mathematics

MDPI

Article

Lightweight Secure Integer Comparison

Thijs Veugen [1,2]

[1] TNO, Unit ICT, 2509 JE The Hague, The Netherlands; thijs.veugen@tno.nl
[2] Centrum Wiskunde & Informatica, Cryptology Department, 1098 XG Amsterdam, The Netherlands

Abstract: We solve the millionaires problem in the semi-trusted model with homomorphic encryption without using intermediate decryptions. This leads to the computationally least expensive solution with homomorphic encryption so far, with a low bandwidth and very low storage complexity. The number of modular multiplications needed is less than the number of modular multiplications needed for one Pallier encryption. The output of the protocol can be either publicly known, encrypted, or secret-shared. The private input of the first player is computationally secure towards the second player, and the private input of the second player is even unconditionally secure towards the first player. We also introduce an efficient client-server solution for the millionaires problem with similar security properties.

Keywords: millionaires problem; homomorphic encryption; quadratic residues; secure integer comparison

check for updates

Citation: Veugen, T. Lightweight Secure Integer Comparison. *Mathematics* **2022**, *10*, 305. https://doi.org/10.3390/math10030305

Academic Editor: Ferucio Laurentiu Tiplea

Received: 10 December 2021
Accepted: 13 January 2022
Published: 19 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the current era of big data, artificial intelligence, and privacy awareness, there is a growing need for securely computing with distributed data. The field of secure multi-party computation is evolving towards maturity, and leads to secure cryptographic solutions for this purpose. In its most general setting, there are multiple parties, each having private inputs, and they want to jointly evaluate a certain function on their inputs, without revealing those inputs. Although these cryptographic solutions achieve a high level of security, the challenge is often to restrict their computational and communication effort.

We focus on the setting of two parties using additively homomorphic encryption, and only one holding the decryption key, with numerous applications in secure signal processing like secure privacy-protected face recognition, privacy-protected k-means clustering, and privacy-protected content recommendation [1]. Other applications are neural network classification for e-health, biometric matching, watermark detection, fingerprinting for DRM, and smartgrids [1].

Within all those applications, there is clear need for a secure comparison protocol as a building block [1]. A few secure protocols based on homomorphic encryption are known for comparing two integers, the so-called millionaires problem. We present a new comparison protocol, which is dedicated to lightweight environments that require little memory and a low computational effort. For this reason, we focus on the semi-trusted model where both players follow the required protocol steps. As far as we know, this protocol is the solution with the lowest number of modular multiplications (of two additively homomorphic encrypted numbers) by now. The main reason is that our protocol does not need intermediate decryptions, which are computationally expensive.

First, the problem definition and notations are explained in the preliminaries. Then, an overview is presented on the most important related work. The actual comparison protocol called LSIC (Lightweight Secure Integer Comparison) is presented in the second section, its correctness and security are proven in Section 3. In Section 4, the computational, communication and storage complexity of our solution is analyzed, counting the total number of multiplications, transmissions, and storage of encryptions. These complexity parameters are compared with the most important alternative solutions in Section 5. In Section 6, it is

shown how any comparison protocol can be transferred to the client-server model in an efficient and secure way. The final section contains the conclusions.

1.1. Preliminaries

We consider two parties, A and B, both having a private integer, respectively, a and b. Both parties would like to know which of their integers is the largest one, without revealing its value to the other party. This problem is called the millionaires problem. Since we are especially interested in lightweight environments, we assume both players are honest but curious.

The output of our secure multiparty computation protocol is the bit t such that $\{t = 1\} \equiv \{a < b\}$. We consider different variants with different types of output:

1. The value t becomes known to both A and B.
2. A obtains the value t encrypted by a public key (homomorphic) cryptosystem, and B holds the private key K.
3. A and B share the output, i.e., they both have a private output bit, respectively, t_A and t_B, such that $t = t_A \oplus t_B$ ($\oplus$ denotes exclusive or).

In the last two variants, the value t is not known to A and B, but it enables them to use the output for other applications. For the public key cryptosystem used in our protocols, any additively homomorphic and semantically secure cryptosystem could be used. In our notation, we consider two classes of encryption systems, namely $[.]$ to denote the encryption of a single bit with, e.g., Goldwasser-Micali (GM) [2,3], and $[\![.]\!]$ to denote encryption of integers with, e.g., Pallier [4] or Damgård, Geisler, and Krøigaard (DGK) [5–7].

Besides the model where both parties know their private integer, we also consider the client–server model. In this model, party A is the client who has both integers, but only in encrypted form, and B is the server, who owns the private key of the cryptosystem. In the client–server model, both parties are not allowed to learn the value of the integers a and b, and the same variants as above apply for obtaining the output t. Such a model is often used in privacy protecting applications where intermediate values should remain unknown, e.g., in face recognition [8].

Let N be the modulus of the encryption system, which is usually equal to the product of two large primes. We recall an important property of homomorphic encryption systems, namely that for bits x and y we have $[x] \cdot [y] = [x \oplus y] \bmod N$, and for integers x and y we have $[\![x]\!] \cdot [\![y]\!] = [\![x + y]\!] \bmod N$. An encrypted integer is negated most efficiently by using the Euclidean algorithm [9]. This is denoted by $[\![-x]\!] \leftarrow [\![x]\!]^{-1} \bmod N$. During the complexity computations, we assume that N is 1024 bits long, and that any security wise comparable symmetric encryption system has a key size of 80 bits [10]. For other key lengths, similar complexity conclusions can be drawn. For convenience, we neglect in our notation that the cipher text size in Pallier is N^2 and use N instead, just like in DGK or RSA.

In DGK and other similar cryptosystems, an integer x is encrypted by computing $[\![x]\!] = g^x \cdot h^r \bmod N$, where g and h are fixed public generators, and r is a secret random number chosen by the encrypting person. In GM, a bit x is encrypted by computing $[x] = g^x \cdot r^2 \bmod N$ where g is a fixed public integer (quadratic nonresidue) of $\log_2 N$ bits, and r is a secret random number of the same size, chosen by the encrypting person. In both cryptosystems, the variable r is used to randomize the outcome of the encryption, so two encryptions of the same value will not be recognized as such. This is also the reason that within a cryptographic protocol, where encryptions are received from the other party, processed within local computations, and subsequently returned, encryptions need to be re-randomized before returning.

We use pseudo-code to describe the protocols. Assertions between $\{.\}$ are used to describe the current value of variables. Additionally, comments are used, prefixed by $\triangleright$, to explain the corresponding line of the protocol. Each statement is prefixed by A or B, indicating the party that performs the statement. e.g., A: $[\![\tau]\!] \leftarrow [\![1]\!] \cdot [\![t]\!] \bmod N$ means that A multiplies the (encrypted) variable $[\![t]\!]$ with an encrypted 1 modulo N, and stores the result in the (encrypted) variable $[\![\tau]\!]$.

To compute the computational complexity of the different protocols, we use the fact that an exponentiation modulo N with an exponent of n bits will on average take $\frac{3}{2}n$ multiplications modulo N. When the factoring of N is known, this can be reduced to $\frac{3}{4}n$ by using the Chinese remainder theorem [9]. Namely, a multiplication modulo N is assumed to be equivalent to four multiplications modulo one of the two prime factors. The effort for negating an encrypted number is considered negligible. A Pallier decryption takes $\frac{3}{2}\log_2 N$ multiplications modulo N [4]. A Pallier encryption of a plaintext of n bits will take $5n + 5\log_2 N$ multiplications modulo N [4].

We use $x \div y$ to denote the integer division of x by y, so $x \div y = (x - x \bmod y)/y$. Let σ be the statistical security parameter, which value is usually chosen around 80. The maximum size of the input variables is denoted by ℓ. We assume all random variables are uniformly chosen.

1.2. Related Work

The first solution of the millionaires problem is by Yao [11] in 1982. His solution, based on garbled circuits, has been improved many times since.

Most solutions are either based on homomorphic encryption or on garbled circuits. In Section 5, we describe one of the best candidates for both categories, and compare it with our solution LSIC. The candidate that uses homomorphic encryption is by Damgård, Geisler, and Krøigaard [5–7], who use a dedicated cryptosystem finetuned for small plaintext values. Their protocol is described in detail in Section 5.2 and compared to our work. One of the most efficient implementations nowadays based on garbled circuits is described by Kolesnikov, Sadeghi and Schneider [12]. The general garbled circuit approach is described in Section 5.3, and their specific implementation is compared to our work.

Other related work is, e.g., by Fischlin [13], who describes a system that enables to compute the product (AND) of two quadratic residues. However, an error parameter λ is required to guarantee the correctness of the result, which increases the computational and communication load, including ℓ decryptions.

Garay, Schoenmakers and Villegas [14] describe a nice solution for the client-server setting in the multi-party case, but since they use the malicious adversary model instead of the honest-but-curious model, their solutions are less efficient.

More recent results focus on the malicious adversary model [15], or other techniques, such as fully homomorphic encryption [16], which reduces communication, but increases computational efforts, or oblivious transfers [17].

Kerschbaum and Terzidis (KT) present an efficient solution to the millionaires problem in the semi-trusted model in [18], as described in detail in Section 5.1. This solution is later extended to multiple parties by Kerschbaum, Biswas and de Hoogh [19].

2. Comparison Protocol

Suppose party A has a private unencrypted number a, and party B has a private and unencrypted number b. The integers a and b have size ℓ. We denote their bits by a_i and b_i, for $0 \le i < \ell$, where a_0 and b_0 are the least significant bits. We use the notation a^l ($1 \le l \le \ell$) to denote the integer $\sum_{i=0}^{l-1} a_i 2^i$, i.e., the first l bits of a, and the same for b. Note that $a = a^\ell$ and $b = b^\ell$.

The idea behind our comparison protocol [20] is to compute the bits t_i, from $i = 1$ towards $i = \ell$, where $(t_i = 1) \equiv (a^i < b^i)$. The bit t_{i+1} can be computed from t_i by using the relation:

$$(t_{i+1} = 1) \equiv (a_i < b_i) \text{ or } \{(a_i = b_i) \text{ and } (t_i = 1)\} \quad (1)$$

The correctness of this recurrence relation is easily seen by observing that a_i and b_i are the most significant bits of a^{i+1} and b^{i+1}, respectively.

In order to compute $t = t_\ell$, we propose the protocol Lightweight Secure Integer Comparison (LSIC), which is shown in Algorithm 1. Algorithm 1 contains assertions between $\{.\}$ to describe the current value of variables. Additionally, comments are used, prefixed by $\triangleright$, to explain the protocol. The correctness of the protocol, i.e., the proof of the

assertions, is shown in Section 3.1. The formal security proof, to show that A and B learned nothing more than $a < b$, is given in Section 3.2.

Algorithm 1 Lightweight secure integer comparison (LSIC).

Input A	$a = a_{\ell-1} \dots a_0$
Input B	$b = b_{\ell-1} \dots b_0$ and the decryption key K
Joined output	bit t, where $(t = 1) \equiv (a < b)$

Party B encrypts and randomizes b_0 and sends $[b_0]$ to A
if $a_0 = 0$ **then**
  A: $[t] \leftarrow [b_0]$
else
5: A: $[t] \leftarrow [0]$ ▷ $[t] \leftarrow 1$, randomized in line 17
end if
$\{(t = 1) \equiv (t_1 = 1) \equiv (a_0 < b_0)\}$
for $i \leftarrow 1, \dots, \ell - 1$ **do** ▷ A computes $t = t_{i+1}$ from $t = t_i$
  $\{t = t_i\}$
10: A blinds $t = t_i$ by tossing a fair coin $c \in \{0, 1\}$
  if $c = 0$ **then**
    A: $[\tau] \leftarrow [t]$
  else
    A: $[\tau] \leftarrow [1] \cdot [t] \bmod N$ ▷ $\tau \leftarrow 1 \oplus t$
15: **end if**
  $\{\tau = c \oplus t_i\}$
  A randomizes $[\tau]$ and sends it to B
  if $b_i = 0$ **then**
    B: $[tb] \leftarrow [0]$ ▷ $[tb] \leftarrow 1$, randomized in line 24
20: **else**
    B: $[tb] \leftarrow [\tau]$
  end if ▷ B computed $tb = \tau \cdot b_i$ without decrypting $[\tau]$
  $\{tb = \tau \cdot b_i\}$
  B encrypts b_i, randomizes $[tb]$, and sends both $[tb]$ and $[b_i]$ to A
25: $\{tb = (c \oplus t_i) \cdot b_i\}$
  if $a_i = c$ **then** ▷ A smartly unblinds tb
    A: $[tb] \leftarrow [tb] \cdot [b_i] \bmod N$ ▷ $tb \leftarrow tb \oplus b_i$
  end if
  $\{tb = (1 \oplus a_i \oplus t_i) \cdot b_i\}$
30: **if** $a_i = 0$ **then**
    $\{tb = (1 \oplus t_i) \cdot b_i\}$
    A: $[t] \leftarrow [t] \cdot [tb] \bmod N$ ▷ $t \leftarrow t \oplus tb$
  else
    $\{tb = t_i \cdot b_i\}$
35: A: $[t] \leftarrow [tb]$
  end if
  $\{t = t_{i+1}\}$
end for
$\{t = t_\ell\}$
40: A sends $[t]$ to B so B can decrypt it and send $t = t_\ell$ back to A

Party A and B encrypt single bits by $[.]$, but only party B can decrypt. The main idea is that A uses variable $[t]$, which is the encryption of t_i, and computes, in a joined protocol with B, $[t_\ell]$. This computation is done recursively, starting with $[t_1]$, $[t_2]$ until $[t_\ell]$. In order for A to compute the next value, B sends the encrypted bits $[b_i]$ in line 24, but since this is not enough for A, as he is computing in the encrypted domain, B also sends the encryption of tb, being the product of b_i and t_i. To compute the product tb, A sends a blinded version

of t_i to B in line 17, because each intermediate value t_i should be unknown to B (and A). The product is smartly unblinded again by A in line 27. Although the computations by A in lines 27 and 32 are not obvious, in Section 3.1, it is shown that A indeed correctly computes the next value $[t_{i+1}]$ from $[t_i]$, a_i, $[b_i]$, and $[b_i \cdot t_i]$. The computation in line 27 is actually an efficient combination of two similar statements:

$\{tb = (c \oplus t_i) \cdot b_i\}$
if $c = 1$ **then** ▷ A unblinds tb
  A: $[tb] \leftarrow [tb] \cdot [b_i] \bmod N$ ▷ $tb \leftarrow tb \oplus b_i$
end if
$\{tb = t_i \cdot b_i\}$
if $a_i = 0$ **then**
  A: $[tb] \leftarrow [tb] \cdot [b_i] \bmod N$ ▷ $tb \leftarrow tb \oplus b_i$
end if
$\{tb = (1 \oplus a_i \oplus t_i) \cdot b_i\}$

Since each statement is the inverse of the other one (as proved in Section 3.1), a double execution should be avoided. Only when $c \neq 1 \oplus a_i$, i.e., $a_i = c$, one execution is necessary.

This protocol works for any number of bits ℓ. Since the protocol doesn't require intermediate decryptions, the computational complexity is low. Algorithm 1 computes $a < b$, but also $a \leq b$ is similarly computed by letting A compute $a \leftarrow 2^\ell - a$ and B: $b \leftarrow 2^\ell - b$ in the beginning, and flipping the value of t at the end by B: $t \leftarrow 1 \oplus t$. The encryption system should be homomorphic and semantically secure. We use GM, because encryption and re-randomization are easy, but one could also use, e.g., Pallier or DGK. This requires a small modification of the homomorphic operations on encrypted numbers, as depicted in Table 1 below, where $[\![.]\!]$ is used to denote Pallier encryption.

Table 1. Operations and their encrypted counterpart.

Operation	GM	Pallier
14: $\tau \leftarrow 1 \oplus t$	$[\tau] \leftarrow [1] \cdot [t]$	$[\![\tau]\!] \leftarrow [\![1]\!] \cdot [\![t]\!]^{-1}$ $\{\tau \leftarrow 1 - t\}$
27: $tb \leftarrow tb \oplus b_i$	$[tb] \leftarrow [tb] \cdot [b_i]$	$[\![tb]\!] \leftarrow [\![b_i]\!] \cdot [\![tb]\!]^{-1}$ $\{tb \leftarrow b_i - tb\}$
32: $t \leftarrow t \oplus tb$	$[t] \leftarrow [t] \cdot [tb]$	$[\![t]\!] \leftarrow [\![t]\!] \cdot [\![tb]\!]$ $\{t \leftarrow t + tb\}$

The modified operation in line 27 works, because $tb = \tau \cdot b_i$, so $b_i \geq tb$. The modified operation in line 32 works, because in this case ($a_i = 0$), $tb = (1 \oplus t_i) \cdot b_i$, so whenever $t = t_i = 1$, we have $tb = 0$. Therefore, the addition of t and tb cannot exceed 1.

When the output $t = t_\ell$ has to remain secret to both players, obviously the final step of the protocol at line 40 can be skipped. This also saves the (only) costly decryption by B. When the output has to be secretly shared among both players, the final step has to be modified slightly:

A blinds $t = t_\ell$ by tossing a fair coin $c \in \{0, 1\}$
if $c = 0$ **then**
  A: $[\tau] \leftarrow [t]$
else
  A: $[\tau] \leftarrow [1] \cdot [t] \bmod N$ ▷ $\tau \leftarrow 1 \oplus t$
end if
$\{\tau = c \oplus t\}$
A: $t_A \leftarrow c$
A sends $[\tau]$ to B
B decrypts $[\tau]$
B: $t_B \leftarrow \tau$
$\{t = t_A \oplus t_B\}$

All homomorphic systems consist of an encryption and a randomization part, e.g., in GM $[x] = g^x \cdot r^2 \bmod N$, where g is a fixed integer (quadratic nonresidue) and r is

randomly chosen. When implementing our comparison protocol, the randomization part can be skipped in most computations. Only when a value has to be sent to the other party (in lines 1, 17, 24, and 40), the result should be re-randomized.

3. Correctness and Security

3.1. Correctness

The number $t = t_i$ is blinded by A in line 14 by negating the bit with probability $\frac{1}{2}$, in which case $\tau = 1 \oplus t_i$ (otherwise, $\tau = t_i$). From τ, B computes $tb = \tau \cdot b_i$. After receiving tb, A computes either $tb = t_i \cdot b_i$ or $tb = (1 \oplus t_i) \cdot b_i$ depending on the value of a_i. To see that the computation in line 27 has been done correctly, observe that the value of tb is toggled from $t_i \cdot b_i$ to $(1 \oplus t_i) \cdot b_i$ and back by computing $tb \leftarrow tb \oplus b_i$:

$$\begin{array}{rrcl} & ((1 \oplus t_i) \cdot b_i) \oplus b_i & = & \\ \left\{\begin{array}{ll} t_i = 0: & b_i \oplus b_i \\ t_i = 1: & 0 \oplus b_i \end{array}\right. & \begin{array}{c} = \\ = \end{array} & \begin{array}{c} 0 \\ b_i \end{array} & = \\ & t_i \cdot b_i & & \end{array}$$

The converse follows by substituting t_i for $1 \oplus t_i$ and vice versa. The correctness of the computation of t_{i+1} (describing the relation $a^{i+1} < b^{i+1}$) from t_i in lines 32 and 35 follows from Table 2 below.

Table 2. Correctness of computing t_{i+1}.

t_i	0	1	0	1	0	1	0	1
a_i	0	0	0	0	1	1	1	1
b_i	0	0	1	1	0	0	1	1
t_{i+1}	0	1	1	1	0	0	0	1
$tb = t_i \cdot b_i$					0	0	0	1
$tb = (1 \oplus t_i) \cdot b_i$	0	0	1	0				
$t_i \oplus tb$	0	1	1	1				

The first three rows show the eight possible values of the triplet (t_i, a_i, b_i). The last four rows are computed from the first three. For computing the row t_{i+1}, the recurrence relation of Equation (1) is used. From the table follows that, for each value of the triplet (t_i, a_i, b_i), $t_{i+1} = t_i \oplus tb$ when $a_i = 0$, and $t_{i+1} = tb$ otherwise.

It follows that in each iteration, the value of t_{i+1} is correctly computed from the previous one, and thus that the decrypted value t at the end of the comparison protocol indeed equals the bit $t = t_\ell$, and therefore $\{t = 1\} \equiv \{a < b\}$.

3.2. Security

In order to show that our protocol privately computes the comparison of two integers in the semi-honest model, we have to show that whatever can be computed by A or B from their view of a protocol execution, can be computed from their input and the comparison result (see Definition 7.2.1 in Goldreich [21]).

The view of A consists of its private number a, the size ℓ, the comparison bit t, the internal coin tosses c_i, $1 \le i < \ell$, and all intermediate messages received from B: the encrypted bits $[b_i]$, $0 \le i < \ell$, which are the encrypted bits of the number b, and the encrypted bits $[tb_i]$, $1 \le i < \ell$, which equal $[b_i \cdot \tau_i]$, τ_i being the blinded version of the bit $(a^i < b^i)$. Summarizing, the view of A equals

$$V_A = (a, \ell, t, c_1 \dots c_{\ell-1}, [b_0] \dots [b_{\ell-1}], [tb_1], \dots [tb_{\ell-1}])$$

It suffices to show that there exists a probabilistic polynomial-time algorithm S_A such that $S_A(a, \ell, t)$ is computationally indistinguishable from V_A [21]. Since the encryption algorithm is semantically secure, every pair of encryptions is computationally indistinguishable, so by letting S_A randomly generate $2\ell - 1$ encryptions and $\ell - 1$ coin tosses, this condition is easily verified.

The view of B consists of its private number b, the decryption key K, the size ℓ, the comparison bit t, and all intermediate messages received from A: the blinded bits $[\tau_i]$, $1 \leq i < \ell$, τ_i being the blinded version of the bit $(a^i < b^i)$. Note that B also receives $[t]$ in the end, but since this number doesn't add anything to the view, we leave it out for convenience. Since B owns the decryption key, all encrypted values $[\tau_i]$ can be decrypted to τ_i. Additionally, B is able to deduce the randomization part of each encryption, but since A carefully uses re-randomization before each transmission, this information can be considered as a random variable and is therefore useless to B. Summarizing, the view of B is equivalent to

$$V_B = (b, K, \ell, t, \tau_1, \ldots \tau_{\ell-1})$$

Again, we have to show that there exists a probabilistic polynomial-time algorithm S_B such that $S_B(b, K, \ell, t)$ is computationally indistinguishable from V_B. This is easily satisfied by letting S_B randomly generate $\ell - 1$ bits. The derivation in Equation (2) shows that the bits τ_i are indeed uniformly distributed.

In fact, we can even prove a much stronger assertion, namely that V_B yields no more information regarding a than its inputs and output (b, k, ℓ, t) do, implying perfect (or information-theoretic) security for A towards B. This is due to the blinding technique of the numbers τ:

(i) The bit t contains the value $t_i = (a^i < b^i)$.
(ii) A tosses a fair coin c, when head then $\tau \leftarrow 1 \oplus t$ else $\tau \leftarrow t$.
(iii) A sends τ to B.

To see that $I(b, K, \ell, t; a) = I(V_B; a)$, where I denotes the mutual information, we show that each τ is uniformly distributed.

$$\begin{aligned} \Pr(\tau = 1) &= \\ \Pr(\tau = 1 c = \text{head}) \cdot \Pr(c = \text{head}) + \Pr(\tau = 1 c = \text{tails}) \cdot \Pr(c = \text{tails}) &= \\ \Pr(t = 0) \cdot \frac{1}{2} + \Pr(t = 1) \cdot \frac{1}{2} &= \\ \frac{1}{2} & \end{aligned} \tag{2}$$

It's easy to see that the random variable τ (and its uniform distribution) is independent of a, b, K, ℓ, t, or any previous value of τ, so the equality of mutual information follows.

We conclude that the private input of B is computationally secure towards A, and that the private input of A is unconditionally (so even with unbounded computation power) secure towards B.

4. Complexity

In this section, we compute the computational, communication, and storage complexity of our protocol. We assume that during the protocol, the randomization part r^2 is omitted where possible. In fact, only when an encrypted value is transmitted to the other party, this encrypted value has to be rerandomized, i.e., multiplied by a random square, such that two pairs of encrypted bits are indistinguishable. Note that although B owns the decryption key (also called private key), re-randomization is even necessary for A, because B might be able to recognize the randomization part.

The summary of the complexity analysis of our comparison protocol is depicted in Table 3. We ignore the efforts for key generation and key distribution.

Table 3. Complexity overview.

	Computation		Communication	Storage
	Average	**Maximal**		
A	$3.5(\ell-1)+2$	$4(\ell-1)+2$	$\ell-1$	3
B	$3(\ell-1)+1.5$	$4(\ell-1)+2$	$2\ell-1$	2
Decryption	$\frac{3}{8}\log_2 N$		1	0
Total with encrypted $[t]$	$6.5(\ell-1)+3.5$	$8(\ell-1)+4$	$3\ell-2$	5
Total with decrypted t	$6.5(\ell-1)+3.5+\frac{3}{8}\log_2 N$	$8(\ell-1)+4+\frac{3}{8}\log_2 N$	$3\ell-1$	5

The computational complexity is measured in the number of multiplications modulo N, the communication complexity is measured in the number of messages. Each message equals an encrypted bit and has a size of $\log_2 N$ bits. The storage complexity is measured in the number of encrypted numbers (of size $\log_2 N$ bits) to be stored. When the comparison result has to be known to both parties, we need an extra decryption of the comparison result by B, as depicted in the table above. If the output bit should be available in encrypted form, no decryptions are required. The unencrypted message of one bit (the comparison result) from B to A is neglected in our analysis.

An important observation is that the (maximal) number of multiplications modulo N is even less than the number of modular multiplications needed for the Pallier encryption of an ℓ ($\ell < N$) bit number: $5l + 5\log_2 N$ [4].

4.1. Computational Complexity

We compute the computational complexity by counting the number of multiplications modulo N, since these form the main computational load. Due to the construction of GM, the encryption of 0 requires one multiplication (squaring) modulo N, and the encryption of 1 requires two multiplications modulo N. A decryption of $[x]$ to x requires the computation of the Jacobi symbol $(\frac{[x]}{N})$, which equals the product of the Legendre symbols $(\frac{[x]}{p})$ and $(\frac{[x]}{q})$. The Legendre symbol $(\frac{[x]}{p})$ is equivalent to $[x]^{(p-1)/2}$ modulo p, and therefore requires approximately $\frac{3}{2}\log_2 p$ multiplications modulo p. A multiplication modulo N is more intensive, and requires more or less four multiplications modulo p (or q), such that the total number of multiplications modulo N for one GM decryption is estimated by $\frac{3}{2}\cdot\frac{1}{4}\log_2 p + \frac{3}{2}\cdot\frac{1}{4}\log_2 q = \frac{3}{8}\log_2 N$.

The expected number of multiplications (in the encrypted domain) for A in this protocol equals $\ell-1$ times: 0.5 for the blinding of t_i in line 14, 2 for the re-randomization of $[\tau]$ in line 17, 0.5 for the possible conversion of $[tb]$ in line 27, and 0.5 for the computation of $[t_{i+1}]$ in line 32. Together with 2 for the final re-randomization of $[t]$ in line 40, for a total of $3.5(\ell-1)+2$ multiplications. The expected computational load for B equals the ℓ encryptions of b_i in lines 1 and 24, and $\ell-1$ re-randomizations of $[tb]$ in line 24, for a total of $1.5\ell + 1.5(\ell-1) = 3(\ell-1)+1.5$ multiplications.

The actual number of multiplications depends on the (bit) values of a and b. Since timing attacks might reveal some information about these numbers, we also mention the maximal number of multiplications, which are computed similarly and equal $4(\ell-1)+2$ for both A and B. This maximum for B is achieved when each bit b_i of b is zero, because then the encryption of b_i will take 2 multiplications, as well as the randomization of $[tb]$. For A, we have to take a closer look at lines 14, 27 and 32 and their conditions $c=1$, $a_i=c$ and $a_i=0$, respectively. One can see that at most two of these three conditions will hold for each i in the for loop. This will be the case when either $c=1$ or $a_i=0$.

No intermediate decryptions are needed in the protocol. Only when the end result $[t]$ has to be available in plain text, or a conversion to another encryption system is needed (see e.g., the client–server solution in Section 6), one decryption by B is desired at the end which costs approximately $\frac{3}{8}\log_2 N$ multiplications modulo N.

Some values can be precomputed to reduce the number of multiplications, but this requires more storage capacity. All encryptions and all random parts (the squarings) can be precomputed and stored.

4.2. Communication Complexity

The numbers that are sent in our protocol are all single bits encrypted by the GM system, resulting in encrypted numbers of $\log_2 N$ bits.

A sends to B $\ell - 1$ times the value of $[\tau]$, and once the number $[t]$, for a total of ℓ numbers of $\log_2 N$ bits. B sends to A the number $[b_0]$, and $\ell - 1$ times the numbers $[b_i]$ and $[tb]$, for a total of $2\ell - 1$ numbers of $\log_2 N$ bits. Our protocol takes ℓ communication rounds (plus half a round in the first step).

4.3. Storage Complexity

We count the number of encrypted values that have to be stored, since plain integers are relatively small.

A has to store the current values of $[b_i]$, $[t_i]$ and $[tb]$, requiring three storage units. When $[\tau]$ is computed, the storage unit of $[tb]$ can be used so this doesn't require an extra storage unit. B has to store $[tb]$ and $[b_i]$, requiring two storage units. When $[\tau]$ is received, this can be stored in the storage unit of $[tb]$ avoiding an extra storage unit.

The storage complexity expands when using precomputations to store encryptions of known bits, or some random squares used for re-randomization. The total number of storage units will depend on the implementation and the requirements with respect to waiting time, communication, computation and storage capacities.

5. Alternative Solutions

In this section, we compare our comparison protocol, the Lightweight Secure Integer Comparison (LSIC), with other solutions for solving the millionaires problem. Since we are only interested in the most efficient solutions, we restrict ourselves to the semi-honest model. In the literature, we find two main classes of solutions that use public key cryptography, namely one based on homomorphic encryptions, and one based on garbled circuits. We compare our solution to the best representatives in each category.

5.1. KT

Kerschbaum and Terzidis (KT) present an efficient solution to the millionaires problem in the semi-trusted model in [18]. Their cryptographic protocol is depicted in Algorithm 2. Party A computes the Pallier encrypted difference $x = b - a$ and lets B decide whether $x \geq 0$. They use the upper half $[\frac{N+1}{2}, N)$ of the plain text range to represent negative numbers. Since B is not allowed to learn x, A uses multiplicative blinding (which preserves the sign of x) in line 5 to hide it for B. To prevent x from exceeding $\frac{N+1}{2}$, ℓ is bounded by $3\ell + 1 < \frac{N+1}{2}$. The main disadvantage is that multiplicative hiding is not perfect and leaks some information about x, and thus a, to B [22].

We compute the computational complexity of KT to compare it with our solution LSIC. In line 2, b is encrypted by Pallier, a number of ℓ bits. In line 5 a number of 3ℓ bits is encrypted, and an exponentiation is computed to the power r which has 2ℓ bits, and in line 8, the number x is decrypted. Therefore, the total number of multiplications modulo N is $(5\ell + 5\log_2 N) + (5 \cdot 3\ell + 5\log_2 N) + \frac{3}{2} \cdot 2\ell + \frac{3}{2}\log_2 N = 23\ell + \frac{23}{2}\log_2 N$. This computational complexity is better than the other alternatives, but worse than LSIC. Although the communication complexity of KT is very good, its weaker notion of security is a serious drawback.

Algorithm 2 Kerschbaum and Terzidis (KT).

Input A	integer a
Input B	integer b and the decryption key K
Joined output	bit t, where $(t = 1) \equiv (a \leq b)$

{Both a and b consist of ℓ bits}
B encrypts b and sends $[\![b]\!]$ to A
A chooses random number r of 2ℓ bits
A chooses random number r' such that $0 \leq r' < r$
A: $[\![x]\!] \leftarrow [\![b]\!]^r \cdot [\![r \cdot a - r']\!]^{-1} \bmod N$ ▷ $x \leftarrow r \cdot (b - a) + r' \bmod N$
$\{(a \leq b) \equiv (x < \frac{N+1}{2})\}$
A sends $[\![x]\!]$ to B
B decrypts $[\![x]\!]$
if $x < \frac{N+1}{2}$ **then**
    B: $t \leftarrow 1$
else
    B: $t \leftarrow 0$
end if
$\{(t = 1) \equiv (x < \frac{N+1}{2})\}$
B sends t to A
$\{(t = 1) \equiv (a \leq b)\}$

5.2. DGK

The DGK (Damgård, Geisler, and Krøigaard) protocol [5–7], which is actually inspired by Blake and Kolesnikov [23] is depicted in Algorithm 3.

Algorithm 3 The Damgård, Geisler, and Krøigaard (DGK) comparison protocol.

Input A	$a = a_{\ell-1} \ldots a_0$
Input B	$b = b_{\ell-1} \ldots b_0$ and the decryption key K
Joined output	bit t, where $(t = 1) \equiv (a < b)$

Party B encrypts the bits b_i, $0 \leq i < \ell$ and sends all $[\![b_i]\!]$ to A.
for all $0 \leq i < \ell$ **do** ▷ A computes the bitwise exclusive or's
    if $a_i = 0$ **then**
        A: $[\![x_i]\!] \leftarrow [\![b_i]\!]$
    else
        A: $[\![x_i]\!] \leftarrow [\![1]\!] \cdot [\![b_i]\!]^{-1}$ ▷ $x_i \leftarrow 1 - b_i$
    end if
    $\{x_i = a_i \oplus b_i\}$
end for
for all $0 \leq i < \ell$ **do**
    A encrypts $a_i + 1$
    A: $[\![c_i]\!] \leftarrow [\![a_i + 1]\!] \cdot [\![b_i]\!]^{-1} \cdot \prod_{j=i+1}^{\ell-1} [\![x_j]\!] \bmod N$
    $\{c_i = a_i + 1 - b_i + \sum_{j=i+1}^{\ell-1} a_j \oplus b_j\}$
    A blinds c_i towards B by raising $[\![c_i]\!]$ to a random non-zero number
end for
A randomly permutes the numbers $[\![c_i]\!]$ and sends them to B
B checks whether one of them is zero, and returns the result t to A
$\{(t = 1) \equiv (\text{there is an } i \text{ such that } c_i = 0) \equiv (a < b)\}$

The main idea of DGK is to search, from left to right, for the first position i where the bits of a and b differ. When $a_i < b_i$, as indicated by $c_i = 0$, then $a < b$. All other

numbers c_j, $j \neq i$ will be positive. The values and order of the numbers c_i have to be blinded because they reveal some information about a towards B. They use a special homomorphic encryption system that is finetuned to small plaintext sizes u and enables efficiently checking whether encrypted values are zero [6].

There is an important computational improvement in Algorithm 3, which is not mentioned by Damgård, Geisler, and Krøigaard, based on the observation that the number c_i can only be zero when $a_i = 0$. This implies that A only has to compute the numbers $[\![c_i]\!]$ for which $a_i = 0$. For the other numbers $[\![c_i]\!]$, an encrypted, non-zero random number can be chosen. This will save some modular multiplications for A.

The computational complexity of the DGK protocol is analyzed as follows:

(i) The bitwise exclusive or's have to be computed for the bits of a and b. This takes on average $\frac{1}{2}\ell$ multiplications.

(ii) From the exclusive or's, the ℓ numbers c_i have to be computed. This can be done in 3ℓ multiplications, by storing the intermediate result of $\prod_{j=i+1}^{\ell-1} [\![x_j]\!]$.

(iii) The numbers c_i have to be blinded, i.e., raised to a random number of length u [6]. This step can be done efficiently due to the smartly chosen encryption system. The number u is relatively small and equal to the first prime larger than $\ell + 2$. The blinding takes around $\frac{3}{2}\ell \cdot \log_2 u$ multiplications.

These estimates are based on the idea that in case one has to raise a number to an exponent of n bits, this will take on average $\frac{3}{2}n$ multiplications.

When taking u, the plaintext size, equal to $\ell + 2$, we come up with a total of $\frac{7}{2}\ell + \frac{3}{2}\ell \log_2(\ell + 2)$ multiplications, which is of order $\ell \log_2 \ell$ (due to the blinding of the numbers c_i).

These are all multiplications on the account of A. For B, the main computational load is in decrypting the received numbers. B receives the ℓ blinded numbers c_i and has to decrypt them to decide whether one of them is zero or not. Therefore, a full decryption is not required, only a check whether the encrypted value is zero or not. This is relatively easy in the DGK encryption system, and is equivalent to raising each c_i to the power v, which is a number of size $t = 160$ [6]. By using the factorization of N while decrypting, the total number of multiplications for B is about $\frac{1}{2} \cdot \frac{3}{2} \cdot \ell \cdot t = 120\ell$. Unless ℓ is very large, the decryptions (by B) determine the main computational load of the DGK protocol.

The encryptions by A and B are easy, except for the randomization part, which is done when sending the encrypted value to the other party. (Re)randomizations in the DGK encryption system are roughly equivalent to raising a number to an exponent of size $t = 160$ bits [6], so we estimate each randomization by $\frac{3}{2}t$ multiplications modulo N. Each party has to (re)randomize ℓ numbers.

In the DGK protocol, A sends to B the ℓ numbers $[\![c_i]\!]$, and B sends to A the ℓ numbers $[\![b_i]\!]$, and the final result t (which is only one bit). So the communication load from B to A is ℓ instead of our $2\ell - 1$. More importantly, the DGK protocol takes only one (and a half) round.

In the DGK protocol, A has to store the ℓ numbers $[\![c_i]\!]$, and $[\![b_i]\!]$, for a total of 2ℓ storage units. Party B has to store the ℓ numbers $[\![c_i]\!]$ before they can be decrypted, for a total of ℓ storage units. The storage complexity can be reduced by increasing the number of communication rounds, but it can not be less than linear in ℓ, because the order of the ℓ numbers $[\![c_i]\!]$ has to be randomized towards B.

The DGK protocol offers perfect (unconditional) security for B towards A, because A only receives blinded values, and computational security for A towards B, protected by the semantically secure encryption system.

5.3. Garbled Circuits

The millionaires problem could also be solved by using some form of garbled circuits. The main components of such a solution are:

(1) A creates a garbled circuit for comparing two ℓ bits numbers and sends it to B. The private inputs of A are incorporated by using only the corresponding input wires.
(2) For each input bit of B, A and B perform an (1 out of 2) oblivious transfer protocol so B can use the correct input wire of the garbled circuit.
(3) B evaluates the garbled circuit, which results in one output wire.
(4) B sends (a part of) the output wire to A, which translates this to the result of the comparison.

The computationally most intensive step is the oblivious transfer of the input bits of B (step 2), since this involves asymmetric cryptography, while evaluation of the circuit can be efficiently done with symmetric techniques. The most efficient implementations of step 2 are based on Elliptic Curve Cryptography (ECC), in which case at least one ECC encryption and decryption is needed per input bit. In [24], it is estimated that one ECC encryption plus decryption is comparable to 200 multiplications modulo an 1024 bit number, when considering a security level similar to an RSA number of 1024 bits, which in ECC corresponds with a 160 bits modulus.

It must be noted that the computational complexity of GC can be considerably reduced by using precomputations, but this is considered out of scope for the environments we are interested in.

The communication complexity of one of the best known GC solutions [12] is $19\ell \cdot t$, where t is the usual key size of symmetrical cryptosystems (we use $t = 80$). The circuit here is also based on our recurrence relation that is depicted in Equation (1).

The storage complexity of GC is more or less equal to the communication complexity, since the communicated garbled circuit, and the obliviously transferred values have to be stored separately. However, when using more communication rounds, where in each round the oblivious transfer values and the garbled circuit part with respect to one bit is communicated, the storage complexity could be reduced to approximately $19 \cdot t \cdot 2$, i.e., $19 \cdot t$ bits per party.

Although other variants of garbled circuits exist, the most efficient implementation considered here works in the semi-honest model, and offers computational security for both parties. Furthermore, the solution considered here uses a weak form of Random Oracle, namely of correlation-robust functions [12].

5.4. Summary

A rough summary of the previous subsections is depicted in Table 4.

Table 4. Complexity analysis of related work.

	LSIC	**KT**	**DGK**	**GC**
Computation	$\frac{3}{2}\ell + 384$	$23\ell + 11{,}776$	$(120 + \frac{7}{2}) \cdot \ell + \frac{3}{2}\ell \cdot \log_2(\ell + 2)$	$200 \cdot \ell$
Re-randomization	$5 \cdot \ell$	-	$480 \cdot \ell$	-
Communication	$3072 \cdot \ell$	2048	$2048 \cdot \ell$	$1520 \cdot \ell$
Rounds	ℓ	1	1	1
Storage	5120	2048	$3072 \cdot \ell$	$1520 \cdot \ell$
Security for A	perfect	weak	perfect	comp.
Security for B	comp.	comp.	comp.	comp.

The size of the asymmetric modulus (used in LSIC, KT, and DGK) was chosen as 1024 bits, 160 bits for the elliptic curves used in the oblivious transfer part of GC, and 80 bits for the symmetric systems used in the circuit evaluation. The amount of computation is measured as the number of multiplications modulo a 1024 bit number. The amount of communication and storage is in bits. The computational load of GC is a lower bound,

since only one ECC encryption and decryption (per input bit) is counted, which is needed in the oblivious transfer part.

The comparison of the computational complexity of the four solutions is visualized in Figure 1. The amount of multiplications needed for (re)randomization is incorporated here too.

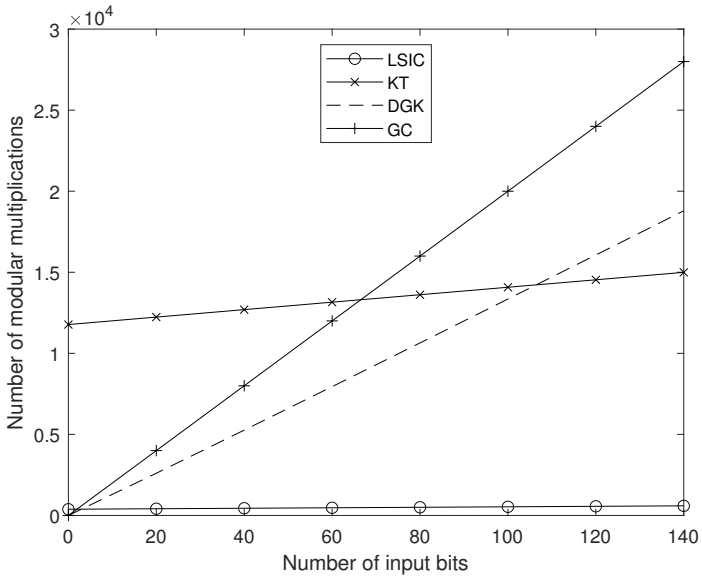

Figure 1. Computational complexity.

When comparing LSIC with other solutions, it's clear that both computational and storage complexity is much smaller than for existing solutions, while the communication complexity is slightly larger. The large computational effort of the Paillier decryption during KT can be clearly seen in Figure 1, and shows our motivation for finding a solution like LSIC that does not require one.

Compared to KT, DGK, and GC, which are constant round solutions, LSIC uses a linear (in ℓ) number of communication rounds. Although extra rounds could lead to delays between rounds depending on the implementation, it offers a significant reduction in storage complexity.

6. Client-Server Model

The LSIC protocol could, like most comparison protocols, also be used in the client–server model. Although we focus on LSIC, the same approach could be used for converting any comparison protocol to the client–server model. We also discuss some variations of the client–server protocol, as depicted in Algorithm 4, making it suitable for different types of output.

Assume A has two encrypted numbers $[\![a]\!]$ and $[\![b]\!]$ of ℓ bits, party B has the private key, and they want to compare the numbers a and b. The actual values of a and b are not known to A and B. Note that $[\![.]\!]$ is used to denote the encryption scheme (e.g., Pallier), which differs from GM, because a and b consist of more than one bit.

The main idea of Algorithm 4 is that the most significant bit of $x = b + 2^\ell - a$ indicates whether $a \leq b$. Since x is an $\ell + 1$ bit number, its most significant bit equals $x \div 2^\ell$. In line 2, the number x is statistically blinded by the random number r, which should contain σ more bits than x. Since we don't allow carry-overs modulo N when computing x, this protocol only works whenever $\ell + 1 + \sigma < \log_2 N$. In line 9 of Algorithm 4 we use the LSIC protocol to compute $c < d$. Since its output t should remain unknown to A and B, to protect the privacy of a and b, we use the output variant where A obtains the encrypted value $[t]$. The correctness of Algorithm 4, i.e., the proof of the assertions, and the formal security proof, is given in later subsections. Algorithm 4 computes $a \leq b$, but also $a < b$ could be similarly

computed by swapping a and b in the computation of x and flipping t at the end, which is most easily done by flipping $r_{\ell+1}$ (by A) or $z_{\ell+1}$ (by B) before encrypting it.

Algorithm 4 Client–server comparison.

Input A	$[\![a]\!]$ and $[\![b]\!]$
Input B	the decryption key K
Output A	encrypted bit $[t]$, where $(t = 1) \equiv (a \le b)$

{Both a and b consist of ℓ bits}
A: $[\![x]\!] \leftarrow [\![b]\!] \cdot [\![2^\ell]\!] \cdot [\![a]\!]^{-1} \bmod N$ ▷ $x \leftarrow b + 2^\ell - a$
A chooses a random number r of $\ell + 1 + \sigma$ bits for blinding x
A: $[\![z]\!] \leftarrow [\![x]\!] \cdot [\![r]\!] \bmod N$ ▷ $z \leftarrow x + r$
5: A sends $[\![z]\!]$ to B
B decrypts $[\![z]\!]$
A: $c \leftarrow 2^\ell - r \bmod 2^\ell$
B: $d \leftarrow 2^\ell - z \bmod 2^\ell$
A: $[t] \leftarrow \mathrm{LSIC}(c, d)$
10: $\{(t = 1) \equiv (c < d)\}$
B encrypts $z_{\ell+1}$ and sends $[z_{\ell+1}]$ to A ▷ $z_{\ell+1}$ is the $(\ell+1)$-th bit of z
A encrypts $r_{\ell+1}$ ▷ $r_{\ell+1}$ is the $(\ell+1)$-th bit of r
A: $[t] \leftarrow [z_{\ell+1}] \cdot [r_{\ell+1}] \cdot [t] \bmod N$ ▷ $t \leftarrow z_{\ell+1} \oplus r_{\ell+1} \oplus t$
$\{t = x \div 2^\ell\}$

This client–server protocol only needs one decryption, namely the Pallier decryption in line 6. In addition to LSIC, which is executed in line 9, only five extra multiplications modulo N are needed (and two GM encryptions), so this client–server protocol has a very low overall computational complexity. In other known solutions, like the client–server protocol by Erkin et al. in [8], the number $x \div 2^\ell$ is computed via the number $x \bmod 2^\ell$, which requires a division of an encrypted number by 2^ℓ, and thus an exponentiation to a number of size $\log_2 N$, which requires substantially more modular multiplications.

Note that although the inputs are encrypted by Pallier, the output is encrypted by QR. It's also possible to obtain the output encrypted by Pallier by replacing lines 11 to 13 by the following lines:

B computes $z \div 2^\ell$, encrypts it with Pallier, and sends $[\![z \div 2^\ell]\!]$ to A
A encrypts $r \div 2^\ell$ with Pallier
A: $[\![t]\!] \leftarrow [\![z \div 2^\ell]\!] \cdot [\![r \div 2^\ell]\!]^{-1} \cdot [\![t]\!]^{-1} \bmod N$ ▷ $t \leftarrow (z \div 2^\ell) - (r \div 2^\ell) - t$
$\{t = x \div 2^\ell\}$

The only problem with this solution is that the output $[t]$ of the comparison protocol in line 9 is encrypted by QR, while Pallier encryption is needed in line 13. To overcome this problem, one solution is to run LSIC entirely with Pallier encrypted bits. This introduces some extra computations for the encryptions and re-randomizations. A computationally less intensive solution is to convert the QR encrypted output bit at the end into a Pallier encryption. This requires a modification of the final line, number 40, of Algorithm 1:

{$[t]$ is the QR encrypted bit t_ℓ}
A blinds $[t]$ by tossing a fair coin $c \in \{0, 1\}$
if $c = 0$ **then**
    A: $[\tau] \leftarrow [t]$
else
    A: $[\tau] \leftarrow [1] \cdot [t] \bmod N$ ▷ $\tau \leftarrow 1 \oplus t$
end if
$\{\tau = c \oplus t\}$
A sends $[\tau]$ to B
B decrypts $[\tau]$, encrypts it with Pallier $[\![.]\!]$, and sends $[\![\tau]\!]$ to A

if $c = 0$ **then**
A: $[\![t]\!] \leftarrow [\![\tau]\!]$
else ▷ A unblinds t
A: $[\![t]\!] \leftarrow [\![1]\!] \cdot [\![\tau]\!]^{-1} \bmod N$ ▷ $t \leftarrow 1 - \tau$
end if
{Now $[\![t]\!]$ is the Pallier encrypted bit t_ℓ}

It's also possible to obtain the comparison result in the client–server model as a shared secret. To this end, the final lines of Algorithm 4, starting with line 9, have to be modified slightly:

$(t_A, t_B) \leftarrow \mathrm{LSIC}(c, d)$
$\{(t_A \oplus t_B = 1) \equiv (c < d)\}$
A: $t_A \leftarrow r_{\ell+1} \oplus t_A$
B: $t_B \leftarrow z_{\ell+1} \oplus t_B$
$\{t_A \oplus t_B = x \div 2^\ell\}$

The shared output solution saves two multiplications by A, and one encryption and transmission of the number $[z_{\ell+1}]$ by B. On the other hand, the shared output version of LSIC requires an extra blinding action and transmission by A and also an extra decryption by B.

6.1. Correctness

Since $x = b + 2^\ell - a$, it is clear that the most significant bit of x, i.e., $x \div 2^\ell$, will be equal to the comparison result of $a \le b$, i.e., $x \div 2^\ell = 1$ if and only if $a \le b$, since both a and b are ℓ bits long. This proves that the assertion in line 14 indeed gives the correct output.

In order to understand line 13, where the number $x \div 2^\ell$ is computed, both in the QR encrypted and the Pallier encrypted version, observe that for each positive integer x, the number $x \div 2^\ell$ is defined by $x = 2^\ell \cdot (x \div 2^\ell) + x \bmod 2^\ell$ such that $0 \le x \bmod 2^\ell < 2^\ell$. Since

$$\begin{aligned} z &= \\ x + r &= \\ 2^\ell \cdot (x \div 2^\ell) + x \bmod 2^\ell + 2^\ell \cdot (r \div 2^\ell) + r \bmod 2^\ell &= \\ 2^\ell \cdot ((x \div 2^\ell) + (r \div 2^\ell)) + ((x \bmod 2^\ell) + (r \bmod 2^\ell)) & \end{aligned}$$

we know that $z \div 2^\ell = (x \div 2^\ell) + (r \div 2^\ell)$ and $z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell)$ whenever $(x \bmod 2^\ell) + (r \bmod 2^\ell) < 2^\ell$, and $z \div 2^\ell = (x \div 2^\ell) + (r \div 2^\ell) + 1$ and $z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell) - 2^\ell$, otherwise. In the first case, $z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell) \ge r \bmod 2^\ell$. In the second case, $z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell) - 2^\ell < r \bmod 2^\ell$. So

$$\begin{aligned} z \div 2^\ell = (x \div 2^\ell) + (r \div 2^\ell) &\equiv \\ (x \bmod 2^\ell) + (r \bmod 2^\ell) < 2^\ell &\equiv \\ z \bmod 2^\ell \ge r \bmod 2^\ell &\equiv \\ d \le c &\equiv \\ t = 0 & \end{aligned}$$

The relation

$$x \div 2^\ell = (z \div 2^\ell) - (r \div 2^\ell) - t$$

easily follows, which shows the correctness of line 13 in the Pallier encrypted version. The correctness in the QR encrypted version follows by observing that $z \div 2^\ell = z_{\ell+1+\sigma} \ldots z_{\ell+1} = 2 \cdot (z_{\ell+1+\sigma} \ldots z_{\ell+2}) + z_{\ell+1}$, so $(z \div 2^\ell) \bmod 2 = z_{\ell+1}$. Since $x \div 2^\ell$ is a bit value, we have $x \div 2^\ell = ((z \div 2^\ell) - (r \div 2^\ell) - t) \bmod 2 = (z_{\ell+1} - r_{\ell+1} - t) \bmod 2$, so

$$x \div 2^\ell = z_{\ell+1} \oplus r_{\ell+1} \oplus t$$

The correctness of the shared output version of the client–server protocol also follows immediately from this equation.

6.2. Security

We prove the security of the first mentioned variant of the client–server protocol as depicted in Algorithm 4. The proof of the other variants is similar.

In order to show that our protocol privately computes the comparison of two integers in the semi-honest model, we have to show that whatever can be computed by A or B from their view of a protocol execution, can be computed from their input and the comparison result (see Definition 7.2.1 in Goldreich [21]).

The view of A consists of the encrypted numbers $[\![a]\!]$ and $[\![b]\!]$, the size ℓ, the random number r, the encrypted comparison bit $[t]$, and all intermediate messages received from B. Besides $[z_{\ell+1}]$, this also includes all intermediate messages in the LSIC subprotocol: the encrypted bits $[d_i]$, $0 \leq i < \ell$, which are the encrypted bits of the number d, and the encrypted bits $[td_i]$, $1 \leq i < \ell$, which equal $[d_i \cdot \tau_i]$, τ_i being the blinded version of the bit $(c^i < d^i)$. Summarizing, the view of A equals

$$V_A = ([\![a]\!], [\![b]\!], \ell, r, [t], [z_{\ell+1}], [d_0] \dots [d_{\ell-1}], [td_1], \dots [td_{\ell-1}])$$

It suffices to show that there exists a probabilistic polynomial-time algorithm S_A such that $S_A([\![a]\!], [\![b]\!], \ell, [t])$ is computationally indistinguishable from V_A [21]. Since both encryption algorithms are semantically secure, every pair of encryptions is computationally indistinguishable, so by letting S_A randomly generate 2ℓ encryptions and one random number with the same distribution as r, this condition is easily verified.

The view of B consists of the decryption key K and the size ℓ, and all intermediate messages received from A. Besides $[\![z]\!]$, this also includes all intermediate messages in the LSIC subprotocol: the blinded bits $[\tau_i]$, $1 \leq i < \ell$, τ_i being the blinded version of the bit $(c^i < d^i)$. The comparison result $x \div 2^\ell$ and the intermediate comparison result t are not received by B. Since B owns the decryption key, all encrypted values $[\tau_i]$ can be decrypted to τ_i, just like the number z. Additionally, B is able to deduce the randomization part of each encryption, but since A carefully uses re-randomization before each transmission, this information can be considered as a random variable and is therefore useless to B. Summarizing, the view of B is equivalent to

$$V_B = (K, \ell, z, \tau_1, \dots \tau_{\ell-1})$$

Again, we have to show that there exists a probabilistic polynomial-time algorithm S_B such that $S_B(K, \ell)$ is computationally indistinguishable from V_B. In Equation (2), it was shown that the bits τ_i are uniformly distributed, so we could let S_B generate $\ell - 1$ random bits for those. The number z is a number of $l + 1 + \sigma$ bits, but is, due to its construction, statistically indistinguishable from a random number of equal length, which means that the difference between both probabilities is bounded by $2^{-\sigma}$. Therefore, we can even prove the stronger assertion that $S_B(K, \ell)$ is statistically indistinguishable from V_B.

We conclude that the private input of B is computationally secure towards A, and that the private input of A is statistically secure towards B. We use the term statistically secure instead of perfectly secure, although the private input of A is secure even when B has unbounded computer power. The difference with perfect security is that the amount of information leakage is not zero, but negligible.

7. Conclusions

We described a new protocol for the millionaires problem using additively homomorphic encryption in the honest-but-curious model. There are no restrictions on the size of the inputs. Since this protocol based on homomorphic encryption doesn't use intermediate decryptions, it has the lowest number of modular multiplications of all known solutions, and also a low communication and storage complexity, making it preferable

for light-weight environments. The (maximal) number of multiplications modulo N is even less than the number of modular multiplications needed for the Pallier encryption of an ℓ bit number. The number of communication rounds is equal to the number of input bits. The private input of the first player (A) is computationally secure towards the second player (B), and the private input of the second player is even perfectly secure towards the first player. Furthermore, we showed how to transform any comparison protocol to the client–server model in an efficient and secure way. The client–server solution offers computational security for B, and even statistical security for A. We showed three output variants for both the private input as the client-server model, namely a publicly known output, an encrypted output, and a secret-shared output.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Lagendijk, R.L.; Erkin, Z.; Barni, M. Encrypted Signal Processing for Privacy Protection. *IEEE Signal Process. Mag.* **2012**, *30*, 82–105. [CrossRef]
2. Goldwasser, S.; Micali, S. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th ACM Symposium on the Theory of Computing (STOC 1982)*; ACM: New York, NY, USA, 1982; pp. 365–377.
3. Goldwasser, S.; Micali, S. Probabilistic encryption. *J. Comput. Syst. Sci.* **1984**, *28*, 270–299. [CrossRef]
4. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology (EUROCRYPT 1999)*; Lecture Notes in Computer Science; Springer: New York, NY, USA, 1999; Volume 1592, pp. 223–238.
5. Damgård, I.; Geisler, M.; Krøigaard, M. Efficient and Secure Comparison for On-Line Auctions. In *Australasian Conference on Information Security and Privacy— Proceedings of the ACSIP 2007, 2–4 July 2007, Townsville, Australia*; Pieprzyk, J., Ghodosi, H., Dawson, E., Eds.; Volume 4586 of LNCS; Springer: Berlin/Heidelberg, Germany, 2007; pp. 416–430.
6. Damgård, I.; Geisler, M.; Krøigaard, M. Homomorphic encryption and secure comparison. *J. Appl. Cryptol.* **2008**, *1*, 22–31. [CrossRef]
7. Damgård, I.; Geisler, M.; Krøigaard, M. A correction to efficient and secure comparison for on-line auctions. *J. Appl. Cryptol.* **2009**, *1*, 22–31. [CrossRef]
8. Erkin, Z.; Franz, M.; Katzenbeisser, S.; Lagendijk, R.L.; Merchan, J.G.; Toft, T. Privacy-Preserving Face Recognition. In *International Symposium on Privacy Enhancing Technologies Symposium*; LNCS 5672; Springer: Berlin/Heidelberg, Germany, 2009; pp. 235–253.
9. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, October 1996; ISBN 0-8493-8523-7.
10. Barker, E. *Recommendation for Key Management: Part 1—General*; NIST Special Publication 800-57; NIST: Gaithersburg, MD, USA, May 2020; Part 1, Revision 5.
11. Yao, A.C. Protocols for secure computations. In Proceedings of the Symposium on Foundations of Computer Science (SFCS'82), Chicago, IL, USA, 3–5 November 1982; pp. 160–164.
12. Kolesnikov, V.; Sadeghi, A.R.; Schneider, T. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *International Conference on Cryptology and Network Security*; Cryptology ePrint Archive, Report 2009/411; Springer: Berlin/Heidelberg, Germany, 2009.
13. Fischlin, M. A cost-effective pay-per-multiplication comparison method for millionaires. In *Cryptographers' Track at the RSA Conference*; Naccache, D., Ed.; Springer: Heidelberg, Germany, 2020; Volume 2020, pp. 457–472.
14. Garay, J.; Schoenmakers, B.; Villegas, J. Practical and Secure Solutions for Integer Comparison. In *Public Key Cryptography—PKC '07*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4450, pp. 330–342.
15. Makri, E.; Rotaru, D.; Vercauteren, F.; Wagh, S. *Rabbit: Efficient Comparison for Secure Multi-Party Computation*; Financial Crypto; Springer: Berlin, Germany, 2021; pp. 249–270.
16. Bourse, F.; Sanders, O.; Traoré, J. Improved Secure Integer Comparison via Homomorphic Encryption. In Proceedings of the RSA Conference 2020, San Francisco, CA, USA, 24–28 February 2020; Topics in Cryptology—CT-RSA 2020.
17. Couteau, G. New protocols for secure equality test and comparison. In *Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2018.
18. Kerschbaum, F.; Terzidis, O. Filtering for Private Collaborative Benchmarking, International Conference on Emerging Trends in Information and Communication Security. 2006. Available online: http://www.fkerschbaum.org/etrics06.pdf (accessed on 9 December 2021).
19. Kerschbaum, F.; Biswas, D.; de Hoogh, S. Performance Comparison of Secure Comparison Protocols. In Proceedings of the 1st International Workshop on Business Processes Security, Linz, Austria, 31 August–4 September 2009. Available online: http://www.fkerschbaum.org/bps09a.pdf (accessed on 9 December 2021).

20. Veugen, T. *Comparing Encrypted Data*; Delft University of Technology: Delft, The Netherlands, 2010; *unpublished*.
21. Goldreich, O. *Foundations of Cryptography, Volume II: Basic Applications*; Cambridge University Press: Cambridge, UK, 2004.
22. Bianchi, T.; Piva, A.; Barni, M. Analysis of the security of linear blinding techniques from an information theoretical point of view. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, Prague Congress Center, Prague, Czech Republic, 22–27 May 2011.
23. Blake, I.F.; Kolesnikov, V. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology, ASIACRYPT 2004, Proceedings of the 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, 5–9 December 2004*; Volume 3329 of LNCS; Springer: Berlin/Heidelberg, Germany, 2004; pp. 515–529.
24. Robshaw, M.J.B.; Yin, Y.L. Overview of Elliptic Curve Cryptosystems. *CryptoBytes Technical Newsletter*. RSA Laboratories. 27 June 1997. Available online: http://www.rsa.com/rsalabs/node.asp?id=2013 (accessed on 9 December 2021).