COMPARING ENCRYPTED DATA

Thijs Veugen

Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and
TNO Information and Communication Technology, Delft, The Netherlands

ABSTRACT

When processing signals in the encrypted domain, homomorphic encryption can be used to enable linear operations on encrypted data. More complicated operations like comparison, modulo reduction, or division, require a comparison protocol as a building block. Since the overall computational complexity of signal processing on encrypted data is often determined by the comparison protocol, it is very important to find the most efficient solution. We present a new comparison protocol and compare it with existing solutions, outperforming them at least when precomputation complexity is included.

***Index Terms*—** homomorphic encryption, millionaires problem, integer comparison, secure multi-party computations

1. INTRODUCTION

Many secure protocols are known for comparing two integers, the so called millionaires problem. We present a new comparison protocol that requires little memory and has a low computational effort. For this reason we focus on the semi-trusted model where both players follow the required protocol steps. The protocols can be extended to the malicious model [1], but this will significantly enlarge the computational and communication complexity.

First an overview is presented on the most important related work. Then the definitions and notations are explained in the preliminaries. The actual comparison protocol, consisting of the client-server model, the new private comparison protocol called LSIC (Lightweight Secure Integer Comparison), and the security analysis, is presented in the second Section. In Section 3 the computational, communication and storage complexity of LSIC is analyzed and compared to other solutions. The final section contains the conclusions.

1.1. Related work

The first solution of the millionaires problem is by Yao [2] 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. One of the most efficient implementations nowadays based on garbled circuits is described by Kolesnikov, Sadeghi and Schneider [3]. A rough comparison with LSIC is made in Section 3. A well known candidate that uses homomorphic encryption is by Blake and Kolesnikov [4], which is optimized by Damgård, Geisler, and Krøigaard [5, 6], who use a dedicated crypto system fine tuned for small plain text values.

Other related work is e.g. by Fischlin [7], 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 sincerely increases the computational and communication load, including ℓ decryptions.

The idea of the conditional gate by Schoenmakers and Tuyls [8] is similar to our approach, but their setting is threshold decryption in the malicious model with multiple parties. Garay, Schoenmakers and Villegas [9] describe a nice solution for the client-server setting in the multi-party case, but since they use the malicious adversary model in stead of the honest-but-curious model, their solutions are less efficient.

Kerschbaum and Terzidis (KT) present an efficient solution to the millionaires problem in the semi-trusted model in [10]. This solution is later extended to multiple parties by Kerschbaum, Biswas and de Hoogh [11]. We compare their solution to LSIC in Section 3.

1.2. Preliminaries

For the public key crypto system in our protocols, any homomorphic and semantically secure crypto system 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. Quadratic Residues [12], and $[\![.]\!]$ to denote encryption of integers with e.g. Paillier [13] or Damgård, Geisler, and Krøigaard (DGK) [5, 6].

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 $[\![x]\!] \cdot [\![y]\!] = [\![x+y]\!] \bmod N$. During the complexity computations, we assume that N is 1024 bits long. For other key lengths, similar complexity conclusions can be drawn. For convenience, we neglect in our notation that the cipher text size in Paillier is N^2 and use N instead, just like in DGK or RSA.

We use pseudo code to describe the protocols. Assertions between $\{.\}$ are used to describe the current value of variables. Also 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 involution 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 [14]. An encrypted integer is negated most efficiently by using the Euclidean algorithm [14]. This is denoted by $[\![-x]\!] \leftarrow [\![x]\!]^{-1} \bmod N$, and its effort is comparable with one modular multiplication [14]. A Paillier decryption takes $\frac{3}{2}\log_2 N$ multiplications modulo N [13].

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.

2. COMPARISON PROTOCOL

When comparing encrypted signals, the most common model is the client-server model, where the client has encrypted signals and the server has the private key [15]. We first show how this problem can efficiently be converted to a comparison of two private integers, i.e. both client and server having a private integer. In the second subsection an efficient solution, called Lightweight Secure Integer Comparison (LSIC) is presented for the comparison of two private integers.

2.1. Client-server model

Assume the client A has two encrypted numbers $[\![a]\!]$ and $[\![b]\!]$ of ℓ bits, the server 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. $z_{\ell+1}$ is used to denote the $(\ell+1)$-th bit of integer z.

The main idea of Protocol 1 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 Protocol 1 any private comparison protocol can be used to compute $d < c$, as long as the output is encrypted.

Protocol 1 Client-server comparison

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

```
    {Both $a$ and $b$ consist of $\ell$ bits}
    A: $[\![x]\!] \leftarrow [\![b]\!] \cdot [\![2^\ell]\!] \cdot [\![a]\!]^{-1} \bmod N$      $\triangleright x \leftarrow b + 2^\ell - a$
    A chooses a random number $r$ for blinding $x$
    A: $[\![z]\!] \leftarrow [\![x]\!] \cdot [\![r]\!] \bmod N$      $\triangleright z \leftarrow x + r$
 5: A sends $[\![z]\!]$ to B
    B decrypts $[\![z]\!]$
    A: $c \leftarrow r \bmod 2^\ell$
    B: $d \leftarrow z \bmod 2^\ell$
    A and B privately compute the encrypted bit $[t]$ such that
    $\{(t = 1) \equiv (d < c)\}$
10: B encrypts $z_{\ell+1}$ and sends $[z_{\ell+1}]$ to A
    A encrypts $r_{\ell+1}$
    A: $[t] \leftarrow [z_{\ell+1}] \cdot [r_{\ell+1}] \cdot [t]$      $\triangleright t \leftarrow z_{\ell+1} \oplus r_{\ell+1} \oplus t$
    $\{t = x \div 2^\ell\}$
```

In addition to the private comparison protocol, which is executed in line 9, only 5 extra multiplications modulo N are needed (and one QR encryption), 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 [16], 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 involution to an exponent of size $\log_2 N$, which requires substantially more modular multiplications.

Note that although the inputs are encrypted by Paillier (or some other integer encryption system), the output is encrypted by QR (or some other bit encryption system). It's also possible to obtain the output encrypted by Paillier by replacing lines 10 to 12 by the following lines:

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

In order to understand line 12, 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 $0 \leq x \bmod 2^\ell < 2^\ell$. Since $z = x + r = 2^\ell \cdot ((x \div 2^\ell) + (r \div 2^\ell)) + ((x \bmod 2^\ell) + (r \bmod 2^\ell))$, 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, $z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell) \geq r \bmod 2^\ell$. In the second case,

$z \bmod 2^\ell = (x \bmod 2^\ell) + (r \bmod 2^\ell) - 2^\ell < r \bmod 2^\ell$. So

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

The relation

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

easily follows, which shows the correctness of line 12 in the Pallier encrypted version. The correctness in the QR encrypted version follows by observing that $z \div 2^\ell = z_{\ell+1+\sigma} \dots z_{\ell+1} = 2 \cdot (z_{\ell+1+\sigma} \dots z_{\ell+2}) + z_{\ell+1}$, 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

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

2.2. Comparing private integers

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 $0 \leq i < \ell$, where a_0 and b_0 are the least significant bits. We use the notation a^l $(1 \leq l \leq \ell)$ 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 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:

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

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 Protocol 2. The formal security proof, to show that A and B learn no private information from each other, is given in Subsection 2.3. 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 25, 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 unblinded again by A in line 28.

Protocol 2 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
Output A	encrypted 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 $t_{i+1}$ from $t_i$
        $\{t = t_i\}$
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 25
20:     else
            B: $[tb] \leftarrow [\tau]$
        end if     ▷ B computed $\tau \cdot b_i$ without decrypting $[\tau]$
        $\{tb = \tau \cdot b_i\}$
        B encrypts $b_i$
25:     B randomizes $[tb]$ and sends [tb] and $[b_i]$ to A
        $\{tb = (c \oplus t_i) \cdot b_i\}$
        if $c = 1$ then                              ▷ A unblinds $tb$
            A: $[tb] \leftarrow [tb] \cdot [b_i] \bmod N$          ▷ $tb \leftarrow tb \oplus b_i$
        end if
30:     $\{tb = t_i \cdot b_i\}$
        if $a_i = 0$ then
            A: $[t] \leftarrow [t] \cdot [b_i] \cdot [tb] \bmod N$   ▷ $t \leftarrow t \oplus b_i \oplus tb$
        else
            A: $[t] \leftarrow [tb]$
35:     end if
        $\{t = t_{i+1}\}$
    end for
    $\{t = t_\ell\}$
```

Altough the computations by A in lines 32 and 34 are not obvious, it's easily shown that A indeed correctly computes the next value $[t_{i+1}]$ from $[t_i]$, a_i, $[b_i]$, and $[b_i \cdot t_i]$. Since A knows the value of a_i, the recurrence relation 1 can be further reduced. In case $a_i = 1$, then $a_i < b_i$ will be false, so $t_{i+1} = t_i \cdot b_i$ which is the variable tb. In case $a_i = 0$, then $a_i < b_i$ will be equivalent to $b_i = 1$, so $t_{i+1} = 1$ whenever $b_i = 1$ or

$t_i = 1$, thus $t_{i+1} = b_i + t_i - b_i \cdot t_i = b_i + t_i - tb$.

The computations in lines 28, 32 and 34 could be further optimized to save one multiplication, but due to space constraints we refrain from that.

This protocol works for any number of bits ℓ. Since the protocol doesn't require intermediate decryptions, the computational complexity is low. The encryption system should be homomorphic and semantically secure. We use Quadratic Residues, because encryption and rerandomization are easy, but one could also use e.g. Paillier or DGK. This requires a small modification of the homomorphic operations on encrypted numbers, as depicted in the table below, where $[\![.]\!]$ is used to denote Paillier encryption.

Line	Operation	Paillier
14	$\tau \leftarrow 1 \oplus t$	$[\![\tau]\!] \leftarrow [\![1]\!] \cdot [\![t]\!]^{-1}$
28	$tb \leftarrow tb \oplus b_i$	$[\![tb]\!] \leftarrow [\![b_i]\!] \cdot [\![tb]\!]^{-1}$
32	$t \leftarrow t \oplus b_i \oplus tb$	$[\![t]\!] \leftarrow [\![t]\!] \cdot [\![b_i]\!] \cdot [\![tb]\!]^{-1}$

The modified operation $tb \leftarrow b_i - tb$ in line 28 works, because $tb = \tau \cdot b_i$, so $b_i \geq tb$. The modified operation $t \leftarrow t + b_i - tb$ in line 32 works, because whenever $t = b_i = 1$, we have $tb = t \cdot b_i = 1$.

2.3. Security analysis

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 [1]). The informal analysis is that all messages from A to B are blinded and thus reveal no private information. The messages from B to A are all encrypted, and since the encryption system is semantically secure, no private information is leaked in this direction too. Only a formal proof for LSIC, as described in subsection 2.2, is given, since the client-server reduction from subsection 2.1 is not new.

The view of A consists of its private number a, the size ℓ, the encrypted comparison bit $[t]$, the internal coin tosses c_i, $1 \leq i < \ell$, and all intermediate messages received from B: the encrypted bits $[b_i]$, $0 \leq i < \ell$, which are the encrypted bits of the number b, and the encrypted bits $[tb_i]$, $1 \leq i < \ell$, 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

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

It suffices to show that there exists a probabilistic polynomial-time algoritm S_A such that $S_A(a, \ell, [t])$ is computationally indistinguishable from V_A [1]. 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, 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 $(a^i < b^i)$. Since B owns the decryption key, all encrypted values $[\tau_i]$ can be decrypted to τ_i. Also, B is able to deduce the randomization part of each encryption, but since A carefully uses rerandomization 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

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

Again, we have to show that there exists a probabilistic polynomial-time algoritm S_B such that $S_B(b, K, \ell)$ is computationally indistinguishable from V_B. This is easily satisfied by letting S_B randomly generate $\ell - 1$ bits. The bits $\tau_i = c_i \oplus t_i$ are indeed uniformly distributed, because $\Pr(\tau = 1) = \Pr(\tau = 1 | c = 0) \cdot \Pr(c = 0) + \Pr(\tau = 1 | c = 1) \cdot \Pr(c = 1) = \Pr(t = 1) \cdot \frac{1}{2} + \Pr(t = 0) \cdot \frac{1}{2} = \frac{1}{2}$.

We conclude that LSIC indeed privately computes the (encrypted) comparison result. Note that unconditional (or information-theoretic) security of A towards B could also be shown because the blinding techniques completely hide the comparison bits, but due to space constraints this is not worked out.

3. PERFORMANCE

In this section we compute the computational, communication, and storage complexity of LSIC. We distinghuish between the cases where precomputations are allowed or not. Typical values that could be precomputed are encryptions of bits, and randomization factors that are used in the homomorphic crypto system. The actual application of the protocol will determine whether precomputations are allowed.

All homomorphic systems consist of an encryption and a randomization part. E.g. in Quadratic Residues $[x] = g^x \cdot r^2 \bmod N$, 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, 25), the result should be rerandomized. Note that altough B owns the decryption key (also called private key), rerandomization is even necessary for A, because B might be able to recognize the randomization part.

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 QR, the encryption of 0 requires one multiplication (squaring) modulo N, and the encryption of 1 requires two multiplications modulo N.

3.1. Without precomputations

When precomputations are not allowed, QR is preferred because the number of multiplications for encryption and rerandomization is very low.

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 rerandomization of $[\tau]$ in line 17, 0.5 for the possible conversion of $[tb]$ in line 28, and 1 for the computation of $[t_{i+1}]$ in line 32, for a total of $4(\ell - 1)$ multiplications. The expected computational load for B equals the ℓ encryptions of b_i in lines 1 and 25, and $\ell - 1$ rerandomizations of $[tb]$ in line 25, 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 $5(\ell - 1)$ for A and $4(\ell - 1) + 2$ B.

3.2. With precomputations

When precomputations are allowed, the choice of crypto system is not really relevant anymore as long as the modulus size is the same. When the comparison result should be encrypted with Paillier, which modulus size is N^2, the best approach would be to use DGK in LSIC, and afterwards transform the comparison result into Paillier. Namely, the advantage of DGK is the smaller modulus size N, and the fast bit decryption which is needed to transform to Paillier.

When precomputing all encryptions and rerandomizations, a rerandomization is reduced to one multiplication, so the average number of modular multiplications needed in LSIC are $3(\ell - 1)$ for A and $\ell - 1$ for B.

3.3. Communication complexity

The numbers that are sent in LSIC are all encrypted bits, resulting in encypted numbers of $\log_2 N$ bits.

A sends to B $\ell - 1$ times the value of $[\tau]$, for a total of $\ell - 1$ 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).

3.4. 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 3 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 2 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, and random factors used for rerandomization. The total number of storage units will depend on the implementation and the requirements with respect to waiting time, communication, computation and storage capacities.

3.5. Summary

The summary of the complexity analysis of LSIC is depicted in the table below. We ignore the efforts for key generation and key distribution.

Precomp.	Computation	Communication	Storage
No	$7(\ell - 1) + 1.5$	$3\ell - 2$	3
Yes	$4(\ell - 1)$	$3\ell - 2$	$\Theta(\ell)$

The computational complexity is measured in the average number of multiplications modulo N, the communication complexity is measured in the number of messages 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.

3.6. Alternative solutions

We compare LSIC to other solutions in the semi-honest model, but due to space limitations we refer for their details to the papers.

When computing the complexity of KT [10], we find a total number of multiplications modulo N equal to $23\ell + \frac{23}{2} \log_2 N$, which is good, but more than LSIC. Its communication complexity consists only of two encrypted messages, but its weaker notion of security (caused by multiplicative hiding) is a serious drawback.

A commonly used comparison protocol is DGK, which needs $(120 + \frac{7}{2}) \cdot \ell + \frac{3}{2}\ell \cdot \log_2(\ell + 2)$ modular multiplications, even when precomputations are used. The computational complexity is mainly determined by ℓ decryptions in the DGK crypto system, and clearly exceeds LSIC. The communication complexity is only 2ℓ which is smaller than LSIC. Even without precomputations, ℓ encryptions have to be separately stored. Its main advantage is that only one communication round is needed.

An interesting alternative is the garbled circuit approach. An important part of the garbled circuit is the oblivious transfer of all the input values of one party to the other party. This part involves public key operations, but even when using elliptic curve crypto, one encryption and decryption is comparable to 200 modular multiplications [17]. When precomputations are not allowed this results in a minimal computational complexity of 200ℓ multiplications which clearly exceeds LSIC. The communication complexity is approximately half times our communication complexity [3], but a garbled circuit requires only one communication round. The storate complexity is roughly equal to the communication complexity, which is the size of the circuit. An interesting property of garbled circuits is that many things can be precomputed, including the oblivious transfers. It's unclear how its reduced complexity compares to the complexity of LSIC, in the case precomputations are allowed, because the comparison of symmetric versus asymmetric operations depends very much on the setting of software and hardware.

4. CONCLUSIONS

We described an efficient implementation of the client-server secure comparison protocol, and a new protocol (LSIC) for the millionaires problem in the honest-but-curious model. Since this protocol doesn't use intermediate decryptions, it has a very low computational complexity, and also a low communication and storage complexity, making it preferable for light-weight environments. 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.

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. It's unclear how the computational complexity of LSIC relates to the garbled circuit approach with precomputed oblivious transfers.

5. REFERENCES

[1] Oded Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2, Cambridge University Press, 2001.

[2] A.C. Yao, "Protocols for secure computations," in *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1982, pp. 160–164.

[3] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," Tech. Rep. 411, Cryptology ePrint Archive, 2009.

[4] I.F. Blake and V. Kolesnikov, "Strong conditional oblivious transfer and computing on intervals," in *ASI-ACRYPT*. Advances in Cryptology, 2004, vol. 3329, pp. 515–529.

[5] I. Damgård and M. Geisler and M. Krøigaard, "Homomorphic encryption and secure comparison," *Journal of applied cryptology*, vol. 1, no. 1, pp. 22–31, 2008.

[6] I. Damgård and M. Geisler and M. Krøigaard, "A correction to efficient and secure comparison for on-line auctions," *Journal of applied cryptology*, vol. 1, no. 4, pp. 323–324, 2009.

[7] M. Fischlin, "A cost-effective pay-per-multiplication comparison method for millionaires," in *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, London, UK, 2001, pp. 457–472, Springer-Verlag.

[8] B. Schoenmakers and P. Tuyls, "Practical two-party computation based on the conditional gate," in *ASI-ACRYPT'04*. Advances in Cryptology, 2004, number 3329 in Lecture Notes in Computer Science, pp. 119–136, Springer.

[9] Berry Schoenmakers Juan Garay and José Villegas, "Practical and secure solutions for integer comparison," in *Public Key Cryptography - PKC'07*. 2007, vol. 4450, pp. 330–342, Springer-Verlag.

[10] Florian Kerschbaum and Orestis Terzidis, "Filtering for private collaborative benchmarking," in *International Conference on Emerging trends in information and communication security*, 2006.

[11] Florian Kerschbaum, Debmalya Biswas, and Sebastiaan de Hoogh, "Performance comparison of secure comparison protocols," in *1st international workshop on business processes security*, 2009.

[12] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[13] P. Pailler, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of Eurocrypt 1999*. 1999, vol. 1592 of *Lecture Notes in Computer Science*, pp. 223–238, Springer-Verlag.

[14] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied cryptography*, CRC Press, 1996.

[15] Tiziano Bianchi, Thijs Veugen, Alessandro Piva, and Mauro Barni, "Processing in the encrypted domain using a composite signal representation: pros and cons," in *IEEE International Workshop on Information Forensics and Security*, 2009.

[16] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Reginald L. Lagendijk, and Tomas Toft, "Privacy-preserving face recognition," in *Proceedings of the Privacy Enhancing Technologies Symposium*, Seattle, USA, 2009, pp. 235–253.

[17] M.J.B. Robshaw and Yiqun Lisa Yin, "Overview of elliptic curve cryptosystems," *CryptoBytes Technical Newsletter*, June 1997.