^{(1)}$ satisfy the following:
   (a) $u_i^{(0)} s_B^{(0)} = u_i^{(1)} s_B^{(1)}$, for $i \notin \{i_0, i_1\}$;
   (b) $u_{i_0}^{(1)} s_B^{(1)} = u_{i_1}^{(0)} s_B^{(0)}$;
   (c) $u_{i_0}^{(0)} s_B^{(0)} = u_{i_1}^{(1)} s_B^{(1)}$.
   First, we note that, for $j = 0, 1$:

$$u_i^{(j)} = (\tilde{r}_i^A + r_i^{B(j)})(\gamma_i^{(j)} - 1) + (x_i^{(j)} - y_i^{(j)}).$$

   For case (a) we essentially need that $s_B^{(0)} s_B^{(1)} u_i^{(0)} = u_i^{(1)}$, which means that

$$s_B^{(0)} s_B^{(1)} \big((\tilde{r}_i^A + r_i^{B(0)})(\gamma_i^{(0)} - 1) + (x_i^{(0)} - y_i^{(0)})\big) = (\tilde{r}_i^A + r_i^{B(1)})(\gamma_i^{(1)} - 1) + (x_i^{(1)} - y_i^{(1)}),$$

   where $i \notin \{i_0, i_1\}$.

   This can be achieved by first selecting $r_i^{B(0)}$ at random, and then isolating and obtaining $r_i^{B(1)}$ (which in turn is random in each selection of b).
   Similarly, in case (b), we require that $s_B^{(1)} s_B^{(0)} u_{i_0}^{(1)} = u_{i_1}^{(0)}$, which is equivalent to

$$s_B^{(1)} s_B^{(0)} \big((\tilde{r}_{i_0}^A + r_{i_0}^{B(1)})(\gamma_{i_0}^{(1)} - 1) + (x_{i_0}^{(1)} - y_{i_0}^{(1)})\big) = (\tilde{r}_{i_1}^A + r_{i_1}^{B(0)})(\gamma_{i_1}^{(0)} - 1) + (x_{i_1}^{(0)} - y_{i_1}^{(0)}),$$

   and it is solved as in case (a).

   For case (c), just taking $r_{i_0}^{B(0)}$ and $r_{i_1}^{B(1)}$ at random is enough: in those positions the γ-sequences take the value 1 and the randomization is "lost" when considering u-sequences.

   The simulator now prepares $[\![\tilde{r}_i^B]\!]$ as $[\![(1-b) r_i^{B(0)} + b r_i^{B(1)}]\!]$ and $[\![\tilde{u}_i^B]\!]$ as $[\![\tilde{r}_i^B (\tilde{\gamma}_i - 1)]\!]$, for all i. These encrypted values are broadcast, and the simulator for the private-multiplier gate is invoked, with multiplicands $[\![\tilde{r}_i^B]\!]$ and $[\![\tilde{\gamma}_i]\!]$, and result $[\![(1-b) r_i^{B(0)} \gamma_i^{(0)} + b r_i^{B(1)} \gamma_i^{(1)}]\!]$.

   The sequence $[\![\tilde{u}_i]\!]$ is constructed as in the protocol:

$$[\![\tilde{u}_i]\!] = [\![\tilde{u}_i^A]\!] [\![\tilde{u}_i^B]\!] [\![\tilde{x}_i - \tilde{y}_i]\!].$$

   By construction, it follows that $[\![\tilde{u}_i]\!] = [\![(1-b) u_i^{(0)} + b u_i^{(1)}]\!]$, for all i.
4. The simulator lets party A mix the sequence $[\![\tilde{u}_i]\!]$, producing a new sequence $[\![\tilde{u}'_i]\!]$. The simulator can also extract the permutation π_A that links both sequences.
5. Now the simulator randomly selects two indices, call them $\tilde{i}^*$ and $\tilde{i}^{**}$, and constructs two permutations $\pi_B^{(0)}$ and $\pi_B^{(1)}$ as follows:

- $\pi_B^{(0)}(\pi_A(i_0)) = \pi_B^{(1)}(\pi_A(i_1)) = \tilde{i}^*$;
- $\pi_B^{(0)}(\pi_A(i_1)) = \pi_B^{(1)}(\pi_A(i_0)) = \tilde{i}^{**}$;
- for the remaining positions the permutations are randomly defined under the condition that $\pi_B^{(0)}(\pi_A(i)) = \pi_B^{(1)}(\pi_A(i))$, $i \notin \{i_0, i_1\}$.

The next step is to call the simulator of the mix proof depending on $[\![b]\!]$, because the simulator will never know which permutation, $\pi_B^{(0)}$ or $\pi_B^{(1)}$, is actually used. For this, he constructs the sequences $v_i^{(j)} = u^{(j)}_{\pi_A^{-1}(\pi_B^{(j)\,-1}(i))}$, for $j = 0, 1$, and then defines the sequence $[\![\tilde{v}_i]\!] = [\![(1-b)v_i^{(0)} + bv_i^{(1)}]\!]$, for all i. With the mixed sequence broadcast by party A in the previous step and this last sequence, the simulator now calls the simulator for the mix proof.

6. Party A multiplies the entire sequence $[\![\tilde{v}_i]\!]$ by a number $\tilde{s}_A$ (which is extracted from the corresponding private-multiplier proof for $[\![\tilde{s}_A]\!]$), resulting in sequence $[\![\tilde{v}'_i]\!]$.
7. Now the simulator has almost all the work already done. At this stage he constructs $[\![\tilde{s}_B]\!] = [\![(1-b)s_B^{(0)} + bs_B^{(1)}]\!]$, and broadcasts it. Then he constructs the sequence $[\![\tilde{w}_i]\!] = [\![(1-b)v_i^{(0)}\tilde{s}_A s_B^{(0)} + bv_i^{(1)}\tilde{s}_A s_B^{(1)}]\!]$. Note that $\tilde{v}'_i = \tilde{v}_i\tilde{s}_A$. The private-multiplier simulator is now invoked on inputs $[\![\tilde{s}_B]\!]$ and $[\![\tilde{v}'_i]\!]$, and output $[\![\tilde{w}_i]\!]$.
8. To simulate the last step, the simulator can link back the plaintext of encryptions $[\![\tilde{w}_i]\!]$ by using permutation $\pi_A \circ \pi_B^{(j)}$, for $j = 0, 1$; note that the sign of these values is affected by the factor $\tilde{s}_A$. Thus,

$$w_i^{(j)} = \tilde{s}_A s_B^{(j)} u^{(j)}_{\pi_A^{-1}(\pi_B^{(j)\,-1}(i))},$$

for all i, due to the construction at step 5.

Moreover, the plaintexts in $[\![w_i^{(0)}]\!]$ and $[\![w_i^{(1)}]\!]$ are equal, as a result of the work of the simulator at step 3. It also follows that $w_i^{(0)} = w_i^{(1)} = \tilde{w}_i$, independently of $[\![b]\!]$. Hence, the simulator for the threshold decryption is called, for instance, over inputs $[\![\tilde{w}_i]\!]$ and $\tilde{s}_A s_B^{(0)} u^{(0)}_{\pi_A^{-1}(\pi_B^{(0)\,-1}(i))}$.

The values generated in this way by the simulator are consistent, and therefore an adversary cannot statistically distinguish them from the ones resulting in a real execution. The case when party B is corrupted is similar with some minor differences, due to the order in which tasks are executed. This completes the proof. □

4 Conclusions

In this paper we have presented two new solutions to the integer comparison problem. Our first solution achieves a logarithmic round complexity of exactly $\lceil \log_2 m \rceil$ rounds for m-bit integers, whereas the second solution achieves a

Table 1. Comparison of different secure solutions for $[x > y]$

Integer Comparison Solution	No. Exponentiations	Broadcast Bits
Linear-depth circuit [ST04]	$100m$	$68m\|q\|$
Logarithmic-depth circuit	$150m$	$102m\|q\|$
Constant-round protocol (two-party)	$124m$	$77m\|q\|$

constant number of rounds (in the two-party case). In Table 1 we show a comparison between the different solutions presented in this paper and the linear-depth circuit of [ST04].

Evidently, going below $O(m)$ rounds comes at the cost of an increase in computational and communication complexity. For the constant round solution, the additional costs are smaller than for the logarithmic round solution; however, the logarithmic round solution also applies to the multi-party case.

From a practical point of view, our multi-party logarithmic-depth solution is very good compared to the known results so far: communication and computation are are only 50% worse than for a linear-depth solution. Even though $O(1)$-round is not achieved this way, the number of rounds is *very low* when considering integers x and y of practical size, e.g., $m = 32$ or $m = 64$, in which cases the depth is only 5 and 6, respectively.

References

[BK04] I. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology—ASIACRYPT '04*, volume 3329 of *Lecture Notes in Computer Science*, pages 515–529, Berlin, 2004. Springer-Verlag.

[Bra06] F. Brandt. Efficient cryptographic protocol design based on distributed El Gamal encryption. In *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, 2006.

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300, Berlin, 2001. Springer-Verlag. Full version `eprint.iacr.org/2000/055`, October 27, 2000.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Berlin, 1994. Springer-Verlag.

[DFK+06] 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. In *Proc. 3rd Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, Berlin, 2006. Springer-Verlag.

[DN00] I. Damgård and J. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Advances in Cryptology—Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 433–451. Springer, 2000.

[DN03] I. Damgård and J.B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Berlin, 2003. Springer-Verlag.

[DiC00] G. Di Crescenzo. Private Selective Payment Protocols. In *FC '00: Proc. 4th International Conference on Financial Cryptography*, *Lecture Notes in Computer Science*, pages 72–89, London, 2001, Springer-Verlag.

[Fis01] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Progress in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 457–471, Berlin, 2001. Springer-Verlag.

[GMY03] J. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. In *Advances in Cryptology—Eurocrypt 2003*, volume 2656, pages 177–194. Springer, 2003.

[Gro03] J. Groth. A verifable secret shuffle of homomorphic encryptions. In *Public Key Cryptography—PKC '03*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160, Berlin, 2003. Springer-Verlag.

[Knu97] D. E. Knuth. *The Art of Computer Programming (Vol. 1: Fundamental Algorithms)*. Addison Wesley, Reading (MA), 3rd edition, 1997.

[LT05] H. Lin and W. Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 456–466. Springer-Verlag, 2005.

[SK95] K. Sako and J. Kilian. Receipt-free mix-type voting scheme—a practical solution to the implementation of a voting booth. In *Advances in Cryptology—EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403, Berlin, 1995. Springer-Verlag.

[ST04] B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *Advances in Cryptology—ASIACRYPT '04*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136, Berlin, 2004. Springer-Verlag.

[ST06] B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encryptions. In *Advances in Cryptology—EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537, Berlin, 2006. Springer-Verlag.

[Yao82] A. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.