Secure Comparison Protocols in the Semi-Honest Model

T. Veugen*†, F. Blom*‡, S. J. A. de Hoogh§, Z. Erkin†
*TNO, Technical Sciences, The Hague, The Netherlands, thijs.veugen@tno.nl
†Delft University of Technology, Cyber Security Group, Delft, The Netherlands, z.erkin@tudelft.nl
‡VU University Amsterdam, Faculty of Sciences, Department of Mathematics, Amsterdam, The Netherlands, f.blom90@gmail.com
§Philips, Research, Eindhoven, The Netherlands, sebastiaan.de.hoogh@philips.com

***Abstract*—Due to high complexity, comparison protocols with secret inputs have been a bottleneck in the design of privacy-preserving cryptographic protocols. Different solutions based on homomorphic encryption, garbled circuits and secret sharing techniques have been proposed over the last few years, each claiming high efficiency. Unfortunately, a fair comparison of existing protocols in terms of run-time, bandwidth requirement and round complexity has been lacking so far. In this paper, we analyse the state-of-the-art comparison protocols for a two-party setting in the semi-honest security protocol. We analyse their performances in three stages, namely initialization, pre-processing and online computation, by implementing them on a single platform. The results of our experiments provide a clear insight for the research community into the advantages and disadvantages of the various techniques.**

I. Introduction

Thanks to the wide spread of the Internet, we have witnessed that more and more online applications have become a common part of daily-life activities for millions of people. We create and share ideas, documents, media content in social networks, do shopping, arrange travels, and even apply for tax refunds. In many of these online systems, user data plays an important role. As an example, consider personalized online recommender systems, which have shown phenomenal success in e-commerce. In such systems, customers receive a service that is fine-tuned to their preferences. As a result, this leads to an increase in the profit made by the service provider since the personalized system increases the probability of a possible purchase a the targeted customer. The online systems achieve personalization by deploying intelligent algorithms, for example collaborative filtering, to generate recommendations [1]. The intelligent algorithms in general heavily depend on the data collected from the users in the form of click-logs, browsing history, previous shopping lists, location, preferences, and so on. Regardless of the type of the data collected from the users, the close relation between many other types of online systems and their dependency on user data is undeniable.

Even though personalized services are very attractive, securing data, particularly for privacy protection against untrustworthy entities, is considered as a big challenge [31]. Traditional approaches like physical security, access control mechanisms, and storing encrypting data are vital components of any secure digital infrastructure. Unfortunately, past experiences have shown us that these measures are not sufficient, particularly against attacks from insiders and careless employees, who forget private data in public places [8]. Furthermore, there is a growing concern among people towards the service providers, who might use their privacy-sensitive data for other purposes than promised, or even share data with other entities [25].

The challenge of protecting private data against any potential misuse, without hampering the service, has attracted great deal of attention in recent years. Among many proposals, the idea of secure signal processing in the encrypted domain (SPED) has appeared to be a feasible solution for privacy protection [24]. SPED basically combines signal processing and cryptography in such a way that privacy-sensitive data is processed using cryptographic tools such as homomorphic encryption [2], secret sharing [35] and garbled circuits [40] for realizing the same system in the encrypted domain. The main idea in SPED is to provide a concealed version of the data, an encrypted version, to the service provider. Since the service provider does not have the decryption key, it cannot access the content. However, it is still possible to perform a number of operations on the encrypted data that will give the same output as the original system. More precisely, the service provider runs cryptographic protocols based on multi-party computation techniques [18]. The result is then given to the user or made available to both the user and the service provider depending on the protocol. Based on this approach, which is privacy-preserving and provably secure, numerous applications of SPED have been proposed in literature, including but not limited to, sugar beet auction [10], biometric data matching [15], [33], [3], recommender systems [16], data mining [37], digital watermarking [5], and anonymous fingerprinting [30].

The efficiency of SPED in terms of run-time, bandwidth and sometimes storage is seen as a major research challenge since the cryptographic tools used in the design of the algorithm are costly: encryption introduces data expansion, storage problems, and transmission costs. Furthermore, realizing operations, or functions, that are necessary for the algorithm are not trivial: custom-tailored protocols are necessary as there is no recipe that can provide an *efficient*, general solution. For example, while linear operations such as scaling are more

efficient when realized using homomorphic encryption, non-linear operations such as comparison, which constitutes a significant portion of the whole computation in [15], claimed to be efficient by using garbled circuits [22].

In this paper, we address one of the most arguable questions in SPED and aim at providing a conclusive answer: what is the best approach to realize a comparison protocol, one of the core operations that appear in almost all signal processing algorithms? To answer this question, we consider a fair setting: given the encryption of two integer values x and y of a certain bit length, compute a single encrypted bit δ that determines whether $x < y$. We consider the cryptographic comparison protocols that are interactive, two-party games between Alice (so called Server), who has the decryption key and Bob (Client), who has the two encrypted integers. This is a common setting in SPED and easily allows integrating one of the secure comparison protocols into a larger platform. To focus on realistic values of personal data, we limited the inputs x and y to 25 bits in our implementation.

Based on the above setting, we evaluate state-of-the-art protocols that are using (i) Homomorphic Encryption (HE), (ii) Garbled Circuits (GC), and (iii) Linear Secret Sharing. The complexity of an interactive protocol is usually measured using the round complexity (the maximal number of times parties wait for other parties to receive their messages), the communication complexity (the number of communicated bits), and the computational complexity (the time required for performing all computations).

To the best of our knowledge, there is no comparison protocol that has the best performance in all of the aforementioned measurements. We analyse the protocols in terms of run-time using reference implementations on the same platform, and count the number of messages that are communicated. We consider the semi-honest security model for all protocols. This model assumes that the involved parties, in our case Alice and Bob, perform the protocol steps properly, but are also allowed to store the messages from previous steps in an attempt to deduce more information than they are entitled to.

A. Notation

In the rest of the paper, we use the symbols summarized in Table I. Furthermore, we use the definition in [36] for a *round* of communication as follows: During one round of a secure multi-party computation protocol, Alice and Bob exchange a couple of messages, and do some local computations. The most important restriction of a round is that Alice is not allowed to send a message to Bob, which requires (computing using) information, received from Bob in the same round, and vice versa.

Throughout the paper, we use $(x < y)$ to denote the comparison result. Here, x and y are two arbitrary non-negative integers, and the comparison result is a single bit, which is one in the case where $x < y$, and zero otherwise.

B. Outline

The rest of the paper is organized as follows. Section II introduces the cryptographic tools we considered in this paper, namely homomorphic encryption, secret sharing, and garbled circuits. Section III describes in detail the comparison protocols, which are ordered by their cryptographic technique. Section IV presents the details of our reference implementation of the comparison protocols. Section V presents the performances of the five protocols implemented, in terms of communication and computational complexity. Finally, Section VI draws conclusions.

TABLE I

x, y	Integer values to be compared
ℓ	Bit length of x and y
z	The integer $x + 2^\ell - y$
$(x < y)$	Binary result of $x < y$
δ	Result of comparing x and y
ϵ	Result of comparing c and r
λ	Result of comparing d and u
γ, ρ, c, r, u	Private (random) integers
d	Public integer
c_i, r_i, u_i, d_i	Bits of c, r, u, d
$d()$	Integer division function
$\oplus$	Exclusive or
$\lceil . \rceil$	Rounding upwards
n	Encryption scheme modulus
$\mathcal{E}_{pk}$	Encryption with public key
$\mathcal{D}_{sk}$	Decryption with secret key
$[\![.]\!]$	Paillier encryption
$[\![.]\!]_D$	DGK encryption
u'	Message space prime of DGK
t	DGK scheme parameter
$[x]$	Sharing of x
p	Prime size of a share
w_i^0, w_i^1	Garbled input wires
b_i	Choice bit of Bob in GC
s_i^0, s_i^1	Choice values of Alice in GC
Δ	Global off-set
τ	Encrypted bit in DGK08
s	Random value in $\{-1, 1\}$ from DGK08
e_i, e'_i	Integers from DGK08
β_A	Random bit of Alice added to KSS09
α_i, ζ_i, μ_i, ξ_i	Integers from CH10

II. Cryptographic Settings

We consider two parties, Alice and Bob, in a SPED setting often referred to as "server" and "client". We assume that Alice and Bob use the semi-honest security model, meaning that they follow the rules of the protocol, but they also collect messages to obtain additional information. Different security models for secure signal processing applications are discussed in detail in [24], which argues that the semi-honest model seems to be accepted widely by the community, since it offers a sufficient level of security within reasonable computational and communication costs. For more security, other models, such as the malicious model, could be used at the expense of increased complexity. However, for applications, like the ones that generate recommendations, which require privacy protection for a limited amount of time, a semi-honest security model is acceptable. For a few other applications, such as biometric data recognition systems, a malicious model should be considered since the privacy risk is higher.

In the rest of this section, we provide a high-level description of the cryptographic tools considered in this paper.

A. Homomorphic Encryption

A number of encryption schemes are known to preserve some structure after encryption, which can be exploited to perform operations on the plaintext without decryption.

In this paper, we rely on *additively* homomorphic encryption schemes like the Paillier scheme [29] and the DGK scheme [19]. Given messages m_1 and m_2, additive homomorphic encryption has the following property:

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \cdot \mathcal{E}_{pk}(m_2)) = (m_1 + m_2) \bmod n,$$

where $\mathcal{E}_{pk}$ and $\mathcal{D}_{sk}$ are the encryption and decryption functions with the public and the secret key, respectively, and n is a large integer. It follows that $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)^c) = (cm) \bmod n$ for any m in the message space, and positive integer c.

In both Paillier and DGK, n is a product of two large primes. For Paillier, the cipher text space is $\mathbb{Z}_{n^2}$ and the message space is $\mathcal{M} = \mathbb{Z}_n$, while in DGK, the cipher text space is $\mathbb{Z}_n$, and the message space $\mathbb{Z}_{u'}$ for a prime u', $u' < n$, of at most 20 bits. Both schemes are semantically secure.

B. Garbled Circuits

As it is a standard protocol for secure computation, we consider the garbled circuits (GC) originating from [40]. We consider two parties, a *constructor* that generates a GC (server Alice), and an *evaluator* that evaluates the circuit (client Bob). Together the parties want to verify the output of some function along a (binary) circuit without giving away the values of their private inputs.

This is achieved as follows. Alice creates a garbled circuit $\mathcal{C}$. For each input wire of the circuit, Alice randomly chooses two garbled values w_i^0, w_i^1. Furthermore, Alice creates a garbled table T_i for each gate in that circuit. Alice sends the garbled circuit to Bob, who also obliviously obtains the garbled inputs of Alice. Bob evaluates the circuit, gate by gate, using the garbled tables, and finally, translates the output of the circuit into the output values.

Obtaining the wires corresponding to the real inputs of Alice, such that the actual inputs are still hidden to Bob, is trivial. Alice simply sends the wires for her input bits to Bob. However, Bob needs the correct garbled value of his own input bits without revealing his input bits to Alice. This is achieved by running an oblivious transfer (OT) protocol. An OT is a two-party protocol between Alice and Bob, where Alice has two values s_i^0 and s_i^1, each associated with 0 and 1, respectively, and Bob has a choice bit $b_i \in \{0, 1\}$. At the end of the protocol, Bob only obtains the value $s_i^{b_i}$, and Alice learns nothing about b_i.

An efficient version of garbled circuits is described in [21]. The main idea is to construct the garbled values of the input wires using a global off-set Δ, that is $w_i^0 = w_i^1 \oplus \Delta$. By doing so, it is possible to compute the xor gate without communication, and negligible computation, by $w_3 = w_1 \oplus w_2$, where w_1 and w_2 are the garbled inputs of the xor gate, and w_3 is the output. However, while in this construction xor gates are *free*, non-xor gates have to be constructed with a different approach: the garbled values consist of a symmetric key and a single permutation key. We refer readers to [23], [21] for more details.

C. Linear Secret Sharing

In our linear secret sharing system, all values are additively shared between Alice and Bob, modulo some prime p. We write $[x]$ for a sharing of x. A shared value $[x]$ is opened, i.e. made public to Alice and Bob, by exchanging the shares of x between Alice and Bob, and locally calculating the sum of the shares.

Additions and multiplications with public values can be computed locally, requiring no interaction between Alice and Bob. Note that only one party should add the public value to their respective share when this public value is added to the sharing. A secure shared multiplication $[xy]$ can be done through a precomputed shared triplet $([a], [b], [c])$ such that $a \cdot b = c \bmod p$ (see e.g. [9]):

1) Alice and Bob locally compute $[e] = [x] - [a]$, and open $e = x - a$.
2) Alice and Bob locally compute $[d] = [y] - [b]$, and open $d = y - b$.
3) Alice and Bob locally compute $[z] = [c] + d[a] + e[b] + de$, which satisfies $z = xy$ because $xy = (a + e)(b + d) = c + ad + eb + de$.

A secure multiplication requires 3 local multiplications for Alice and Bob. The triplet $([a], [b], [c])$ has to be (pre)computed and communicated as described in Appendix C. The values d and e have to be opened during the secure multiplication, which requires communication between Alice and Bob.

III. Comparison protocols

The secure comparison protocols we consider have the following specification.

Party	Alice	Bob
Inputs	private Paillier key	$[\![x]\!]$ and $[\![y]\!]$
Output	–	$[\![\delta]\!]$
Constraints	$0 \leq x, y < 2^\ell < n$ and $\delta = (x < y)$	

Each protocol will consist of three main steps:

1) Alice and Bob transform the encrypted inputs to two privately held inputs.
   a) Bob picks a random ρ of size $\ell + \kappa$ to mask $z = x + 2^\ell - y$, calculates $[\![\gamma]\!] = [\![\rho + z]\!] = [\![\rho + 2^\ell + x - y]\!] = [\![\rho + 2^\ell]\!][\![x]\!][\![y]\!]^{-1}$, and sends $[\![\gamma]\!]$ to Alice. Also, Bob calculates $r = \rho \bmod 2^\ell$.
   b) Alice obtains γ after decryption, and calculates $c = \gamma \bmod 2^\ell$.
2) Alice and Bob securely compare the two ℓ-bit values c, held by Alice, and r, held by Bob.
3) Given ρ, γ, and the secured bit $\epsilon = (c < r)$, Alice and Bob compute the encrypted comparison bit $[\![\delta]\!]$.

The first step will be the same for all protocols, and requires $2^{\kappa+\ell+3} < n$, which is easily fulfilled as the message space of any additively homomorphic encryption scheme is large in respect to the size of the integers we compare. Note that $(x < y) = 1 - z_\ell$, where z_ℓ is the most significant bit of z. The first step is important because it is easier to determine the bits of c and r, than of x and y, which makes a *bitwise* comparison protocol possible. A secure comparison is less complex when

the bits of the inputs have been (securely) determined. During the first step, the inputs x and y are statistically hidden for Alice, and computationally secure for Bob.

The second and third step will be different for each protocol. The secure bitwise comparison of c and r will yield either an encryption, a sharing, or a garbled value of the comparison bit, depending on the cryptographic setting. This setting also influences the transformation to $[\![\delta]\!]$ in the third step.

We use five different cryptographic solutions for the second step, and analyze their advantages and disadvantages. An overview is depicted in Table II. The first solution is by Damgård, Geisler and Krøigaard [19] and uses a dedicated homomorphic encryption system. The second is based on garbled circuits. We slightly modified the circuit of Kolesnikov, Sadeghi and Schneider [21], implemented the oblivious transfers as described by Naor and Pinkas [26], and precomputed those as suggested by Beaver [4]. The last three solutions are all based on linear secret sharing, but their behavior is different in terms of communication rounds, and the size p of each sharing, because of the required head room between ℓ and p. The three secret sharing protocols we consider are by Nishide and Ohta [27], Catrina and de Hoogh [7], and Garay, Schoenmakers and Villegas [17].

Medium term security refers to the NIST key length specifications for 2011-2030, as depicted in Table III. We call CH10 the limited range variant because it needs a head room of κ bits in each sharing. All five protocols are described in more detail in the following subsections.

A. Homomorphic Encryption

The bit z_ℓ, which is the result of the comparison of x and y, can be computed as the integer result of the division of z by 2^ℓ, denoted by the function $d : \mathbb{N} \to \mathbb{N}$ where $d(\alpha) := \lfloor \alpha/2^\ell \rfloor$. Once the encrypted comparison bit $[\![\epsilon]\!] = [\![(c < r)]\!]$ has been computed, the encrypted output $[\![\delta]\!]$ can be computed by Bob through

$$\delta = 1 - d(z) = 1 - d(\gamma) + d(\rho) + (c < r), \qquad (1)$$

as shown by Veugen [38]. It only requires Alice in step 3 to calculate $d(\gamma)$, encrypt it, and send $[\![d(\gamma)]\!]$ to Bob, so he can compute $[\![\delta]\!] = [\![1 + d(\rho)]\!] \cdot [\![d(\gamma)]\!]^{-1} \cdot [\![\epsilon]\!] \bmod n^2$.

The remaining problem is specified by

Party	Alice	Bob
Inputs	private DGK key and c	r
Output	–	$[\![\epsilon]\!]$
Constraints	$0 \le r, c < 2^\ell < n$ and $\epsilon = (c < r)$	

We use the dedicated DGK homomorphic encryption scheme, and the bitwise comparison protocol DGK08 from [19], which has the following steps, to compute $[\![\epsilon]\!]$.

1) Alice separately encrypts, with the dedicated scheme $[\![.]\!]_D$, the first ℓ bits of c, that is $c_0, \ldots, c_{\ell-1}$, and sends them to Bob.
2) Bob randomly chooses an $s \in \{1, -1\}$ and computes the following values.

$$[\![e_i]\!]_D = [\![s + r_i - c_i + 3 \sum_{j=i+1}^{\ell-1} c_j \oplus r_j]\!]_D \ , \quad \text{and}$$

$$[\![e_\ell]\!]_D = [\![s - 1 + 3 \sum_{j=0}^{\ell-1} c_j \oplus r_j]\!]_D$$

for $0 \le i < \ell$, where c_i and r_i are the i'th bit of c and r, respectively.

3) After computing all e_i values, Bob masks them by picking $s_i \in \mathbb{Z}_u^*$ and $2t$ bit integers s_i' at random and setting $[\![e_i']\!]_D = [\![e_i]\!]_D^{s_i} \cdot h^{s_i'}$.
4) Bob applies a random permutation and sends them to Alice.
5) Alice checks whether there is a zero among the decrypted $[\![e_i']\!]_D$'s. If there is, Alice sends $[\![\tau]\!] = [\![0]\!]$, and $[\![\tau]\!] = [\![1]\!]$, otherwise.
6) Bob corrects the encrypted bit τ according to the value of s:

$$[\![\epsilon]\!] = [\![(c < r)]\!] = \begin{cases} [\![\tau]\!] & \text{if } s = 1, \\ [\![1]\!][\![\tau]\!]^{-1} & \text{if } s = -1. \end{cases}$$

In this protocol, s hides the comparison direction from Alice, that is, Alice cannot learn whether the protocol outputs the comparison result of $c > r$ or $c < r$. The weight 3 in step 2 is to prevent obtaining false zeros due to the sum of s, c_i and r_i. The value e_ℓ is used to achieve perfect security towards Alice in case $c = r$. More details can be found in [39].

B. Garbled Circuits

In this section, we describe the bitwise comparison KSS09 using garbled circuits [21]. Instead of a comparison circuit, they actually use a closely related subtraction circuit. To obtain the comparison bit $\delta = 1 - z_\ell$, they bitwise compute $z = \gamma - \rho$ until the $(\ell + 1)$-th bit, and invert that subtraction bit, which is similar to computing $\epsilon = (c < r)$.

We introduced an efficient way of transforming the binary result back to a Paillier encryption. Namely, Alice generates a random bit β_A, encrypts it, and sends $[\![\beta_A]\!]$ to Bob. The bit is used to randomize the $(\ell + 1)$-th bit of Alice's input to the circuit, to eventually blind the result δ to Bob. Bob, as an evaluator of the circuit, will eventually obtain the ungarbled value $\beta_A \oplus \delta$, which will tell him whether to invert the encrypted coin $[\![\beta_A]\!]$ or not, to obtain $[\![\delta]\!]$.

The circuit is composed of ℓ sequential 1-bit subtraction circuits $(-)$, as shown in Figures 1 and 2. Since we are not interested in the actual bits of the subtraction, only the carry-over to the next bit position is computed. As the 1-bit subtraction circuit consists of one AND gate with four table entries and three free XOR gates, the overall size of the subtraction circuit is 4ℓ table entries. The inversion is established by setting the first carry-over bit to 1. For a correctness proof of the circuit, we refer the reader to [21].

To obliviously transfer Alice's inputs to Bob, we implemented the protocol as described by Naor and Pinkas [26], and precomputed the oblivious transfers as suggested by Beaver [4]. Since the implementation focussed on a limited number of comparison protocols with a small number of input bits, we did not use OT-extension. This technique, originally proposed by Ishai et al. [20], provides a way of significantly speeding up the simultaneous precomputation of many oblivious transfers. This is described in more detail in Subsection IV-B, and its effect on performance has been taken into account of our analysis.

TABLE II
PROPERTIES OF OUR BITWISE COMPARISON PROTOCOLS

Bitwise comparison	KSS09 (GC)	DGK08 (HE)	NO07	CH10	GSV07
Crypto technique	garbled circ.	homom. encr.	linear secret sharing		
Constraints	medium term security		$2^{\ell+1} < p$	$2^{\kappa+\ell+3} < p$	$2^{\ell+1} < p$
Comm. rounds	constant	constant	constant	constant	linear
Security	comp.	Alice: perfect, Bob: comp.	comp.	statistical	perfect

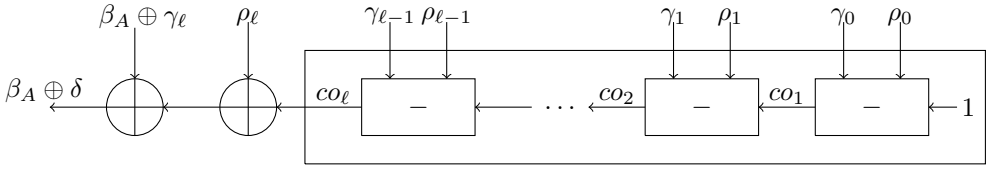

Fig. 1. Subtraction circuit derived from [21].

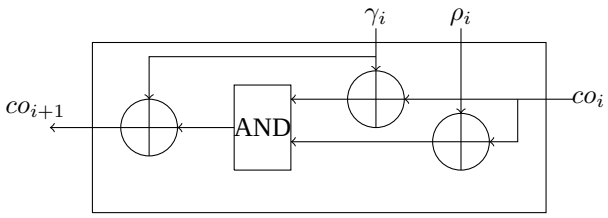

Fig. 2. 1-bit subtraction circuit from [21].

C. Linear Secret Sharing

We consider three different protocols based on secret sharing for a bitwise secure comparison of c and r. The outcome of these protocols will be a sharing of the bit $\epsilon = (c < r)$. To transform this sharing back to the encrypted comparison result $[\![\delta]\!]$, the following steps are required.

1) Alice and Bob hold shares ϵ_A and ϵ_B such that $\epsilon = \epsilon_A + \epsilon_B = (c < r) \bmod p$. Alice encrypts $[\![\epsilon_A]\!]$ and sends it to Bob.
2) Bob considers three different cases. If $\epsilon_B = 0$, then $[\![\epsilon]\!] \leftarrow [\![\epsilon_A]\!]$. If $\epsilon_B = 1$, then $[\![\epsilon]\!] \leftarrow [\![1-(p-1)^{-1}\epsilon_A]\!] = [\![1]\!] \cdot [\![\epsilon_A]\!]^{-(p-1)^{-1}}$, where $(p-1)^{-1}$ is the multiplicative inverse of $p-1$ modulo n. If $\epsilon_B > 1$, then $[\![\epsilon]\!] \leftarrow [\![\epsilon_A + \epsilon_B - p]\!] = [\![\epsilon_A]\!] \cdot [\![\epsilon_B]\!] \cdot [\![p]\!]^{-1}$.
3) Alice calculates $d(\gamma)$, encrypts it, and sends it to Bob, who computes $[\![\delta]\!]$ through Equation 1.

The correctness of step 2 follows from $\epsilon_A + \epsilon_B \in \{0, 1, p, p+1\}$ and $p < n$. Because the bitwise comparison protocols in the linear secret sharing setting require c to be public rather than privately known to Alice, we introduce a small transformation due to Nishide and Ohta [27].

Party	Alice	Bob
Input	c	r
Outputs	Sharings of random bits $[u_i]$, $0 \le i < \lceil \log_2 p \rceil$, and a public value d	
Constraints	$u = \sum_{i=0}^{\lceil \log_2 p \rceil - 1} 2^i u_i$, $0 \le d, u < p$ $0 \le c, r < 2^\ell$, $2^{\ell+1} - 2 < p$, $d = 2(c-r) + u \bmod p$	

1) Alice and Bob generate $\lceil \log_2 p \rceil$ shared random bits $[u_i]$, $0 \le i < \lceil \log_2 p \rceil$, such that $u = \sum_{i=0}^{\lceil \log_2 p \rceil - 1} u_i 2^i < p$ (see Appendix B).
2) Alice sets $v_A = c$, and Bob sets $v_B = -r \bmod p$, to create a sharing of $v = c - r$.
3) They compute $[d] = [2v + u] = 2[v] + [u]$, and open d.

Since the random integer u is uniformly drawn from the set $\{0, 1, \ldots, p-1\}$, the value v is perfectly hidden while opening d. Because $0 \le c, r < p/2$, we have $\epsilon = (c < r) = 1 - (v < p/2)$, which equals the right most bit of $2v$ because p is odd [27]. So once the sharing $[(d < u)]$ has been determined through a bitwise comparison protocol, this is sufficient to compute the shared bit $[(c < r)]$, because $\epsilon = (c < r) = (2v)_0 = (d < u) \oplus d_0 \oplus u_0 = d_0 \oplus u_0 + (d < u) - (d_0 \oplus u_0) \cdot (d < u)$. This requires one secure multiplication of $[d_0 \oplus u_0]$, which is easily computed because d_0 is public, with $[(d < u)]$. And from $[\epsilon]$, the encryption $[\![\delta]\!]$ can be computed as explained above.

We now describe three different secret sharing protocols for a bitwise comparison of a public value d with a bitwise shared random value u. The protocol by Catrina and de Hoogh uses a large p and provides statistical security, whereas the other two protocols use a small p with full blinding, and thus give perfect security.

1) Nishide and Ohta: Nishide and Ohta consider inputs c and r that can have full range, so $0 \le c, r < p$. They show that $\epsilon = (c < r)$ can be determined by comparing c, r, and $(c - r) \bmod p$ with $p/2$, and combining the results of these three comparisons. In order to reduce the effort to only one comparison, we choose p such that $0 \le c, r < p/2$. As explained above, the comparison $(c < r) = 1 - (v < p/2)$ can be computed through a bitwise comparison of d and u.

When picking appropriate p, we should be aware that our choice can influence the (pre)computation of a bit-wise shared random value u greatly, as explained in Appendix B. The bitwise comparison protocol of Nishide and Ohta is a constant rounds protocol, and uses subprotocols like "Unbounded Fan-In Or" and "Prefix-Or". We followed the exact implementation as described in their paper [27], to compute $[(d < u)]$. It requires 7 interactive rounds and 17ℓ secure multiplications,

of which 2 rounds and 9ℓ secure multiplications are used to precompute the random values.

2) Garay, Schoenmakers and Villegas: Just like Nishide and Ohta, Garay, Schoenmakers and Villegas [17] use small primes and a bitwise shared random value of full range. The following recurrence relation provides a linear depth circuit computing the bit $\lambda_{\lceil \log_2 p \rceil} = (d < u)$, where $\lambda_0 = 0$ and

$$\lambda_{i+1} = (1 - (u_i - d_i)^2)\lambda_i + u_i(1 - d_i).$$

To compute the secret shared bit $[(d < u)]$, given public d and sharings $[u_{\lceil \log_2 p \rceil - 1}], \cdots, [u_0]$, along this circuit, we proceed as follows:

1) Initiate $[\lambda] = [u_0(1 - d_0)]$, which is easy since d_0 is known.
2) For $i = 1, \ldots, \lceil \log_2 p \rceil - 1$ do:
   If $d_i = 0$ then $[\lambda] = (1 - [u_i])[\lambda] + [u_i]$, else $[\lambda] = [u_i][\lambda]$.

This protocol requires $\lceil \log_2 p \rceil$ interactive rounds and a same number of secure multiplications. We have exploited the fact that d is public to save 1 interactive round and $\lceil \log_2 p \rceil$ secure multiplications compared to the protocol of [17]. A similar protocol using homomorphic encryption can be found in [16].

3) Catrina and de Hoogh: The main difference between Catrina and de Hoogh's solution, and the previous two, is the size of p. Catrina and de Hoogh use a p of $\kappa + \ell + 3$ bits, which allows them to additively blind the inputs without causing a carry-over. This avoids an additional bitwise comparison protocol to ensure $u < p$ (see Appendix B), but requires computing with larger numbers. Furthermore, the transformation from the privately known c and r, to the public d and bitwise shared u, is slightly different:

1) Alice and Bob generate ℓ shared random bits $[u_i]$, $0 \leq i < \ell$ (see Appendix A).
2) Alice and Bob generate a shared random integer θ of κ bits, and compute $[u] = [\theta \cdot 2^\ell + \sum_{i=0}^{\ell-1} u_i 2^i] = [\theta]2^\ell + \sum_{i=0}^{\ell-1} [u_i] 2^i$.
3) Alice sets $v_A = c$, and Bob sets $v_B = 2^\ell - r$, to create a sharing of $v = 2^\ell + c - r$.
4) They compute $[d] = [v + u] = [v] + [u]$, and open d.

Since $v + u < p$, we have $d = v + u$, and can use Equation 1 to transform the comparison bit $(d \bmod 2^\ell < u \bmod 2^\ell)$ back to $(c < r)$: $\epsilon = (c < r) = 1 - v_\ell = 1 - d(d) + d(u) + (d \bmod 2^\ell < u \bmod 2^\ell) = 1 - \lfloor d/2^\ell \rfloor + \theta + (d \bmod 2^\ell < u \bmod 2^\ell)$. The first element of the comparison, $d \bmod 2^\ell$, is publicly known, and the second element, $u \bmod 2^\ell = u_{\ell-1} \ldots u_0$, is bitwise shared.

The constant round solution of Catrina and de Hoogh [7] for computing $[(d \bmod 2^\ell < u \bmod 2^\ell)]$ is based on the observations made by Reistad and Toft in [32]. Let, for $j = 0, \ldots \ell - 1$, $\alpha_j = (u_j \oplus d_j)(1 - d_j)\mu_{j+1}$, where $\mu_j = 2^{\sum_{i=j}^{\ell-1}(u_i \oplus d_i)} \neq 0$.

Observe that $\alpha_j = \mu_{j+1}$, if and only if, $d_j < u_j$, and that $\mu_{j+1} = 1$, if and only if, the $\ell - j$ most significant bits of $d \bmod 2^\ell$ and $u \bmod 2^\ell$ are equal. Thus, $(d \bmod 2^\ell < u \bmod 2^\ell)$, if and only if, there is exactly one j, where $\alpha_j = 1$. Since in all other cases, either $\alpha_j = 0$, or μ_{j+1} is even, it follows that $(d \bmod 2^\ell < u \bmod 2^\ell)$, if and only if, $E = \sum_{j=0}^{\ell-1} \alpha_j$ is odd.

To compute $[(d \bmod 2^\ell < u \bmod 2^\ell)]$, Alice and Bob both perform the following steps:

1) For $j = 0, \ldots, \ell - 1$ do:
   If $d_j = 0$, then set $[\xi_j] = [u_j]$, otherwise set $[\xi_j] = 1 - [u_j]$. We have $\xi_j = u_j \oplus d_j$.
2) For $j = 0, \ldots, \ell - 1$ do in parallel using a prefix multiplication protocol ([7]):
   Compute $[\mu_j] = \prod_{i=j}^{\ell-1} ([\xi_i] + 1)$.
3) Set $[\zeta_{\ell-1}] = [\mu_{\ell-1}] - 1$.
4) For $j = 0, \ldots, \ell - 2$ do:
   Compute $[\zeta_j] = [\mu_j] - [\mu_{j+1}]$, which satisfies $\zeta_j = \xi_j \mu_{j+1}$.
5) For $j = 0, \ldots, \ell - 2$ do:
   If $d_j = 0$, then set $[\alpha_j] = [\zeta_j]$, otherwise set $[\alpha_j] = [0]$.
6) Compute $[E] = \sum_{j=0}^{\ell-1} [\alpha_j]$.
7) Use the LSB protocol from [34] to compute the secret-shared least significant bit E_0 of E, which equals the comparison bit $(d \bmod 2^\ell < u \bmod 2^\ell)$.

This protocol only requires interaction between Alice and Bob while executing the prefix multiplication protocol and the LSB protocol. The others steps are performed by Alice and Bob on their own shares without communication, using the fact that d is known by both. It requires 5 interactive rounds and $5\ell + 1$ secure multiplications, of which 2 rounds, and 3ℓ secure multiplications are used to precompute the random values.

Overall, the complexity of the prefix multiplication protocol is a bit worse than the complexity of the prefix multiplication protocol from [7], because the protocols in [7] are based on Shamir Secret sharing, which allows for an efficient *secure multiplication with public result.*

IV. Implementation

In this section, we provide the details on the choice of parameters, and some optimisation for our implementation.

A. Picking a prime

In both large-range linear secret-sharing implementations, namely NO07 and GSV07 (see Subsection III-C), prime p is chosen depending on the input bit length, ℓ. Because $x, y < p/2$, we consider only primes larger than $2(2^\ell - 1)$. Furthermore, p should be large to reduce the number of retries for generating a bitwise-shared r such that $r < p$ (see Appendix B), but not too large because computing with larger numbers takes more time. We decided to look for the first prime larger than $2^{\ell+1} - 2$, and then take the largest prime with the same number of bits. More formally,

$$p = \min_k \left\{ \max_q \left\{ q \in \{0,1\}^k \mid q \text{ is prime, and } q > 2^{l+1} - 2 \right\} \right\}.$$

B. Oblivious transfer parameters

For the garbled circuits variant, we implemented a $\binom{2}{1}$-OT as Protocol 2.1 from [26]. As a requirement for this protocol, we need a group G_q of prime order q which is a subgroup of $\mathbb{Z}_p^*$ where p is prime and $q|p-1$. We also need a generator g such that $\langle g \rangle = G_q$, and for which the computational Diffie-Hellman assumption holds. In order to do this efficiently with a security strength of 112 (see Table III), we took p as a 2048 bit 'safe' prime, i.e. $p = 2q+1$. This way we ensure that $p-1$ has a large prime factor q. Generating such a safe prime on the spot can be rather costly, hence we defined it as a constant parameter in the implementation. Furthermore, a generator g' of $\mathbb{Z}_p^*$ can be found by randomly drawing elements from $\mathbb{Z}_p^*$, testing whether$(g')^2$ and $(g')^q$ are both non-equivalent with 1 modulo p, and finally setting $g \leftarrow (g')^2$.

For one comparison, $\ell + 1$ oblivious transfers are needed for transferring the first $\ell+1$ bits of γ from Alice to Bob. We use Beaver's solution [4] to precompute these OTs. However, for applications that require many secure comparisons, and which setting allows of simultaneously precomputing the comparisons, it will be worthwhile to use OT-extension [20]. This idea enables efficiently precomputing a (very) large number of OTs, basically using the effort of only k OTs, and extending the results. Here k is the size of the symmetric key used in the garbling process, which we set to 112 in our implementation, as depicted in Table III. Therefore, when more than $112/(\ell+1)$ comparisons have to be precomputed, OT-extension will be able to reduce the precomputation effort.

C. Negligible probability of abort

We have two constant-round solutions based on linear secret-sharing, namely NO07 and CH10. These protocols need randomly generated shared numbers satisfying certain conditions, which are not guaranteed to finish within a constant number of rounds. Therefore, these protocols incorporate a negligible probability of abort. We chose our parameters in such a way that this probability is always less than $2^{-\kappa}$, where the statistical security parameter $\kappa = 40$. In practice, this means that the time needed, and the amount of data communicated during the generation of these shared random numbers, heavily depends on κ.

The first type of random number which is difficult to generate, is a sharing of a random element of $\mathbb{Z}_p^*$, which is needed in both NO07 and CH10. Namely, when both Alice and Bob generate a random share, their sum might equal zero modulo p. Therefore, a large number of such random values are generated in parallel, and failing instances are replaced.

The second type is a bit-wise shared random element r, $r < p$ (see Appendix B), which is used only in NO07 (in GSV07 too, but they don't have a constant round limitation). This is done by generating the $\lceil \log_2 p \rceil$ shared random bits of r and than checking, via bit-wise comparison, whether $r < p$. In order to keep the probability of failure to generate r such that $r < p$ to a minimum, we pre-compute in parallel a number of such values, and execute multiple bit-wise comparisons.

D. Generating multiplication triplets

The protocol to generate multiplication triples is explained in Appendix C. To generate the triplets necessary for the implementation to be as efficient as possible, we preferred to use the Okamoto-Uchiyama scheme [28], which is also additively homomorphic, since it has a smaller cipher text space that leads to less costly exponentiations. New methods [9], [6] exist that might be more efficient for generating multiplication triplets, but time constraints refrained us from testing them.

E. Paillier's 'shortcut' scheme

The Paillier scheme uses a generator g of order n. To achieve high efficiency in encryption and decryption, we choose g as $g = n + 1$ such that calculating $g^m \bmod n^2$ becomes $g^m = (n+1)^m = (mn+1) \bmod n^2$, resulting in significant computational performance gain.

V. Performance results

In this section, we present the experimental results of aforementioned comparison protocols in two aspects: (i) communication in terms of number of rounds, data transmitted, and bandwidth, and (ii) computation time for initialization, pre-computation, and on-line computation stages. The bandwidth denotes the maximal amount of data transmitted in one round, which forms an interesting requirement on the communication channel. The number of communication rounds is often the bottleneck in practice, because the round trip delay can be relatively large, but this really depends on the communication technology used.

The code for our implementations (see http://cys.ewi.tudelft.nl/content/secure-comparison-protocols-semi-honest-model) was written in C/C++ using external libraries, namely MPIR [12], Boost [11], SeComLib [14], and OpenSSL [13]. The cryptographic key lengths were chosen according to the current NIST standard valid up till 2030, as given in Table III. The statistical security parameter κ was set to 40 bits. The experiments were conducted on a machine running Windows 7 Enterprise, 64-bit operating system, with an Intel(R) Core(TM) i3 CPU M 330 @ 2.13GHz and 4 GB of RAM. We varied the input size from 1 to 25 bits, run each experiment 100 times, and averaged the various results.

The number of rounds required in each protocol, given in Table IV, shows that while all other protocols have constant number of rounds, only GSV07 has a linear round complexity, which means that the number of rounds is linear in the input size.

TABLE IV
Number of rounds in each protocol

	GC	HE	GSV07	NO07	CH10
Initialization	1	1	1	1	1
Pre-processing	2	-	$3 + \log_2(p)$	14	4
Online computation	2	4	$3 + \log_2(p)$	13	6

The security properties of each protocol have already been given in Table II. Although in a SPED setting, the inputs x and

TABLE III
KEY LENGTHS

Security strength	Asymmetric key	Discrete Log key	Discrete Log Group	Hash
112	2048	224	2048	SHA-256

y are only computationally secure towards Alice, the security of the bitwise comparison protocols, having private inputs c and r, differ.

A. Initialization

For all of the five protocols, the initialization step consists of creating keys and variables to be used later in the protocols. In particular, this stage of the protocols is independent of the size of integer values to be compared. In Table V, we provide the average time for this stage for all protocols, for the sake of completeness.

TABLE V
AVERAGE TIME REQUIRED FOR INITIALIZATION

	GC	**HE**	**GSV07**	**NO07**	**CH10**
Time (s)	0.0071	154	4.1	4.1	4.1

The time for generating a safe prime in GC (see Section IV) is not counted here, because this can take several hours, so we fixed its value in our code.

B. Pre-Computation

During the precomputation phase, we compute most values that do not depend on the input values. In HE, this mainly concerns the random factors $h^{s'_i} \bmod n$ of the DGK system. In GC, the most intensive part of the oblivious transfers is precomputed. And for the secret sharing protocols, we compute the multiplication triplets, and the bitwise shared random numbers.

Figure 3 shows the time required for the pre-computations. Clearly, the HE based comparison protocol outperforms all other approaches, while NO07 has the worst performance. On top of that, NO07 also has several peaks in the required time for the pre-computations, due to the *probabilistic* abort conditions explained in the previous section. More precisely, for input length $\ell = 22$, the largest prime with 23 bits is $p = 2^{24} - 3$, which is much closer to the maximum $2^{24} - 1$ than is the case with $\ell = 21$ ($p = 2^{23} - 15$) and $\ell = 23$ ($p = 2^{25} - 39$). So for $\ell = 22$ we only need to generate a few 23-bit random numbers to have sufficiently many smaller than p. This peak effect of NO07 is also reflected in Figures 4 and 5.

The amount of data transmitted during the pre-computations is similar, and given in Figure 4: HE and GC based comparison protocols have less data transmission than other secret-sharing based protocols. Recall that secret sharing protocols require a significant amount of triplets to be computed in advance for the multiplications. Again, NO07 performs the worst among all. Notice that NO07 has also more peaks in the amount of data transmitted, due to a relatively large amount of multiplications in one round.

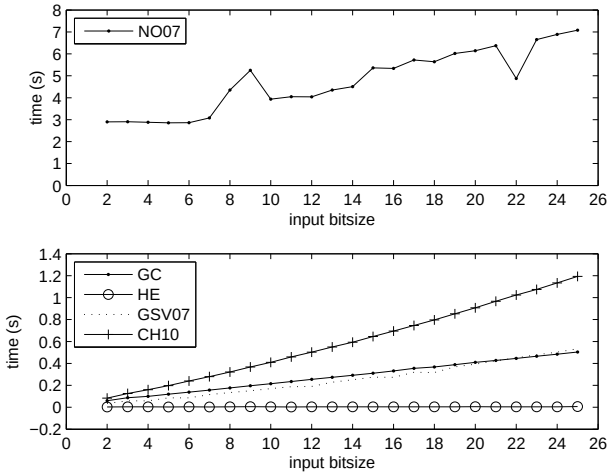

Fig. 3. Time for precomputing

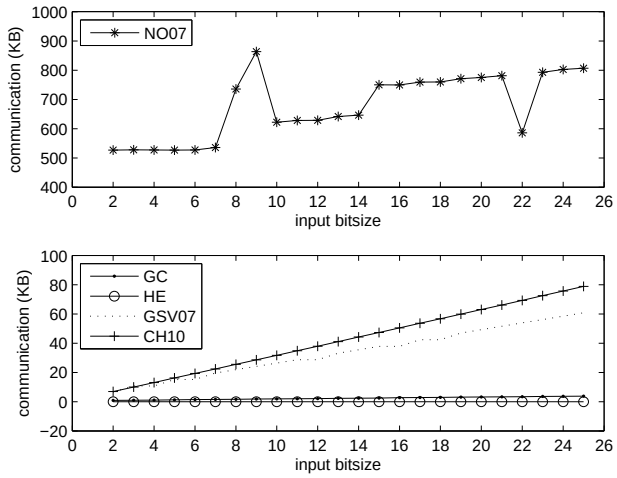

Fig. 4. Amount of transmitted data for precomputing

Considering bandwidth, we use the definition of maximum amount of data to be transmitted. Similar to the amount of data transmitted, the bandwidth requirement for all of the protocols follow a similar pattern: HE and GC have less bandwidth demand than the secret sharing based protocols, and NO07 has a significant amount of bandwidth usage, as shown in Figure 5.

As explained in Subsection IV-B, by using OT-extension the precomputation effort of GC can be considerably reduced when many comparisons have to be precomputed simultaneously. The total precomputation effort would then be roughly equal to the precomputation effort of one comparison with input bitsize 112, the average effort per comparison diminish-

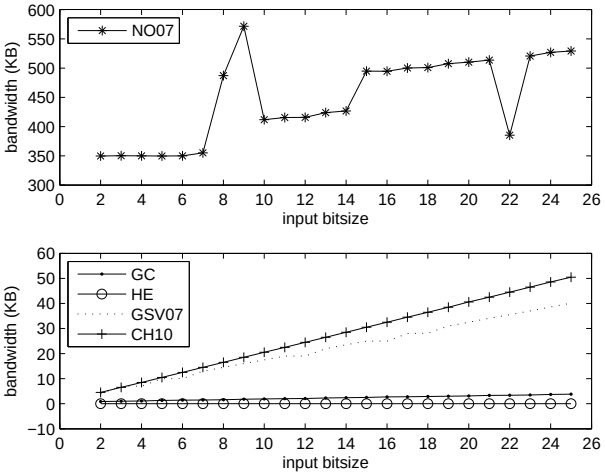

Fig. 5. Required bandwidth for precomputing

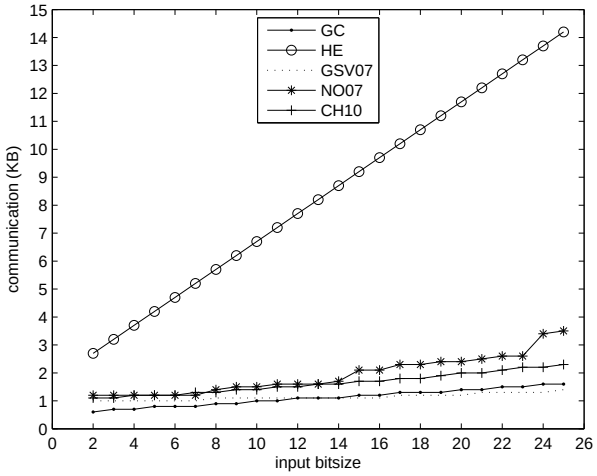

Fig. 7. Amount of transmitted data for online phase

ing when the number of comparisons grows. By extrapolating our results to $\ell = 112$, a precomputation time for GC of 2.18 seconds is expected. Within that same time frame, the precomputation for HE can be performed of inputs with bitsize $\ell = 10,219$, which equals the time for precomputing $10,219/\ell$ comparisons of inputs with bitsize ℓ. This shows that even with OT-extension, HE could often be preferred over GC, when considering only the precomputation effort. Especially since the precomputation phase of HE requires no data transmission at all.

C. On-line phase

Figure 6 presents the time required for the online computations for all of the protocols. All secret-sharing based solutions have a similar online execution time around 0.04 seconds, as well as the GC protocol. Due to the large size of encrypted numbers, the online execution time is somewhat more for HE, and grows more or less linearly with the bitsize.

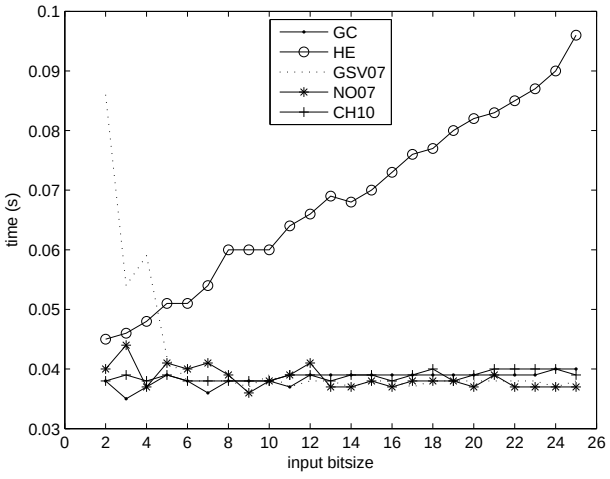

Fig. 6. Time for online phase

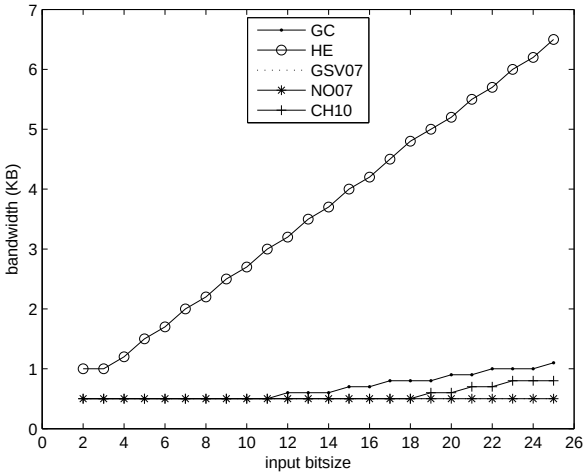

Fig. 8. Required bandwidth for online phase

For the amount of transmitted data during the online phase, as given in Figure 7, GSV07 outperforms GC slightly, while HE is the worst: the amount of data to be transmitted is significantly larger than any other protocol. This can easily be explained with the ciphertext size of the encryption scheme used in the protocol.

Figure 8 shows that the bandwidth requirement, on the other hand, for all protocols but HE is very similar; there is an increase in every protocol, while NO07 has a constant bandwidth demand. However, HE based protocol has the highest bandwidth demand, which linearly increases with the size of the inputs.

D. General Performance

We provide a summary in Table VI, where the effort for generating a safe prime in the initialisation phase of GC has not been counted. Figure 9 shows the total run-time of all protocols for inputs of 25 bits, considering the total time required for pre-computation and online computation:

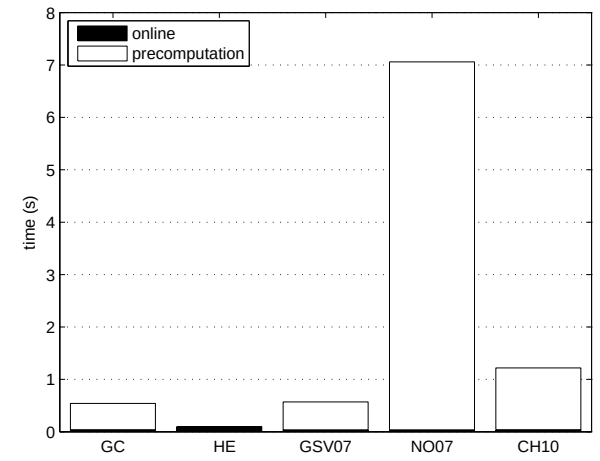

Fig. 9. Total time (25-bit inputs)

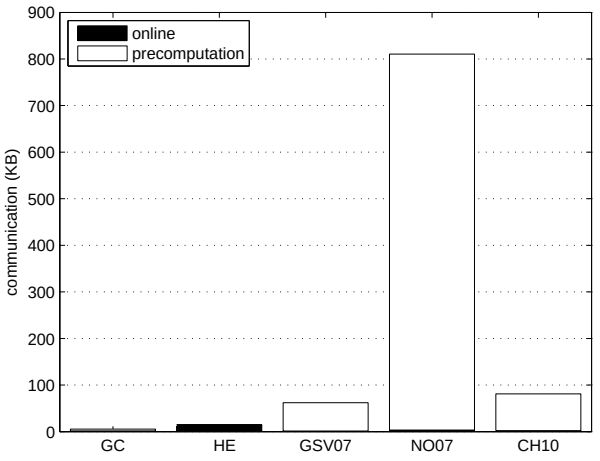

Fig. 10. Total amount of transmitted data (25-bit inputs)

the HE based protocol performs best. However, while all others have significant pre-computation load, the HE based protocol has a very dominant online computation demand. As explained earlier, when many secure comparisons have to be performed, OT-extension gives GC the opportunity to simultaneous precomputation, which considerably reduces its precomputation effort, and makes GC a preferred candidate more often.

As for the total amount of data transmitted for inputs of 25 bits, Figure 10 shows that the GC based protocol has the best performance, while the HE based protocol is significantly better than the secret sharing based protocols. While all others have an overwhelming amount of communication during pre-computation, for HE the main data transmission happens during the online phase.

VI. Conclusion

We implemented five different secure comparison protocols on a single platform. To assure interoperability with the SPED setting, the encrypted inputs were held by Bob, the client. We assumed inputs were limited to 25 bits, and both Alice and Bob would follow the rules of the protocol. We distinguished between an initialisation phase, which includes for example key generation, a precomputation phase, in which some values needed for one comparison can be precomputed, and an on-line phase which involves the actual comparison. In all three phases, we measured the execution time, and the communication complexity in terms of required bandwidth, amount of transmitted data, and number of communication rounds.

From the experiments follows that NO07 has the worst performance during the precomputation phase, but performs quite well in the online phase in terms of time and bandwidth. The remaining two secret-sharing protocols, GSV07 and CH10, have a good performance in the on-line phase, GSV07 performing a little better in terms of both execution time and communication. But in the precomputation phase, the performance of all three drops down, since many multiplication triplets have to be generated. Because of its linear round complexity, GSV07 requires much more communication rounds than CH10.

In terms of execution time, including precomputation, HE outperforms the other protocols, although GC needs a smaller amount of communication. However, when the application requires a high number of secure comparisons, a simultaneous precomputation of GC values by means of OT-extension provides additional gains, and makes the choice for GC more attractive. Both HE and GC need the smallest number of four communication rounds, although within GC, two of them can be precomputed.

Perfect security is achieved by the bitwise comparison protocols of NO07 and GSV07, and in one direction (Bob's input is perfectly save to Alice) in HE. Because of the additive blinding, CH10 achieves a slightly less statistical security. And finally, the inputs in GC, and Alice's input in HE, are only computationally safe.

Our results provide insight into which secure comparison protocol could be used, given a certain application. The actual choice will depend on different aspects like the round trip delay of the used communication technology (communication rounds), the available bandwidth, the maximally allowed execution time, whether the application allows a separate precomputation phase, the desired level of security, etc.

Acknowledgement

This work has been carried out in the Dutch COMMIT project.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[2] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey. Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain. *Signal Processing Magazine, IEEE*, 30(2):108–117, 2013.

TABLE VI
A SUMMARY OF THE PERFORMANCE RESULTS FOR 25-BIT INPUTS

	GC			HE			GSV07			NO07			CH10		
seconds / bandwidth in KB / rounds	time	data	#	time	data	#	time	data	#	time	data	#	time	data	#
Precomputation	0.5032	3.8	2	0.0052	0	0	0.5322	40	29	7.0228	529	14	1.1784	50	4
Online	0.0392	1.1	2	0.0922	6.5	4	0.0378	0.5	29	0.0374	0.5	13	0.0392	0.8	6
Total	0.5424	4.9	4	0.0974	6.5	4	0.57	40.5	58	7.0602	529.5	27	1.2176	50.8	10
Initialization	0.0071	0.7	1	154.9314	0.8	1	4.1242	0.8	1	4.1304	0.8	1	4.0336	0.74	1
Level of security	Computational			Alice: Perfect Bob: Computational			Perfect			Statistical			Perfect		

[3] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Privacy-preserving ecg classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.
[4] D. Beaver. Precomputing oblivious transfer. In *Advances in Cryptology - CRYPTO95*, volume 963 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1995.
[5] T. Bianchi and A. Piva. Secure watermarking for multimedia content protection: A review of its benefits and open issues. *IEEE Signal Process. Mag.*, 30(2):87–96, 2013.
[6] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In P. Rogaway, editor, *Advances in Cryptology - Crypto 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
[7] O. Catrina and S. de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *SCN*, pages 182–199, 2010.
[8] CBC. Federal agency loses data on 583,000 canadians. CBC News, 11 January 2013. Online, http://www.cbc.ca/news/canada/federal-agency-loses-data-on-583-000-canadians-1.1327172.
[9] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - Crypto 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, 2012.
[10] I. Damgård and T. Toft. Trading sugar beet quotas - secure multiparty computation in practice. *ERCIM News*, 2008(73), 2008.
[11] B. development team. Boost library. http://www.boost.org, 2013.
[12] M. development team. Multiple precision integers and rationals library. http://www.mpir.org, 2013.
[13] O. development team. Openssl library. https://www.openssl.org, 2013.
[14] S. development team. Secure computation library. http://cybersecurity.tudelft.nl/content/secomlib, 2013.
[15] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 235–253, Berlin, Heidelberg, 2009. Springer-Verlag.
[16] Z. Erkin, T. Veugen, T. Toft, and R. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Transactions on Information Forensics and Security*, 7:1053–1066, 06/2012 2012.
[17] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *PKC # 07*, volume 4450 of *LNCS*, pages 330–342, Berlin, 2007. Springer-Verlag.
[18] O. Goldreich. *Foundations of Cryptography. Basic Applications*, volume 2. Cambridge University Press, first edition, May 2004. ISBN 0-521-83084-2.
[19] I. Damgård and M. Geisler and M. Krøigaard. Homomorphic encryption and secure comparison. *Journal of applied cryptology*, 1(1):22–31, 2008.
[20] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer-Verlag, 2003.
[21] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, volume 5888 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2009.
[22] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *Journal of Computer Security*, 21(2):283–315, 2013.
[23] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *ICALP (2)*, pages 486–498, 2008.
[24] R. L. Lagendijk, Z. Erkin, and M. Barni. Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. *IEEE Signal Process. Mag.*, 30(1):82–105, 2013.
[25] E. MacAskill. NSA paid millions to cover prism compliance costs for tech companies. The Guardian, 23 August 2013. Online, http://www.theguardian.com/world/2013/aug/23/nsa-prism-costs-tech-companies-paid.
[26] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In *Proc. SIAM Symposium on Discrete Algorithms (SODA 2001)*, Jan. 2001.
[27] T. Nishide and K. Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In *PKC 2007*, volume 4450 of *LNCS*, pages 343–360. Springer-Verlag, 2007.
[28] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT*, pages 308–318, 1998.
[29] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.
[30] J. P. Prins, Z. Erkin, and R. L. Lagendijk. Anonymous fingerprinting with robust QIM watermarking techniques. *Eurasip Journal on Information Security*, 2007:1–7, 2007.
[31] N. Ramakrishnan, B. J. Keller, B. J. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62, 2001.
[32] T. Reistad and T. Toft. Linear, constant-rounds bit-decomposition. In *ICISC*, pages 245–257, 2009.
[33] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, pages 229–244, 2009.
[34] B. Schoenmakers and P. Tuyls. Efficient Binary Conversion for Paillier Encryptions. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 522–537. Springer-Verlag, 2006.
[35] A. Shamir. How to share a secret. In *Communications of the ACM, 22(11)*, pages 612–613, 1979.
[36] T. Toft. *Primitives and Applications for Multi-party Computation*. PhD dissertation, University of Aarhus, Denmark, BRICS, Department of Computer Science, 2007.
[37] J. Vaidya and C. Clifton. Privacy-preserving data mining: Why, how, and when. *IEEE Security & Privacy*, 2(6):19–27, 2004.
[38] T. Veugen. Encrypted integer division. In *IEEE Workshop on Information Forensics and Security*, Seattle, 12 2010. IEEE.
[39] T. Veugen. Improving the DGK comparison protocol. In *IEEE Workshop on Information Forensics and Security*, Tenerife, 12 2012. IEEE.
[40] A. C. Yao. Protocols for secure computations. In *Proc. of the 23th IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

APPENDIX A
GENERATING A SHARED RANDOM BIT

In the linear secret sharing variations, Alice and Bob need to generate shared random bits. To generate a shared random bit b the following protocol can be used:

1) Alice and Bob generate random bits b_A and b_B, respectively.
2) They both create a sharing of their bit by setting the other's share to zero, i.e. $[b_A] = (b_A, 0), [b_B] = (0, b_B)$.
3) They use a secure multiplication to compute $[b] = [b_A \oplus b_B] = [b_A] + [b_B] - 2[b_A b_B]$.

APPENDIX B
BITWISE SHARED RANDOM VALUE

Generating a bitwise shared random values means that Alice and Bob have to share a random value $r \in \mathbb{Z}_p$, and also share the $\lceil \log_2 p \rceil$ random bits r_i of r. Given the shared bits $[r_i]$, the shared random value can be computed locally through $[r] = \sum_i 2^i [r_i]$.

To guarantee $r < p$, Alice and Bob have to run a bitwise comparison protocol to compute $[(r < p)]$, and consequently open the comparison result. The steps above have to be repeated until the random number r doesn't exceed p. Since the random bits are uniformly distributed, the probability that $r < p$ equals $\frac{p}{2^{\lceil \log_2 p \rceil}}$, so on average $N_p = \frac{2^{\lceil \log_2 p \rceil}}{p} < 2$ attempts are needed.

In the protocol by Garay, Schoenmakers and Villegas [17], we use this approach to generate the bitwise shared random value u, and use their linear-round bitwise comparison protocol to test whether $u < p$. The protocol by Nishide and Ohta [27] however, is a *constant* round protocol. We use their constant-round bitwise comparison protocol to check $u < p$, but to guarantee a result within a constant number of rounds, Nishide and Ohta allow a negligable (smaller than $2^{-\kappa}$) probability of abortion, in case no suitable u has been found. This requires generating in parallel a large number of candidates for u.

APPENDIX C
GENERATING MULTIPLICATION TRIPLETS

The multiplication triplets have to be (pre)computed jointly by Alice and Bob. This could be achieved by using any additively homomorphic encryption system $[\![.]\!]$, but for efficiency reasons we chose Okamoto and Uchiyama [28]. We assume party 1 holds the private decryption key. The following protocol generates a shared triplet $([a], [b], [c])$:

1) Alice generates random $a_A, b_A \in \mathbb{Z}_p$, encrypts them and sends $[\![a_A]\!]$ and $[\![b_A]\!]$ to Bob.
2) Bob generates random $a_B, b_B \in \mathbb{Z}_p$, chooses a large random number r, computes $[\![a_A \cdot b_B + b_A \cdot a_B + r]\!] = [\![a_A]\!]^{b_B} \cdot [\![b_A]\!]^{a_B} \cdot [\![r]\!]$ and sends it to Alice.
3) Alice decrypts it and computes its share $c_A = a_A b_A + (a_A \cdot b_B + b_A \cdot a_B + r) \bmod p$.
4) Bob computes its share $c_B = a_B b_B - r \bmod p$.

Correctness is easily verified: $c_A + c_B = (a_A + a_B)(b_A + b_B) \bmod p$. The computation of a multiplication triplet requires mainly two encryptions and one decryption. Three encrypted values have to be communicated.

Thijs Veugen received two M.Sc. degrees in Mathematics and Computer Science, both Cum Laude, and the Ph.D. degree in Information Theory, all from Eindhoven University of Technology. After that, he worked as a Scientific Software Engineer at Statistics Netherlands, Heerlen, The Netherlands. Since 1999, he has been a Senior Scientist with the Information Security research group of the Technical Sciences department of TNO, The Hague, The Netherlands. He is also affiliated as a Senior Researcher with the Cyber Security group of Delft University of Technology, and has specialised in applications of cryptography. He has written many scientific papers on computing with encrypted data, serves frequently as a member of the program committee board of information security related conferences, and holds numerous related patents in various countries.

Frank Blom was born in the Netherlands in 1990. He received his B.Sc. and M.Sc. in Mathematics from the VU University, Amsterdam, Netherlands, in 2012 and 2014 respectively. He joined TNO Technical Sciences, in September 2013 as an intern for his graduation project. Since December 2014 he has been with Quintiq, where he is currently an R&D Support Consultant.

Sebastiaan de Hoogh is a Scientist at Philips Research Laboratories, Eindhoven, The Netherlands, in the department of Data Science. His reseacrh interests include applied cryptography, secure multiparty computation and whitebox cryptography. He received his Ph.D. degree from Eindhoven University of Technology.

Zekeriya Erkin is an assistant professor in the department of Intelligent Systems, Delft University of Technology. He is an expert in the field of secure signal processing, a sub-division of Privacy Enhanced Technologies. His research interest is in the direction of information security in cyber space and computational privacy as an approach to protect privacy- and/or commercially sensitive data. He received his B.Sc. and M.Sc. from Istanbul Technical University, Turkey, in 2002 and 2005, respectively. He received his Ph.D. degree in Secure Signal Processing from Delft University of Technology in 2010.

He is in the editorial board of Proceeding on Privacy Enhancing Technologies, in the technical committee of IEEE IFS, in the program committee of Privacy Enhanced Technologies (2014, 2015) and Smart Energy Grid Security Workshop 2014, general co-chair of PRETSELS Workshop (2015). He is also active in the technical committees of IEEE ICIP, IEEE ICASSP and ACM IH& MMSec.