Encrypted integer division and secure comparison

Thijs Veugen

Multimedia Signal Processing Group,
Delft University of Technology,
The Netherlands
and
Technical Sciences, TNO,
P.O. Box 5050,
2600 GB Delft, The Netherlands
E-mail: thijs.veugen@tno.nl

Abstract: When processing data in the encrypted domain, homomorphic encryption can be used to enable linear operations on encrypted data. Integer division of encrypted data however requires an additional protocol between the client and the server and will be relatively expensive. We present new solutions for dividing encrypted data in the semi-honest model using homomorphic encryption and additive blinding, having low computational and communication complexity. In most of our protocols we assume the divisor is publicly known. The division result is not only computed exactly, but may also be approximated leading to further improved performance. The idea of approximating the result of an integer division is extended to similar results for secure comparison, secure minimum, and secure maximum in the client-server model, yielding new efficient protocols with demonstrated application in biometrics. The exact minimum protocol is shown to outperform existing approaches.

Keywords: homomorphic encryption; integer division; comparison; minimum; maximum; approximation; client-server model; secure multi-party computations.

Reference to this paper should be made as follows: Veugen, T. (2014) 'Encrypted integer division and secure comparison, *Int. J. Applied Cryptography*, Vol. 3, No. 2, pp.166–180.

Biographical notes: Thijs Veugen received his two MSc degrees (Mathematics and Computer Science) (both Cum Laude), and his PhD degree in the field of Information Theory, all from Eindhoven University of Technology. After that, he worked as a Scientific Software Engineer at Statistics Netherlands. Since 1999, he has been a Senior Scientist in the Information Security research group of the Technical Sciences Department of TNO, Delft, The Netherlands. He is also affiliated as a Senior Researcher with the Multimedia Signal Processing group of Delft University of Technology, and has specialised in applications of cryptography.

This paper is a revised and expanded version of a paper entitled 'Encrypted integer division' presented at IEEE Workshop on Information Forensics and Security, Seattle, December 2010.

1 Introduction

When processing data in the encrypted domain, homomorphic encryption can be used to enable linear operations on encrypted data. Integer division of encrypted data however requires an additional protocol with the server and will be relatively expensive. It is needed in many applications like secure clustering (Bunn and Ostrovsky, 2007; Erkin et al., 2009b), secure personal recommendation (Erkin et al., 2010), SFR (Erkin et al., 2009a), secure statistical analysis (Guajardo et al., 2009), secure auctions (Naor et al., 1999; Bogetoft et al., 2009), but also for computing the Levenshtein distance (Rane and Sun, 2010). An important application of integer division is unpacking. When processing data in the encrypted domain, it is computationally and communicatively interesting to pack several samples into one encryption (Bianchi et al., 2009b; Erkin et al., 2012). It enables simultaneous execution of linear operations on all samples that are packed into that encryption. However, for more complicated operations, the samples have to be unpacked, which requires an additional protocol for integer division.

We present new solutions for dividing encrypted data, having low computational and communication complexity. One for computing exact division by a public divisor, and two for approximated division result, with public and private divisor respectively.

Furthermore, we show that encrypted integer division is closely related to secure comparison in the client-server model, developing new protocols for securely approximating the comparison result, and the minimum (or maximum). The minimum (or maximum) is approximated in two ways, either in terms of likelihood, or accuracy. These three new protocols expand the results in Veugen (2010) and are optimised further. The

approximation protocols take the computational, but especially the communication complexity, to a lower level than their exact counterparts. Typical applications would be secure clustering, statistical analysis, and all kinds of biometrical applications where all (intermediate) results are either heuristic or estimated, so approximating minimums within a certain accuracy will not degrade the end result.

1.1 Preliminaries

We consider the client-server model where party A, the client, has some encrypted number $[x]$, and the server B has the decryption key K. Party A would like to divide the integer x by some integer d. Both A and B are not allowed to learn the number x. The divisor d could be public, but could also be privately known to the server B. We assume the semi-honest model where A and B follow the rules of the protocol, but collect as much information as possible to deduce private information.

More precisely, A has an encrypted number $[x]$, $0 \le x < N$, in the field Z_N^*. A cannot decrypt but would like to divide the encrypted number $[x]$ by a number d, $0 < d < N$. More specifically, he wants to find the encrypted number $[x \div d]$, such that $x = d \cdot (x \div d) + (x \bmod d)$, where $x \div d \ge 0$ and $0 \le x \bmod d < d$.

We use $\log_2 x$ to denote the base two logarithm of a positive integer x as a measure for the number of bits of x. For convenience we ignore the fact that the result should be rounded upwards to the closest integer as this logarithm is usually not integer valued.

Homomorphic encryption, denoted by [.], is used to encrypt the data represented by integers. Any additively homomorphic encryption system could be used, as long as it is semantically secure. We use the Paillier (1999) cryptosystem throughout our protocols where the integer N equals the product of two large primes. Plaintexts in the Paillier cryptosystem are computed modulo N, but cipher texts are computed modulo N^2. We recall an important property of additively homomorphic encryption systems, namely that for integers x and y we have $[x] \cdot [y] = [x + y] \bmod N^2$. To emphasise that our protocols work with any additive homomorphic encryption system, we omit the reduction modulo N^2 throughout cipher text operations in our protocols. However, when computational or communication complexities of our protocols are depicted, an instantiation with Paillier is assumed.

The multiplicative inverse of x modulo N^2 is denoted by x^{-1} and equals the integer y, $0 \le y < N^2$, such that $x \cdot y = 1 \bmod N^2$. The multiplicative inverse is efficiently computed by using the Euclidean algorithm (Menezes et al., 1996), and can also be used to negate an encrypted integer: $[-x] \leftarrow [x]^{-1} \bmod N^2$. To estimate the computational complexity of the different protocols, we use the fact that an involution modulo N^2 with exponent n will on average take $\frac{3}{2}\log_2 n$ multiplications modulo N^2. A Paillier decryption costs around $\frac{3}{8}\log_2 N$ multiplications modulo N^2 [through Chinese remaindering and counting four multiplications mod p^2 as one multiplication mod N^2 as shown in Paillier (1999)], and a Paillier (1999) encryption only two multiplications modulo N^2 when the random factor is precomputed. The complexity of the most basic operations is shown in Table 1 and phrased in number of multiplications (NoM) modulo N^2, an entity that will be used to express the computational complexity of all protocols. To abstract from Paillier we will count separately the NoM for the encryptions (NoM_{Enc}) and decryptions (NoM_{Dec}) and only use Paillier specific NoM's when displaying the complexities in figures.

Table 1 NoM for basic operations

Operation	*NoM mod* N^2
Paillier encryption ($\mathrm{NoM}_{Enc}(1)$)	2
Paillier decryption ($\mathrm{NoM}_{Dec}(1)$)	$\frac{3}{8}\log_2 N$
Negation: $[-x] \leftarrow [x]^{-1} \bmod N^2$	1
Involution: $[x]^n \bmod N^2$	$\frac{3}{2}\log_2 n$

The communication complexity will be expressed in the NoC encryptions, assuming each encryption contains $2 \log_2 N$ bits like in the Paillier cryptosystem. When naming variables in our protocols we use the symbol ∅ to denote an (approximate) division result. Let σ be the statistical security parameter, whose value is usually chosen around 80 (bits). When estimating the computational complexity of our protocols, we assume N is a 2,048 bit number. Furthermore, it is assumed that all random variables, excluding the inputs of the secure multi-party computation protocol, are uniformly chosen, and that encrypted values are rerandomised by both parties when necessary.

As a subprotocol we often use a private comparison protocol from the secure multi-party computation setting, which means that each party has its private input in plaintext. Any private comparison protocol with encrypted output could be used here, including a solution based on garbled circuits (GCs) (Kolesnikov et al., 2009), as long as it is secure in the semi-honest model. A commonly used private comparison protocol in signal processing and other domains is by Damgård, Geisler and Krøigaard (DGK) (Damgård et al., 2008, 2009; Erkin et al., 2009a), which we will use to give a fair comparison of the computational and communication complexity with existing solutions. The number of modular computations of the DGK comparison protocol given ℓ, the number of input bits, is given below. They use a dedicated cryptosystem instead of the Paillier cryptosystem, and since its modulus is N instead of N^2 we corrected NoM by a factor 4, and NoC by a factor 2. We assume all (re)randomisations are precomputed.

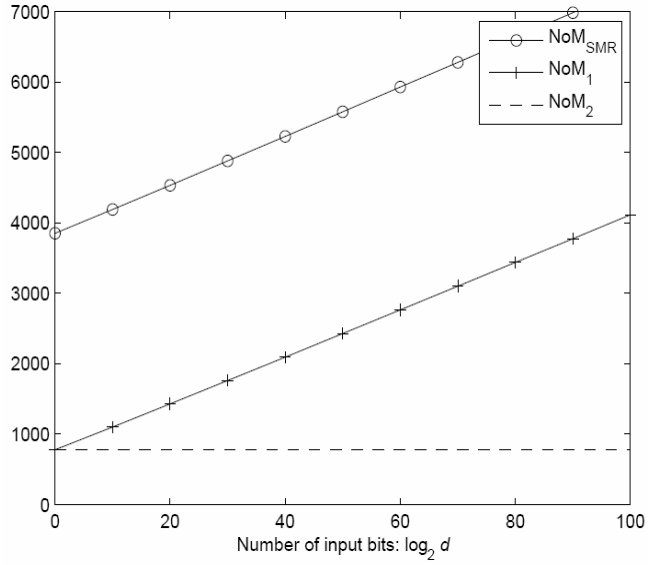
$$\begin{aligned}\mathrm{NoM}_{DGK}(\ell) &= \mathrm{NoM}^{A}_{DGK}(\ell) + \mathrm{NoM}^{B}_{DGK}(\ell)\\ &= \{(7/8)\ell + (3/8)\ell \log_2(\ell + 2)\} + 30\ell\\ \mathrm{NoC}_{DGK}(\ell) &= \ell\end{aligned}$$

The DGK protocol consists of two communication rounds in which $2\ell + 1$ encryptions (of size $\log_2 N$) are sent to the other party.

1.2 Related work in secure integer division

The problem of dividing an encrypted number has been studied recently in different settings. Much prior work is based on computing integer division through secure modulo reduction (SMR). We present a way to directly compute the integer division result leading to lower computational complexity as depicted in Figure 1.

Figure 1 NoM for encrypted integer division



Schoenmakers and Tuyls (2006) present a method, based on threshold homomorphic crypto, to convert an encrypted integer to its encrypted bits. One of their results is a gate that securely computes the least significant bits (LSBs). It can be used to compute $[x \bmod 2^m]$ for some known integer m, and from that $[x \div 2^m]$, thus it enables the division of an encrypted integer by a known power of 2. Toft (2007) shows how to reduce a secretly shared integer with respect to a known modulus, thereby generalising the results of Schoenmaker and Tuyls to arbitrary, but publicly known divisors. He also extends this result to a secretly shared modulus, using a secure comparison protocol among others.

Damgård et al. (2006) present secure protocols for modulo reduction of secretly shared integers within constant rounds. The modulus is either public or secretly shared. This SMR could consequently be used to compute the division.

Catrina and Saxena (2010) work out a way of securely computing with rational numbers using a fixed-point representation. A linear secret sharing scheme is used to protect the inputs and the output. Part of this framework is a truncation protocol that is capable of secure division with a known power of two. Like previous solutions mentioned above, it computes integer division through SMR. Therefore, when their solution were converted to our setting, it would suffer from the same efficiency problems.

While processing homomorphically encrypted data, packing is used to reduce communication and computational complexity. Bianchi et al. (2009a, 2009b) show how to unpack an encrypted integer that represents a concatenation (i.e., packing) of (sensitive) signals. A division protocol is presented to efficiently and securely divide an encrypted value by a known divisor, via SMR. A similar protocol for reducing an encrypted number with respect to a known modulus was simultaneously found by Guajardo et al. (2009), and was used by Erkin et al. (2009a) to compute the minimum. This protocol is further improved in our Subsection 3.1 by avoiding an intermediate modular result.

Bunn and Ostrovsky (2007) describe a protocol for securely clustering two databases. They propose a subprotocol that securely computes the division of two shared integers, using ideas of long division. Since their denominator is also a secret value, the complexities of our division protocols are much lower.

Schröpfer et al. (2011) describe a compiler language for solving secure multi-party computations either with homomorphic encryption or GCs and compare different approaches for integer division. In their setting, each player has a private input whereas in our client-server model the client has all inputs encrypted. This and similar solutions (Henecka et al., 2010; Ben-David et al., 2008; Lazzeretti and Barni, 2011) are further discussed in Subsection 4.6.

Atallah et al. (2004) present secure protocols for integer division, either with an additively shared or a public denominator. Their scaling protocol (integer division by a known constant) is similar to our Protocol 1, except that they use an oblivious transfer instead of a secure comparison. The main difference is the setting since their input and output are additively shared between two parties, and ours are encrypted. They do not consider approximations.

Our approximation protocols are related to the field of secure multi-party computation of approximations (Feigenbaum et al., 2006), where generally hard problems are relaxed, but still securely computed. Integer division is not such a typical hard problem. Furthermore, in our client-server model the client's inputs are encrypted as well as his output, whereas in the general multi-party computation setting each party has his own privately known input, and the output will be known to some of the parties so the (approximated) output value might leak some information about the inputs. Encrypted integer division is considered as one of the possibly many computations to be performed when processing encrypted signals and only at the very end some values will be decrypted.

Protocol 1 Exact integer division with public divisor

Party	A	B
Input	$[x]$ and d	d and K
Output	$[x \div d]$	
Constraints	$0 < x < N \cdot 2^{-\sigma}$ and $0 < d < N$	
NoM_1	$\mathrm{NoM}_{Enc}(3) + \mathrm{NoM}_{Dec}(1) + 4 + \mathrm{NoM}_{DGK}(\log_2 d)$	
NoC_1	$2 + \mathrm{NoC}_{DGK}(\log_2 d)$	

1 A chooses a random number r of $\log_2 N - 1$ bits, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

2 B decrypts $[z]$, and computes $z \bmod d$.

3 A and B perform a private comparison protocol. The input of A is $r \bmod d$, the input of B is $z \bmod d$, and the output for A will be the encrypted bit $[t]$ such that $\{t = 1\} = \{z \bmod d < r \bmod d\}$.

4 B computes $z \oslash d \leftarrow z \div d$, and sends it to A encrypted.

5 A computes $r \oslash d \leftarrow r \div d$, encrypts it, and computes $[x \div d] - [(z \div d) - (r \div d) - t] = [z \oslash d] \cdot ([r \oslash d] \cdot [t])^{-1}$.

Jakobsen (2006) describes a couple of methods for dividing two secretly shared integers. Also approximations are considered, but all methods use some kind of (expensive) binary search. A similar exact approach for Paillier encrypted integers was recently found by Dahl et al. (2012).

Kiltz et al. (2005) present an oracle-aided protocol for computing the mean of several database entries, which is described as a way of securely approximating the division of two additively shared values, namely the nominator and the denominator. Their main primitive is an oblivious polynomial evaluation which requires an amount of exponentiations and communications linear in the size of the denominator. Since their denominator is also a secret value, the complexities of our division protocols are much lower.

Another known approach when an approximated output is sufficient is the use of random perturbation (Adam and Wortmann, 1989; Verykios et al., 2004). Random perturbation in general has the problem of information leakage, which can be reduced by greatly increasing the number of inputs like in a database. Since we consider the problem of dividing only one number and not an entire database of numbers, this technique is not suitable to our setting.

1.3 Organisation of the paper

In Section 2 we show a protocol for secure integer division based on additive homomorphic encryption and additive blinding that uses a private comparison protocol as a subprotocol. The divisor is known to both parties.

For reduced complexity, in Section 3 two protocols are presented that approximate the result of division by simply eliminating the private comparison protocol. In the first protocol as described in Subsection 3.1 the divisor is publicly known, whereas the second protocol from Subsection 3.2 only requires the divisor to be privately known to party B.

Section 4 is dedicated to secure comparison of encrypted integers and especially to approximating the comparison result for reducing the complexity which is achieved by reducing the size of the inputs of the private comparison subprotocol. This leads to two new protocols for approximating the minimum of two encrypted integers in Subsection 4.3 in terms of likelihood and accuracy respectively. In the remainder of Section 4 our results are explicitly compared to the secure minimum protocol (Atallah et al., 2003) and the GC approach.

All protocols are formally proven secure in the semi-honest model in Section 5.

2 Exact division with public divisor

We describe an efficient solution for the exact computation of $[x \div d]$ from $[x]$, given the public divisor d, $0 < d < N$. This solution requires a private comparison protocol as a subprotocol.

An efficient way for A to compute $[x \div d]$ from $[x]$ and d, is to use additive blinding. Because B is not allowed to learn the value x, it is additively blinded by a random number r of (at least) $\log_2 x + \sigma$ number of bits, where σ, usually in the order of 80 bits, is the statistical security parameter. It leads to Protocol 1, where the random number r is chosen as large as possible to ensure the best statistical hiding of x.

The correctness of the computation of $x \div d$ follows from the observation that $z = d \cdot (z \div d) + (z \bmod d)$, and $z = x + r$, so $z \div d = (x \div d) + (r \div d)$, exactly when $(x \bmod d) + (r \bmod d) < d$, and $z \div d = (x \div d) + (r \div d) + 1$, otherwise. The condition $(x \bmod d) + (r \bmod d) < d$ is equivalent to $(z \bmod d) = (x \bmod d) + (r \bmod d)$, i.e., $(z \bmod d) \geq (r \bmod d)$. This analysis leads to equation (1), where t is the binary result of the comparison $(x + r) \bmod d < r \bmod d$.

$$(x + r) \div d = (x \div d) + (r \div d) + t \tag{1}$$

Since the addition of x and r is not allowed to initiate a carry-over modulo N, Protocol 1 only works when x is not too large: $\log_2 x < \log_2 N - \sigma$. It is possible to extend this protocol such that x can attain arbitrary integer values between 0 and N, but this requires at least an extra comparison protocol ($z < r$) to determine whether a carry-over has occurred. A similar approach is illustrated in Protocol 2.1.

Protocol 1 requires one execution of the private comparison protocol, with inputs of $\log_2 d$ bits. The remaining steps only require two encryptions, three modular multiplications and one inversion by A, and one decryption and one encryption by B. This method outperforms the division protocols of Erkin et al. (2009a), Guajardo et al. (2009), Bianchi et al. (2009a, 2009b), Schoenmakers and Tuyls (2006), Toft (2007) and Damgård et al. (2006) based on SMR, which require at least an extra involution to the power d^{-1} to compute $[x \div d]$ from $[x \bmod d]$, roughly needing an extra $\frac{3}{2}\log_2 n$ multiplications modulo N^2. The

gain in computational complexity is depicted in Figure 1, where NoM_{SMR} denotes the NoM for the SMR approach which forms the heart of the cited previously published division protocols.

The number of communication rounds for Protocol 1 is constant, whenever the used private comparison protocol has constant rounds complexity.

In Veugen (2010) a second protocol is described computing exact division by dividing the plain text interval into d bins each corresponding with a particular division result of the input. A secure binary search then enables finding the division result. Although this protocol is less attractive from a complexity point of view it only requires d to be privately known to party B.

3 Approximate division

Instead of exactly computing (the encrypted) $x \div d$, as argued in the introduction it might be enough for some applications to approximate $x \div d$. This approach is interesting since it leads to more efficient solutions that do not require a private comparison protocol as a subprotocol. The lack of a private comparison subprotocol can lead to a significant reduction of NoM, as depicted Figure 1, especially for larger divisors. It especially reduces the communication complexity and the number of communication rounds due to the lack of a private comparison subprotocol.

We present two protocols for approximating $x \div d$, one in which d is publicly known, and one in which d is privately known to B, and A only knows its number of bits $\log_2 d$.

3.1 Approximate division with public divisor

This approach is derived from the blinding approach described in Section 2. There, the comparison result t is used to slightly correct the division result. The analysis in Section 2 shows that $x \oslash d = (z \div d) - (r \div d)$ is a good approximation of $x \div d$, since either $x \oslash d = x \div d$ or $x \oslash d = (x \div d) + 1$, depending on the outcome of the comparison. This leads to Protocol 2.

The computational complexity of Protocol 2 is depicted in Figure 1 and shows a considerable gain with respect to exact division. Although unpacking protocols require larger divisors, we only display the results for divisors up to 100 bits in our figures which should be sufficient for most data processing algorithms.

In order to avoid limitations on the length of x, the protocol could be extended to Protocol 2.1. The comparison result c will inform A whether a carry-over has occurred during the computation of z, in which case $z = x + r - N = x - r_c$, and thus $x \div d \approx (z \div d) + (r_c \div d) = (z \oslash d) + (r_c \oslash d)$. When no carry-over has occurred, i.e., $c = 0$, then $z = x + r$, so $x \div d \approx (z \div d) - (r \div d) = (z \oslash d) - (r \oslash d)$. The computation of s by A assures that $s = -(r \oslash d)$ when $c = 0$, and $s = r_c \oslash d$ otherwise. Therefore, $x \oslash d = (z \oslash d) + s$ will be a good approximation for $x \div d$. In fact, $x \oslash d = (x \div d) \pm t$ where t is the result of comparing z mod d with either r mod d or r_c mod d.

Protocol 2 Appr. div. with public divisor and input constraints

Party	A	B
Input	$[x]$ and d	d and K
Output	$[x \oslash d]$	
Constraints	$0 \le x < N \cdot 2^{-\sigma}$ and $0 < d < N$ and $x \oslash d \in \{x \div d, (x \div d) + 1\}$	
NoM_2	$\text{NoM}_{Enc}(2) + 2$	$\text{NoM}_{Enc}(1) + \text{NoM}_{Dec}(1)$
NoC_2	1	1

1 A chooses a random number r of $\log_2 N - 1$ bits, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

2 B decrypts $[z]$, computes $z \oslash d \leftarrow z \div d$, encrypts it, and sends $[z \oslash d]$ to A.

3 A computes $r \oslash d \leftarrow r \div d$, encrypts $-(r \oslash d)$, and computes $[x \oslash d] \leftarrow [z \oslash d] \cdot [-(r \oslash d)]$.

Protocol 2.1 Appr. div. with publ. divisor without input constr.

Party	A	B
Input	$[x]$ and d	d and K
Output	$[x \oslash d]$	
Constraints	$0 \le x < N$, $0 < d < N$, and $x \oslash d \in \{(x \div d) - 1, x \div d, (x \div d) + 1\}$	
$\text{NoM}_{2.1}$	$\text{NoM}_{Enc}(3) + \text{NoM}_{Dec}(1) + 3 + \frac{3}{2}\log_2(N/d) + \text{NoM}_{DGK}(\log_2 N)$	
$\text{NoC}_{2.1}$	$2 + \text{NoC}_{DGK}(\log_2 N)$	

1 A chooses a random number r, $0 \le r < N$, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

2 B decrypts $[z]$, computes $z \oslash d \leftarrow z \div d$, encrypts it, and sends $[z \oslash d]$ to A.

3 A and B perform a private comparison protocol. The inputs of A and B are r and z resp., and A's output will be the encrypted bit $[c]$ such that $\{c = 1\} \equiv \{z < r\}$.

4 A computes $r \oslash d \leftarrow r \div d$, $r_c \leftarrow N - r$, $r_c \oslash d \leftarrow r_c \div d$, encrypts $-(r \oslash d)$, and computes $[s] \leftarrow [c]^{(r \oslash d) + (r_c \oslash d)} \cdot [-(r \oslash d)]$.

5 A computes $[x \oslash d] \leftarrow [z \oslash d] \cdot [s]$.

The main drawback of this extended protocol is the extra private comparison protocol, whose absence was the big advantage of using an approximation instead of an exact division result. Although the security towards B is increased (see Section 5), it does not seem worthwhile in practice, but nicely illustrates the idea of perfect blinding thereby achieving information theoretic security instead of statistical security towards B.

3.2 Approximate division with private divisor

Instead of having d publicly known, Protocol 2 can be modified to the case where d is only known to B. In such a setting, A is no longer able to compute $r \div d$ from an

arbitrary random number r. But by constructing r as $r = r_d \cdot d + r_m$, we have $r \div d \approx r_d$ when r_m has the same size as d. This leads to Protocol 3.

Protocol 3 Approximate division with private divisor

Party	A	B
Input	$[x]$	d and K
Output	$\log_2 d$ and $[x \oslash d]$	
Constraints	$0 \le x < N \cdot 2^{-\sigma}$, $0 < d < N$, and $x \div d \le x \oslash d \le (x \div d) + 2$	
$\mathrm{NoM}_{2.1}$	$\mathrm{NoM}_{Enc}(2) + 3 + \frac{3}{2}\log_2 \frac{N}{d}$	$\mathrm{NoM}_{Enc}(2) + \mathrm{NoM}_{Dec}(1)$
$\mathrm{NoC}_{2.1}$	1	2

1 B encrypts d, and sends $[d]$ and $\log_2 d$ to A.

2 A chooses random numbers r_d and r_m of $\log_2 N - 1 - \log_2 d$ and $\log_2 d$ bits respectively. A encrypts r_m and computes $[r] \leftarrow [r_d \cdot d + r_m] = [d]^{r_d} \cdot [r_m]$.

3 A computes $[z] \leftarrow [x + r] = [x] \cdot [r]$ and sends it to B.

4 B decrypts $[z]$, computes $z \oslash d \leftarrow z \div d$, encrypts it, and sends $[z \oslash d]$ to A.

5 A encrypts $-r_d$ and computes $[x \oslash d] \leftarrow [(z \oslash d) - r_d] = [z \oslash d] \cdot [-r_d]$.

Due to the sizes of r_d and r_m, we know that number r contains at least $\log_2 x + \sigma$ random bits, which ensures that z statistically hides x towards B. Since r_m contains $\log_2 d$ bits, we derive $0 \le r_m < 2d$, so $(r \div d) \le r_d \le (r \div d) + 1$. Therefore $x \oslash d = (z \div d) - r_d$ is a good approximation of $x \div d$, in fact $(x \div d) \le x \oslash d \le (x \div d) + 2$.

Similar to Protocol 2.1, the limitations on the size of x can be eliminated at the cost of an extra private comparison protocol.

Protocol 3 is more intensive than Protocol 2, since it requires an additional involution by A, and one additional encryption by B. The involution by A takes roughly $\frac{3}{2}(\log_2 N - 1 - \log_2 d)$ modular multiplications.

Both approximation protocols have a constant number of communication rounds.

4 Secure comparison of two encrypted inputs

In our protocols for encrypted integer division, a *private* comparison protocol with private inputs is often used as a subprotocol. When computing *secure* comparison in the client-server model, party A has two encrypted integers $[a]$ and $[b]$, both upper bounded by d, and wants to compute the encrypted bit $[t]$ which is one exactly when $a \le b$, and zero otherwise. This is denoted by $t = (a \le b)$. The relation with encrypted integer division is shown in equation (2).

$$(a \le b) = (d + b - a) \div d \tag{2}$$

Since a and b are upper bounded by d, it is clear that $0 \le x = d + b - a < 2d$. If $a \le b$ then $x = d + b - a \ge d$ and $t = 1$, and otherwise $x < d$ and $t = 0$, so the correctness easily follows. The only restriction is that an upper bound d on a and b should be known, which is not only a common assumption in secure multi-party computations, but also feasible for most applications. This shows that Protocol 1 can also be used for secure comparison in the client-server model with $[x] \leftarrow [d + b - a] = [d] \cdot [b] \cdot [a]^{-1}$. In that case, Protocol 1 can be considered as an efficient transformation of secure comparison in the client-server model (with encrypted inputs) to private comparison with *private* inputs.

The gain towards other known transformations from secure to private comparison (Erkin et al., 2009a; Guajardo et al., 2009; Bianchi et al., 2009a, 2009b; Schoenmakers and Tuyls, 2006; Toft, 2007; Damgård et al., 2006) is, as in exact division, the lacking of SMR. A transformation based on SMR works as follows:

1 A chooses a random number r of $\log_2 N - 1$ bits, encrypts it, and computes $[x] \leftarrow [d + b - a] = [d] \cdot [b] \cdot [a]^{-1}$ and $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

2 B decrypts $[z]$, and computes $z \bmod d$.

3 A and B perform a private comparison protocol. The input of A is $r \bmod d$, the input of B is $z \bmod d$, and the output for A will be the encrypted bit $[t]$ such that $\{t = 1\} \equiv \{z \bmod d < r \bmod d\}$.

4 B encrypts $z \bmod d$ and sends it to A.

5 A encrypts $r \bmod d$ and computes $[x \bmod d] \leftarrow [z \bmod d] \cdot [r \bmod d]^{-1} \cdot [t]^{d}$.

6 A computes $[(a < b)] \leftarrow [x \div d] = ([x] \cdot [x \bmod d]^{-1})^{1/d}$.

Here $1 / d$ denotes the multiplicative inverse of d modulo N. The complexities of this transformation, excluding the private comparison protocol with private inputs, are

$$\mathrm{NoM}_{SMR} = \mathrm{NoM}_{Enc}(3) + \mathrm{NoM}_{Dec}(1) + 9 + \frac{3}{2}\log_2 N + \frac{3}{2}\log_2 d \quad \text{and} \quad \mathrm{NoC}_{SMR} = 2.$$

Although the communication complexity is identical, the computational complexity clearly exceeds the $\mathrm{NoM}_{Enc}(3) + \mathrm{NoM}_{Dec}(1) + 7$ multiplications needed for the transformation in Protocol 1 (including the computation of $[x]$ from $[a]$ and $[b]$).

Given the relation between encrypted integer division and secure comparison, one could try to reduce computations by 'approximating' the comparison result through an approach similar to Protocol 2. In the following subsection is shown how this results in a new protocol for securely determining the comparison result with high probability of correctness. Subsequently, this is extended further to new protocols for securely approximating the minimum of two values. In each of these protocols, a private comparison protocol with private inputs is used as a subprotocol. The advantage of the 'approximated' versions with respect to their exact counterparts, which are all based on Protocol 1, is the reduced size of the inputs needed for the private comparison subprotocol. All private comparison

protocols process the inputs bit by bit and each additional bit requires an additional step during the computation.

The idea behind the three new Protocols 4, 5 and 6 is that only a few most significant bitplanes are tested. If the most significant differing bit position lies within the number of tested most significant bits, then the comparison bit or minimum will be found exactly. If, instead, the most significant differing bit position does not lie within the number of tested most significant bits, then whatever the comparison bit or minimum found, it will have a bounded error.

Although we chose DGK, *any* private comparison protocol could be used as a subprotocol in our solutions. In general, the computational and communication complexity of a private comparison protocol is (at least) linear in the number of input bits, so our 'approximated' versions reduce the complexity irrespective of the chosen private comparison protocol. This even holds for a GC-based implementation as shown hereafter.

As an example, we show how our new protocols can be applied to the SFR application to significantly increase its performance.

4.1 Probabilistic guarantees for $a \le b$

To illustrate 'approximated' secure comparison, assume we have two ℓ bit numbers α and β, where α has an arbitrary random distribution but β is independently (from α) uniformly distributed. Denote their ℓ bits by α_i and β_i respectively. When comparing α and β by scanning the bit strings $\alpha_{\ell-1} \ldots \alpha_0$ and $\beta_{\ell-1} \ldots \beta_0$ from left to right, one is actually computing bits $t^i = (\alpha_{\ell-1} \ldots \alpha_i \le \beta_{\ell-1} \ldots \beta_i)$ from $i = \ell - 1$ to $i = 0$. The final result is $t = t^0 = (\alpha \le \beta)$, but the intermediate results $\tilde{t} = t^i$ will equal t with high probability because the most significant bits of α and β are used. In particular, when $t^i = 1$ for a particular i, i.e., $\alpha_{\ell-1} \ldots \alpha_i \le \beta_{\ell-1} \ldots \beta_i$, then definitely $t^j = 1$ for all j, $0 \le j \le i$. And when $t^i = 0$, this value can only change to $t^0 = 1$ when all the $\ell - i$ most significant bits equal: $\alpha_{\ell-1} \ldots \alpha_i = \beta_{\ell-1} \ldots \beta_i$ which will occur with probability 2^{i-l}. This concept can be generalised by using $\alpha \div d$ for any $0 < d$ instead of $\alpha_{\ell-1} \ldots \alpha_i = \alpha \div 2^i$, but to simplify the analysis we will use only powers of two.

The above idea of approximating $t = (a \le b)$ by a likely identical bit $\tilde{t}$ is worked out in Protocol 4, where we used a combination of equation (2) and Protocol 2, and approximated ($z \bmod d < r \bmod d$) in line 3 of Protocol 2 by $\tilde{t}$. Because the size of the inputs of the private comparison subprotocol is reduced from d to d', significant savings can be achieved.

From the earlier analysis, it follows that $\tilde{t}$ is likely equal to ($z \bmod d < r \bmod d$). In particular, they can only differ when their $D' = \log_2 d'$ most significant bits are equal, i.e., $y_A = y_B$. By construction, $y_A = r_{D-1} \ldots r_{D-D'}$, and since r has been uniformly chosen, each value of y_A is equally likely, so $\Pr\{y_A = y_B\} = \frac{1}{d'} = 2^{-D'}$. Therefore,

$$\begin{aligned}
&\Pr\{t = (a \le b)\} \\
&= \Pr\{t = (x \div d)\} \\
&= \Pr\{\tilde{t} = ((z \bmod d) < (r \bmod d))\} \ge 1 - \Pr\{y_A = y_B\} \\
&= 1 - \frac{1}{d'}.
\end{aligned}$$

Protocol 4 Probabilistic guarantees for $a \le b$

Party	A	B
Input	$[a]$, $[b]$, d, and d'	d, d', and K
Output	$[t]$, where $\Pr\{t = (a \le b)\} \ge 1 - \frac{1}{d'}$	
Constraints	$0 \le a, b < d$, and $0 < d' < d < N \cdot 2^{-(\sigma+1)}$; $d = 2^D$, $d' = 2^{D'}$	
NoM_4	$\mathrm{NoM}_{Enc}(3) + \mathrm{NoM}_{Dec}(1) + 7 + \mathrm{NoM}_{DGK}(D')$	
NoC_4	$2 + \mathrm{NoC}_{DGK}(D')$	

1	A encrypts d and computes $[x] \leftarrow [d + b - a] = [d] \cdot [b] \cdot [a]^{-1}$.
2	A chooses a random number r of $\log_2 N - 1$ bits, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.
3	B decrypts $[z]$, and computes $z \bmod d$.
4	A and B perform a private comparison protocol. The input y_A of A consists of the D' most significant bits $r_{D-1} \ldots r_{D-D'}$ of $r \bmod d$, the input y_B of B similarly consists of the bits $z_{D-1} \ldots z_{D-D'}$ and the output for A will be the encrypted bit $[\tilde{t}]$ such that $\tilde{t} = (y_B < y_A)$.
5	B computes $z \varnothing d \leftarrow z \div d$, encrypts it, and sends $[z \varnothing d]$ to A.
6	A computes $r \varnothing d \leftarrow r \div d$, encrypts it, and computes $[t] \leftarrow [(z \div d) - (r \div d) - \tilde{t}] = [z \varnothing d] \cdot ([r \varnothing d] \cdot [\tilde{t}])^{-1}$.

It is important to note that this is independent of the random distribution of the variables a and b.

The complexities of Protocol 4 are similar to Protocol 1 (with $\log_2 d = D'$). A drawback of Protocol 4 is that the computed output value t is not necessarily zero or one. In case $a \le b$, $y_A \le y_B$, and ($z \bmod d$) < ($r \bmod d$), then $\tilde{t} = 0$, so $t = (z \varnothing d) - (r \varnothing d) = (z \div d) - (r \div d)$, which by equation (1) equals $(x \div d) + 1 = (a \le b) + 1 = 2$. This can easily be overcome by computing one small encrypted integer division (or secure comparison) as a final step: $[t] \leftarrow [(t + 1) \div 2]$, or using the optimisation as described in Subsection 4.4.

4.2 Secure face recognition

Protocol 4 could be perfectly used in secure face recognition (SFR) (Erkin et al., 2009a). In SFR a database contains M

(feature vectors of) faces and whenever a person would like to authenticate himself a facial picture is taken and compared to the M stored faces. This results in M distances with the faces in the database. The face yielding the minimum distance will be chosen by the system. The reliability of their implementation, so the probability that the right person is authenticated, equals 96%. The distances between different faces take values of $\ell = 50$ bits and are encrypted for privacy reasons.

In Erkin et al. (2009a) exact comparison is used requiring M executions of a secure comparison protocol with inputs of size ℓ. However, given that their database contains $M = 320$ faces, it is sufficient to have a secure comparison protocol that computes the correct result with probability $p = 1 - 2^{-19}$. This would slightly degrade the reliability to $96\% \cdot p^M \approx 95.94\%$ yielding a similar performance. The required private comparison subprotocol would need inputs of a significant lower size namely 19 bits. As they used DGK, this would reduce the amount of communication by a factor $\mathrm{NoC}_4(19)/(\mathrm{NoC}_{SMR}(50) + \mathrm{NoC}_{DGK}(50)) \approx 0.4$ and the amount of computation by a factor $\mathrm{NoM}_4(19)/(\mathrm{NoM}_{SMR}(50) + \mathrm{NoM}_{DGK}(50)) \approx 0.25$, for finding the most resembling face in the database.

4.3 Approximating the minimum

The idea of computing a probabilistic guarantee for $a \le b$ is easily extended to securely computing a likely minimum (or maximum) of two (or more) encrypted numbers. Namely, when the comparison result $(a \le b)$ is known, the minimum $\min(a, b)$ can readily be computed through a 1–2 oblivious transfer or a secure multiplication (SM):

$$\min(a,\ b) = b + (a \le b) \cdot (a - b)$$

The SM approach is illustrated in Protocol 5. To implement the SM, A has to blind the comparison result t before giving it to B, to avoid leakage of information. After the SM, the minimum is unblinded again. This step is also necessary when using a 1–2 oblivious transfer. The main computational effort of the SM are the involutions to the powers $z = x + r$ and $d + r$ by B and A respectively, so it is important that A restricts the size of r to $D + \sigma$ bits which is sufficient for securely blinding the corresponding values. Note that B could alternatively decrypt $[\tau]$ instead of using the homomorphic property, but this will be less efficient for most values of D, i.e., $D + \sigma < \frac{1}{4}\log_2 N$. Furthermore, the value z from Protocol 4 has been reused in the SM to avoid an extra decryption by B.

Unfortunately, computing a likely minimum does not guarantee that the computed value m is actually close to $\min(a, b)$. However, there is an alternative way of actually approximating the minimum value. The main idea is not to approximate ($z \bmod d < r \bmod d$), as is done in Protocol 4, but to approximate $(a \le b)$ by only comparing the most significant bits. In a general multi-party computation model, where each party has its own private inputs, such a solution could be easily implemented because A and B have their input values a and b available in plain. In our client-server model however, the main steps of such a protocol would be:

1 securely compute (or approximate) $[a \div d']$ and $[b \div d']$

2 securely compute the exact comparison result $[t] \leftarrow [(a \div d') \le (b \div d')]$

3 compute $m \leftarrow b + t \cdot (a - b) \approx \min(a, b)$ in the encrypted domain.

Protocol 5 Likely minimum

Party	A	B
Input	$[a]$, $[b]$, d, and d'	d, d', and K
Output	$[m]$, where $\Pr\{m = \min(a,b)\} \ge 1 - \frac{1}{d'}$	
Constraints	$0 \le a, b < d$, and $0 < d' < d < N \cdot 2^{-(\sigma+1)}$ $d = 2^D,\ d' = 2^{D'}, m \in \{a,b\}$	
NoM_5	$\mathrm{NoM}_4 + 5.5 + 3\ (D + \sigma)$	
NoC_5	$\mathrm{NoC}_4 + 2$	

1 A and B perform Protocol 4 to securely approximate the comparison result $(a \le b)$ such that A obtains a bit $[t]$, where $\Pr\{t = (a \le b)\} \ge 1 - \frac{1}{d'}$

2 A tosses a random coin $c \in \{0, 1\}$ to blind t: $\{\tau = c \oplus t\}$. If $c = 1$ then $[\tau] \leftarrow [1] \cdot [t]^{-1} \bmod N^2$ else $[\tau] \leftarrow [t]$.

3 A sends $[\tau]$ to B, who computes $[z \cdot \tau] \leftarrow [\tau]^z$, and sends $[z \cdot \tau]$ back to A.

4 A computes $[\mu] \leftarrow [b + \tau \cdot (a - b)] = [b] \cdot [z \cdot \tau]^{-1} \cdot [\tau]^{d+r}$.

5 A unblinds μ: If $c = 1$ then $[m] \leftarrow [a + b - \mu] = [a] \cdot [b] \cdot [\mu]^{-1}$ else $[m] \leftarrow [\mu]$.

Using our previous (sub)protocols for each step separately would require several decryptions by B which are relatively expensive operations. Therefore, a solution which needs only one decryption has been developed. It is depicted in Protocol 6. The best results are obtained when $(d \bmod d') < \frac{d'}{2}$, but without this constraint $|m - \min(a, b)| \le 2 \cdot d'$ can be similarly achieved by choosing $d \oslash d' = (d \div d') + 2$. The correctness of Protocol 6 can be found in the Appendix. The main part of the computational and communication effort is contained in the private comparison subprotocol. The size of its input variables is $\log_2(d \oslash d')$ bits which more or less equals $(\log_2 d) - (\log_2 d')$. To achieve an approximate minimum having a relative accuracy of 2^{-k}, i.e., $d' = 2^{-k} \cdot d$, the computational effort of the private comparison subprotocol would be therefore (small and) independent of d. The overall computational complexity is related to the complexity of Protocol 5. Let $\mathrm{NoM}(d, d')$ denote NoM given parameters d and d′, then $\mathrm{NoM}_6(d, \frac{d}{d'}) = \mathrm{NoM}_5(d, d') + 2$.

Protocol 6 Approximate minimum

Party	A	B
Input	$[a]$, $[b]$, d, and d'	d, d', and K
Output	$[m]$, where $\Pr\{m = \min(a,b)\} \le d'$	
Constraints	$0 \le a, b < d$, and $0 < d' < d < N \cdot 2^{-(\sigma+1)}$ $(d \bmod d') < \frac{d'}{2}$, $m \in \{a,b\}$	
NoM_6	$\mathrm{NoM}_{Enc}(4) + \mathrm{NoM}_{Dec}(2) + 12.5$ $+ 3(\sigma + \log_2 \mathrm{d}) + \mathrm{NoM}_{DGK}(\log_2 \frac{d}{d'})$	
NoC_6	$4 + \mathrm{NoC}_{DGK}(\log_2 \frac{d}{d'})$	

1 A encrypts d and computes $[x] \leftarrow [d + b - a] = [d] \cdot [b] \cdot [a]^{-1}$.

2 A chooses a random number r of $\sigma + \log_2 d$ bits, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

A and B approximate $x \div d'$ by $x \oslash d' = (z \div d') - (r \div d')$:

3 B decrypts $[z]$, and computes $z \oslash d' \leftarrow z \div d'$.

4 A computes $r \oslash d' \leftarrow r \div d'$.

We have $z \oslash d' = (x \oslash d') + (r \oslash d')$. Let $d \oslash d' = (d \div d') + 1$. A and B securely compute $[t] \leftarrow [(x \oslash d') \div (d \oslash d')]$, so $t \approx ((a \div d') < (b \div d'))$.

5 A computes $y_A \leftarrow (r \oslash d') \bmod (d \oslash d')$ and B computes $y_B \leftarrow (z \oslash d') \bmod (d \oslash d')$.

6 A and B perform a private comparison protocol. The inputs of A and B are y_A and y_B resp. The output for A will be the encrypted bit $[t']$ such that $[t'] = (y_A < y_B)$.

7 B computes $z \oslash d \leftarrow (z \oslash d') \div (d \oslash d')$, encrypts it, and sends $[z \oslash d]$ to A.

8 A computes $r \oslash d \leftarrow (r \oslash d') \div (d \oslash d')$, encrypts it, and computes $[t] \leftarrow [(z \oslash d) - (r \oslash d) - t'] = [z \oslash d] \cdot ([r \oslash d] \cdot [t'])^{-1}$.

The value $[m]$ is computed from $[t]$ using a SM:

9 A tosses a random coin $c \in \{0, 1\}$ to blind t. If $c = 1$ then $[\tau] \leftarrow [1] \cdot [t]^{-1}$ else $[\tau] \leftarrow [t]$. $\{\tau = c \oplus t\}$.

10 A sends $[\tau]$ to B, who computes $[z \cdot \tau] \leftarrow [\tau]^z$, and sends $[z \cdot \tau]$ back to A.

11 A computes $[\mu] \leftarrow [b + \tau \cdot (a - b)] = [b] \cdot [z \cdot \tau]^{-1} \cdot [\tau]^{d+r}$.

12 A unblinds μ: If $c = 1$ then $[m] \leftarrow [a + b - \mu] = [a] \cdot [b] \cdot [\mu]^{-1}$ else $[m] \leftarrow [\mu]$.

The computational complexity of Protocol 6 is depicted in Figure 2. We used a constant relative accuracy of 2^{-10}, i.e., $d' = 2^{-10} \cdot d$. It is compared to the secure minimum protocol used in SFR (Erkin et al., 2009a) which has $\mathrm{NoM}_{SFR} = \mathrm{NoM}_{SMR} + \mathrm{NoM}_{DGK}(50) + \mathrm{NoM}_{SM}(50)$ and $\mathrm{NoC}_{SFR} = \mathrm{NoC}_{SMR} + \mathrm{NoC}_{DGK}(50) + \mathrm{NoC}_{SM}(50)$. Although we only displayed the result for inputs of maximally 100 bits, there is already a considerable gain. Also depicted is the exact minimum protocol consisting of the exact secure comparison protocol implemented by the encrypted integer division Protocol 1, followed by a SM protocol for computing the actual minimum. In Figure 3 is shown how the computational complexity scales with the relative error. The most accurate approximations relatively offer the highest reduction.

Figure 2 NoM for secure minimum, both exact result and relative error 2^{-10}

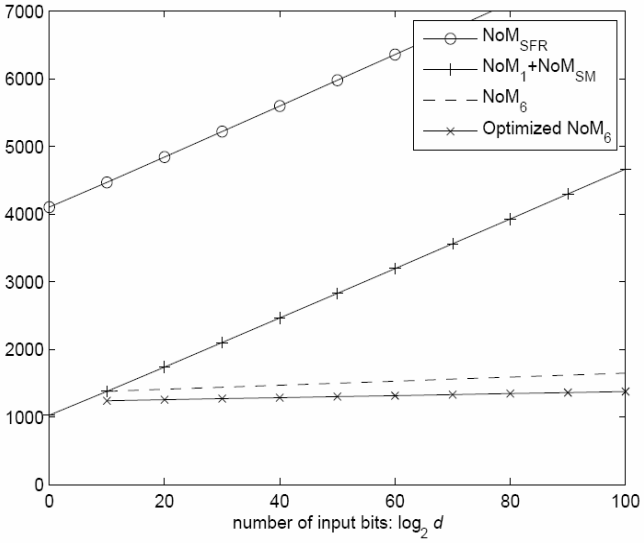
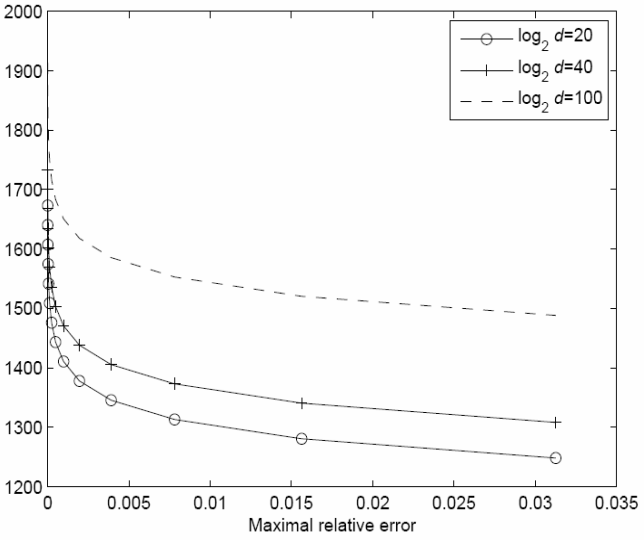


Figure 3 NoM for approximate minimum



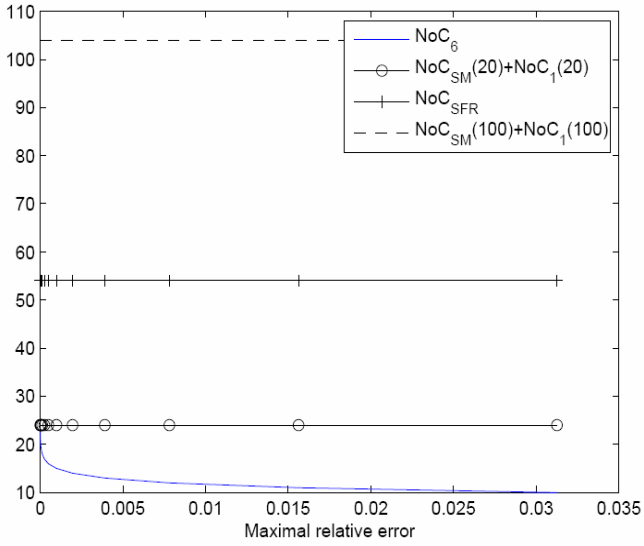
The number of communication rounds of Protocol 6 is two more than the private comparison protocol that is used as a subprotocol. The communication complexity is depicted in Figure 4 and compared to its exact counterparts like SFR and our Protocol 1 (extended by an extra SM for computing the minimum), showing that approximating the results offers substantial communication gains.

4.4 Optimisations

Whereas the output $x \div d$ in the first five protocols can be any non-negative integer, the value $t = (d + b - a) \div d$, approximated during the last three protocols, will always be a binary number. So instead of letting party A compute the encryption $[t]$, one could also compute a binary sharing,

i.e., party A obtains a private bit t_A, party B obtains a private bit t_B, and the output t equals the exclusive or $t_A \oplus t_B$. It turns out that this will lead to additional reductions of the computational and communication costs as explained below for Protocol 6. The optimisations for Protocols 4 and 5 are similar.

Figure 4 NoC for secure minimum, both approximated and exact result (see online version for colours)



By representing encrypted bits as binary sharings, steps 6 to 12 of Protocol 6 can be improved as follows:

6 A and B perform a private comparison protocol.

The inputs of A and B are y_A and y_B respectively. The output for A will be the private bit t'_A and the output for B will be the private bit t'_B such that $t' = (y_A < y_B) = t'_A \oplus t'_B$.

7 B computes $z \varnothing d \leftarrow (z \varnothing d') \div (d \varnothing d')$ and $t_B \leftarrow ((z \varnothing d) + t'_B) \bmod 2$.

8 A computes $r \varnothing d \leftarrow (r \varnothing d') \div (d \varnothing d')$ and $t_A \leftarrow ((r \varnothing d) + t'_A) \bmod 2$.

We have $t = (z \varnothing d) \div (r \varnothing d) - t' = t_A \oplus t_B$. The value $[m]$ is computed from t using a SM:

9 B encrypts t_B and sends $[t_B]$ to A.

10 A incorporates its share t_A: If $t_A = 1$ then $[t] \leftarrow [1] \cdot [t_B]^{-1}$ else $[t] \leftarrow [t_B]$.

11 B computes $z \cdot t_B$, encrypts it, and sends $[z \cdot t_B]$ to A.

12 A incorporates its share t_A: If $t_A = 1$ then $[z \cdot t] \leftarrow [z] \cdot [z \cdot t_B]^{-1}$ else $[z \cdot t] \leftarrow [z \cdot t_B]$.

13 A computes $[m] \leftarrow [b] \cdot [z \cdot t]^{-1} \cdot [t]^{d+r}$.

The correctness of the computations of t_A and t_B in steps 7 and 8 is straightforward because t is a binary number. Since $z = d + b - a + r$, $t \approx (a < b)$ and $m = b + (a \le b) \cdot (a - b)$, correctness of the final step easily follows. Compared to the original protocol, one encryption less (the number $[z \varnothing d]$) has to be sent, but the main computational advantage is the disappearance of the exponentiation $[\tau]^z \bmod N^2$ by B during the SM. This advantage is shown in Figure 2.

In step 6 is assumed that the private comparison subprotocol is capable of providing a binary sharing as output. The DGK algorithm naturally offers this option. Party B will set $t_B = 1$ only when it finds a zero in the ℓ encryptions provided by party A, and A's bit t_A will reveal whether party A requested a 'less than' or a 'greater than' comparison as indicated by the variable s (Erkin et al., 2009a).

4.5 Atallah, Kerschbaum and Du

We described a way of securely computing (and approximating) the minimum of two encrypted numbers $[a]$ and $[b]$ through a private comparison followed by a SM. An alternative approach by Atallah et al. (2003) uses additive sharing and random permutations to obtain the same result.

We adjusted their protocol to our setting of encrypted inputs and output, see Protocol 7. This protocol has similar computational and communication complexities as their original protocol with shared inputs and output. We assume party A has the private key of a second additively homomorphic cryptosystem (a second instance of the Paillier cryptosystem in our example) which we denote by $[\![\cdot]\!]$.

In Protocol 7 both parties choose a permutation bit c_A and c_B respectively to privately change the order of the numbers a and b so that eventually both parties can safely observe the outcome of the comparison. With the public comparison bit t both parties are able to compute additive shares of the minimum in an oblivious way so not learning which of the two values actually was the smallest.

From the protocol follows that $(\alpha - r_\alpha,\ \beta - r_\beta) = \Pi^{c_A}(a,b)$ where $\prod^0$ denotes the identity permutation and $\prod^1$ the permutation that swaps both values. Similarly, $(r_\alpha + s_\alpha,\ r_\beta + s_\beta) = \Pi^{c_B}(\rho_\alpha, \rho_\beta)$. Through easy but careful analysis it can be shown that $y_B - y_A = b - a$ when $c_A \oplus c_B = 1$ and $a - b$ otherwise. Therefore both parties will not learn additional information due to the publishing of the value t because the permutation bits c_A and c_B are private.

The advantage of their approach is that a SM is no longer required for computing the minimum value from the comparison result. Their disadvantage is that both parties need to decrypt twice which is quite expensive.

Furthermore, Protocol 7 can be optimised by two modifications:

1 By using packing (Erkin et al., 2012) in steps 3 and 8 the four decryptions can be reduced to one decryption per party. The costs of packing are $\log_2 d + \sigma$ and $\log_2 d + 2\sigma$ squarings in steps 3 and 8 respectively which makes it especially useful for smaller values of d.

2 The inputs y_A and y_B of the private comparison subprotocol contain $\log_2 d + 2\sigma + 1$ bits whereas our private comparison subprotocol only requires inputs of size d, i.e., $\log_2 d$ bits. It can be shown that a comparison with inputs y_A mod d and y_B mod d is sufficient for computing the same result t by following the optimisations described in Subsection 4.4.

Protocol 7 Minimum protocol of Atallah, Kerschbaum and Du for encrypted inputs

Party	A	B
Input	$[a]$, $[b]$, d, and $K_{[\cdot]}$	d and K
Output	$[m]$, where $m = \min(a, b)$	
Constraints	$0 \le a, b < d$, and $0 < d < N \cdot 2^{-(2\sigma+1)}$	
NoM_7	$\text{NoM}_{Enc}(6) + \text{NoM}_{Dec}(4) + 5 + \text{NoM}_{DGK}(\log_2 d + 2\sigma + 1)$	
NoC_7	$5 + \text{NoC}_{DGK}(\log_2 d + 2\sigma + 1)$	

1 A chooses random numbers r_a and r_b consisting of $\log_2 d + \sigma$ bits to blind a and b respectively: $[a'] \leftarrow [a] \cdot [r_a]$ and $[b'] \leftarrow [b] \cdot [r_b]$.

2 A tosses a random coin $c_A \in \{0, 1\}$ to randomly permute a' and b': if $c_A = 0$ then $([\alpha], [\beta]) \leftarrow ([a'], [b'])$ else $([\alpha], [\beta]) \leftarrow ([b'], [a'])$.

3 A sends $([\alpha], [\beta])$ to B, who decrypts both values.

4 A similarly permutes r_a and r_b: if $c_A = 0$ then $(r_\alpha, r_\beta) \leftarrow (r_a, r_b)$ else $(r_\alpha, r_\beta) \leftarrow (r_b, r_a)$.

5 A encrypts r_α and r_β with his own public key and sends $[\![r_\alpha]\!]$ and $[\![r_\beta]\!]$ to B.

6 B chooses random numbers s_α and s_β consisting of $\log_2 d + 2\sigma$ bits to blind r_α and r_β respectively: $[\![r'_\alpha]\!] \leftarrow [\![r_\alpha]\!] \cdot [\![s_\alpha]\!]$ and $[\![r'_\beta]\!] \leftarrow [\![r_\beta]\!] \cdot [\![s_\beta]\!]$.

7. B tosses a random coin $c_B \in \{0, 1\}$ to randomly permute r'_α and r'_β: if $c_B = 0$ then $([\![\rho_\alpha]\!],[\![\rho_\beta]\!]) \leftarrow ([\![r'_\alpha]\!],[\![r'_\beta]\!])$ else $([\![\rho_\alpha]\!],[\![\rho_\beta]\!]) \leftarrow ([\![r'_\beta]\!],[\![r'_\alpha]\!])$.

8 B sends $([\![\rho_\alpha]\!], [\![\rho_\beta]\!])$ to A, who decrypts both values.

9 A computes his input $y_A \leftarrow \rho_\alpha - \rho_\beta$.

10 B computes his input y_B: if $c_B = 0$ then $y_B \leftarrow \alpha - \beta + s_\alpha - s_\beta$ else $y_B \leftarrow \beta - \alpha + s_\beta - s_\alpha$.

11 To ensure both inputs are positive they add the constant $d \cdot 2^{2\sigma+1}$ to it.

12 A and B perform a private comparison protocol. The input of A is y_A, the input of B is y_B, and the output will be the public bit t such that $\{t = 1\} \equiv \{y_A < y_B\}$.

A and B compute the minimum by combining the correct shares:

13 If $t = 0$ then A computes $[m_A] \leftarrow [-\rho_\alpha]$ else $[m_A] \leftarrow [-\rho_\beta]$, by encrypting.

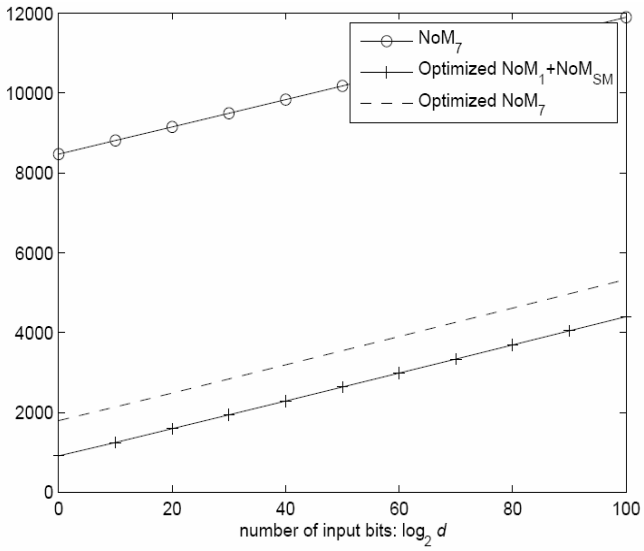
14 If $t \oplus c_B = 0$ then B computes $[m_B] \leftarrow [\alpha + s_\alpha]$ else $[m_B] \leftarrow [\beta + s_\beta]$, by encrypting.

15 B sends $[m_B]$ to A who combines the encrypted shares: $[m] \leftarrow [m_A] \cdot [m_B]$.

These optimisations are also applicable to their original minimum protocol with shared inputs and output (Atallah et al., 2003).

After these optimisations, Protocol 7 will still be computationally less efficient than our approach because (besides from their extra decryption) the costs for packing outweigh the costs for the extra SM which requires only $\frac{3}{2}(\log_2 d + \sigma)$ multiplications[1]. This is demonstrated in Figure 5 which shows the computational costs of their original protocol, their computational costs after our optimisations, and the computational costs of our approach, 'optimised NoM_1 + NoM_{SM}'. Our approach consists of Protocol 1 for securely computing the comparison result followed by an (according to Subsection 4.4) optimised SM to derive the minimum.

Figure 5 NoM for secure exact minimum



A comparison of the communication costs of these three protocols would show a similar picture with the same order, because the communication costs mainly depend on the number of bits of the inputs of the comparison protocol. Remember that our approach yields $\text{NoC}_1 + \text{NoC}_{SM} = 4 + \text{NoC}_{DGK}(\log_2 d)$ communicated encryptions.

4.6 Garbled circuits

Another well known technique for privately computing two-party functions is 'GC' (Kolesnikov et al., 2009; Henecka et al., 2010; Ben-David et al., 2008; Schröpfer et al., 2011). In this setting, each party has a private (unencrypted) input, one party constructs the (garbled) circuit for computing the particular function, and the other party evaluates the circuit without learning the intermediate (encrypted) results. However, we consider a different setting namely the client-server model where client A has one ($[x]$) or two ($[a]$ and $[b]$) encrypted integers, and server B has the decryption key

A well-known way (Kolesnikov et al., 2009) of using GC for integer division with public divisor d in our client-server model is as follows:

1 A, who has the encrypted input $[x]$ consisting of l bits, chooses a random number r of $\ell + \sigma$ bits, encrypts it, and computes $[z] \leftarrow [x + r] = [x] \cdot [r]$. A sends $[z]$ to B.

2 B decrypts $[z]$ and obtains his private input $z \bmod 2^{\ell}$.

3 A chooses a second random number $\tilde{r}$ of $\ell - \log_2 d + \sigma$ bits for blinding the output $x \div d$.

4 A constructs a GC C consisting of a series of garbled output tables for each gate. The circuit successively:

   a subtracts A's private input $r \bmod 2^{\ell}$ bitwise from B's input $z \bmod 2^{\ell}$, obtaining the garbled value x

   b divides x by d obtaining the garbled value $x \div d$

   c adds A's second private input $\tilde{r}$ bitwise to $x \div d$ obtaining the garbled output $y = (x \div d) + \tilde{r}$.

5 Before B can evaluate C he has to obtain from A the proper garbled input corresponding to his private input $z \bmod 2^{\ell}$ without A learning B's input. This can be achieved by a so called oblivious transfer (OT) protocol. See Subsection 2.3 of Kolesnikov et al. (2009) for an overview of efficient OT protocols.

6 A sends C and the output decryption table for y to B. The output table is used to convert the garbled output of C to plain value y.

7 B evaluates C obtaining the garbled output $y = (x \div d) + \tilde{r}$. During the evaluation each consecutive garbled output table corresponding with the next circuit gate is decoded. See Subsection 2.4 of Kolesnikov et al. (2009) for more details.

8 Using the output decryption table, B obtains the plain value $y = (x \div d) + \tilde{r}$, encrypts it and sends it to A.

9 A encrypts $-\tilde{r}$ and computes $[x \div d] = [y - \tilde{r}] = [y] \cdot [-\tilde{r}]$.

An important difference with our approach is that the value ℓ will become known to both parties leaking some information about x.

A comparison of the above approach with Protocol 1 shows that the main difference is the private comparison subprotocol (with inputs consisting of $\log_2 d$ bits) of Protocol 1 and the GC (with inputs of size ℓ) for one subtraction, integer division, and one addition. The remaining operations are decryption of $[z]$ and a few modular multiplications for both approaches. The relation between comparison and integer division is shown in equation (2), suggesting that a GC for comparison with inputs of size $\log_2 d$ is more or less equivalent to a GC for integer division by divisor d with an input x also consisting of $\log_2 d$ bits. Thus, given that computational and communication complexity increases in the size of the inputs and usually $\ell > \log_2 d$, we can conclude that the above GC approach can be improved by using Protocol 1 with a GC-based private comparison protocol as a subprotocol. This would also overcome the problem of leaking the value ℓ. A similar argument can be used for using a GC-based approach for Protocol 4.

For the minimum protocols 5 and 6 one could similarly use a GC-based subprotocol for the private comparison subprotocol. An additional advantage could be achieved by performing the SM (for computing the minimum from the comparison result) within the GC.

Our protocols work with any private comparison protocol as a building block as long as it is secure in the semi-honest model. The computational and communication costs of a GC-based comparison protocol are in fact lower than the DGK protocol (Barni et al., 2011). GCs can also be quite efficient when the divisor is not public but privately known (Lazzeretti and Barni, 2011). On the other hand, a GC only provides computational security, whereas DGK (and in that case also our entire protocol as shown in the next section) offers statistical security towards B. Note that providing a binary shared output, which enables using the optimisations described in the previous subsection, is easy for both DGK and GCs.

For approximate division with public divisor, as depicted in Protocol 2, it makes no sense to use GC in a client-server model because the transformation to private inputs and back to homomorphic encryption is equally costly as the entire Protocol 2. Nevertheless, in a model with private inputs our ideas of increasing efficiency by approximating the output could be similarly used to speed up a GC-based solution.

5 Security proof

We have to show that our division, comparison, and minimum protocols are secure in the semi-honest model. Since all messages towards party A are encrypted by the homomorphic encryption system of B, which is assumed to be semantically secure, it is informally clear that A will not learn any private information from B. On the other hand, all messages from A towards B are blinded, either multiplicatively or additively, by some random number chosen by A, so party B will neither learn private information from A.

We give a formal security proof for Protocol 1, the other security proofs are analogous. We closely follow Goldreich's (2001b) notation so A's input is $\bar{x}$, B's input is $\bar{y}$, and the output $f(\bar{x}, \bar{y})$ equals the pair $(f_1(\bar{x}, \bar{y}), f_2(\bar{x}, \bar{y}))$, where f_1 denotes A's output function and f_2 B's output function.

Theorem: Assume the homomorphic cryptosystem denoted by [.] is semantically secure and assume the secure comparison protocol in Protocol 1 privately computes the encrypted comparison result of both private inputs.

Then on inputs $\overline{x} = ([x], d)$ and $\overline{y} = (d, K)$, Protocol 1 privately computes the output $f(\overline{x}, \overline{y}) = ([x], \perp)$.

Proof: Definition 7.2.1 in Goldreich's (2001b) book, especially the case where *f* is deterministic, precisely states what we have to proof which loosely speaking comes down to 'Whatever can be computed by A or B from their view of a protocol execution, can be computed from their input and output'. Since we use the comparison protocol as a building block of *f*, we can present it as an oracle in our proofs and use Goldreich's (2001b) Composition Theorem 7.3.3. The only assumption we made about the private comparison protocol is that the comparison result is privately computed which fulfils Goldreich's premise for applying the composition theorem.

In Protocol 1, the view of A consists of its private numbers [*x*], and *d*, its random number *r* (of $\log_2 N - 1$ bits), its output [$x \div d$], and all intermediate messages received from B: the encrypted comparison bit [*t*], and the encrypted number [$z \varnothing d$]. Summarising, the view of A equals

$$V_1 = \left([x], d, r, [x \div d], [t], [z \varnothing d]\right).$$

According to Definition 7.2.1 (Goldreich, 2001b), it suffices to show that there exists a probabilistic polynomial-time algorithm S_1 such that $S_1(\overline{x}, f_1(\overline{x}, \overline{y}))$ is computationally indistinguishable from V_1. Since the encryption algorithm is semantically secure, every pair of encryptions is computationally indistinguishable (Goldreich, 2001b), so by letting S_1 randomly generate an encrypted bit [t_R], an encrypted integer [$(z \varnothing d)_R$] of $\log_2 d$ bits, and a random number r_R of $\log_2 N - 1$ bits, this condition is easily verified.

The view of B consists of its private number *d*, the decryption key *K*, and all intermediate messages received from A: the encrypted number [*z*], where $z = x + r$. Since B owns the decryption key, [*z*] can be decrypted to *z*. Summarising, the view of B is equivalent to

$$V_2 = (d, K, z)$$

Again, we have to show that there exists a probabilistic polynomial-time algorithm S_2 such that $S_2(\overline{y}, f_2(\overline{x}, \overline{y}))$ is computationally indistinguishable from V_2. This is easily satisfied by letting S_2 randomly generate an integer z_R of $\log_2 N - 1$ bits. It follows that

$$\Pr(z) = \sum_r \Pr(z \mid r) \cdot \Pr(r) = 2^{-(\log_2 N - 1)} \sum_r \Pr(x = z - r) \leq 2^{-(\log_2 N - 1)},$$

and thus that $|\Pr(z_R) - \Pr(z)| < 2^{-(\log_2 N - 1)} < 2^{-\sigma}$, which decreases faster than the reciprocal of any polynomial for sufficiently large security parameter σ, so *z* and z_R are statistically indistinguishable, and thus also computationally indistinguishable (Goldreich, 2001a). □

We conclude that Protocol 1 privately computes A's output [$x \div d$] in the semi-honest model. In fact, we showed that the integer *x* is even statistically secure towards B. Whether this holds for the entire protocol will depend on the chosen comparison protocol. In the variation described at the end of Section 2, where (*x* and) *r* can cover the entire plain text interval, this could be further extended to information theoretic security towards B. This means that even when server B would have unbounded computational power, A's number *x* would remain completely unknown to B.

Because Protocol 6 is somewhat more involved than the other protocols, we also give the additionally required arguments to show that Protocol 6 privately computes A's output [*m*] in the semi-honest model.

Firstly, the view of A consists of its private numbers $\overline{x} = ([a], [b], d, d')$, its random numbers *r* and *c*, its output $f_1(x, y) = [m]$, and all intermediate messages received from B: the encrypted comparison bit [t'], and the encrypted numbers [$z \varnothing d$] and [$z \cdot \tau$]. Summarising, the view of A equals

$$V_1 = \left([a], [b], d, d', r, c, [m], [t'], [z \varnothing d], [z \cdot \tau]\right).$$

Again, using the argument that the encryption algorithm is semantically secure, it is easy to show that there exists a probabilistic polynomial-time algorithm S_1 such that $S_1(\overline{x}, f_1(\overline{x}, \overline{y}))$ is computationally indistinguishable from V_1.

Secondly, the view of B consists of its private numbers $\overline{y} = (d, d', K)$, and all intermediate messages received from A which B can decrypt: *z* and τ. Like in the other protocols, party B has no output: $f_2(\overline{x}, \overline{y}) = \perp$. Summarising, the view of B is equivalent to

$$V_2 = (d, d', K, z, \tau).$$

We have to show that there exists a probabilistic polynomial-time algorithm S_2 such that $S_2(\overline{y}, f_2(\overline{x}, \overline{y}))$ is computationally indistinguishable from V_2. As shown before, for *z* it is sufficient to let S_2 generate a random integer z_R of $\log_2$ N – 1 bits. For τ, one can deduce that $\Pr(\tau = 0) = \Pr(c \oplus t = 0) = \Pr(c = t = 0) + \Pr(c = t = 1) = \frac{1}{2} \cdot \Pr(t = 0) + \frac{1}{2} \cdot \Pr(t = 1) = \frac{1}{2}$, because *c* is a fair coin tossed by A. Therefore, it suffices to let S_2 generate a random bit τ_R which will be statistically indistinguishable from τ, and thus also computationally indistinguishable (Goldreich, 2001a).

Concluding, similarly using Goldreich's composition theorem for the private comparison subprotocol, we have that Protocol 6 privately computes A's output [*m*] in the semi-honest model.

6 Conclusions

We described three new protocols for dividing an encrypted number, each with its own merits. The first protocol computes the exact division result with known divisor avoiding an intermediate SMR step like most existing approaches. The other two approximate the division result, one with known and one with unknown divisor, both of

which have a very low computational and communication complexity compared to their exact counterparts.

We were able to significantly reduce the computational and communication complexity of the secure exact minimum protocol by Atallah et al. (2003). We also showed that our approach for securely computing the exact minimum still outperforms their optimised protocol when the input values are not too large.

Using the ideas of the approximated division protocols, a new protocol was derived for computing secure comparison with a probabilistic guarantee. Furthermore, two new protocols were developed for securely computing the minimum (or maximum), either in terms of likelihood, or accuracy. Application within biometrics has been shown to significantly improve both communication and computational complexity because of the reduced size of the inputs required for the private comparison subprotocol.

Suggestions were given for extending the protocols in order to avoid limitations on the size of the input value and for improving the security properties. All protocols are provably secure in the client-server model with semi-honest behaviour.

Acknowledgements

This publication was supported by the Dutch National Programme COMMIT.

References

Adam, N.R. and Wortmann, J.C. (1989) 'Security-control methods for statistical databases: a comparative study', *ACM Computing Surveys*, Vol. 21, No. 4, pp.515–556.

Atallah, M., Bykova, M., Li, J., Frikken, K. and Topkara, M. (2004) 'Private collaborative forecasting and benchmarking', in *Proceedings of the ACM Workshop on Privacy in an Electronic Society*, pp.103–114.

Barni, M., Failla, P., Lazzeretti, R., Sadeghi, A-R. and Schneider, T. (2011) 'Privacy-preserving ECG classification with branching programs and neural networks', *IEEE Transactions on Information Forensics and Security (TIFS)*, Vol. 6, No. 2, pp.452–468.

Ben-David, A., Nisan, N. and Pinkas, B. (2008) 'Fairplaymp – a secure multi-party computation system', in *ACM CCS*.

Bianchi, T., Veugen, T., Piva, A. and arni, M. (2009b) 'Processing in the encrypted domain using a composite signal representation: pros and cons', in *IEEE International Workshop on Information Forensics and Security*.

Bianchi, T., Veugen, T., Piva, A. and Barni, M. (2009a) 'Processing in the encrypted domain using a composite signal representation', in *SPEED '09*, Lausanne, Switzerland.

Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M. and Toft, T. (2009) 'Secure multiparty computation goes live', *in Financial Cryptography and Data Security*, Vol. 5628, pp.325–343, Springer-Verlag.

Bunn, P. and Ostrovsky, R. (2007) 'Secure two-party k-means clustering', in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp.486–497, USA.

Catrina, O. and Saxena, A. (2010) 'Secure computation with fixed-point numbers', in *Financial Cryptography and Data Security*, Vol. 6052 of Lecture Notes in Computer Science, pp.35–50, Springer, Berlin/Heidelberg.

Dahl, M., Ning, C. and Toft, T. (2012) 'On secure two-party integer division', in *The 16th International Conference on Financial Cryptography and Data Security*.

Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B. and Toft, T. (2006) 'Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation', in *Proceedings of the third Theory of Cryptography Conference, TCC*, Vol. 3876 of Lecture Notes in Computer Science, pp. 285–304.

Damgård, I., Geisler, M. and Krøigaard, M. (2009) 'A correction to efficient and secure comparison for on-line auctions', *Journal of Applied Cryptology*, Vol. 1, No. 4, pp.323–324.

Erkin, Z., Beye, M., Veugen, T. and Lagendijk, I. (2010) 'Privacy enhanced recommender system', in *Thirty-first Symposium on Information Theory in the Benelux*, pp.35–42, Rotterdam.

Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, R.L. and Toft, T. (2009a)]Privacy-preserving face recognition', in *Proceedings of the Privacy Enhancing Technologies Symposium*, pp.235–253, Seattle, USA.

Erkin, Z., Veugen, T., Toft, T. and Lagendijk, R.L. (2009b) 'Privacy-preserving user clustering in a social network', in *IEEE International Workshop on Information Forensics and Security*.

Erkin, Z., Veugen, T., Toft, T. and Lagendijk, R.L. (2012) 'Generating private recommendations efficiently using homomorphic encryption and data packing', *IEEE Transactions on Information Forensics and Security*, Vol. 7, No. 3, pp.1053–1066.

Feigenbaum, J., Ishai, Y., Malkin, T., Nissim, K., Strauss, M.J. and Wright, R.N. (2006) 'Secure multiparty computation of approximations', *ACM Transactions on Algorithms*, Vol. 2, No. 3, pp.435–472.

Goldreich, O. (2001a) *Foundations of Cryptography*, Vol. 1, Cambridge University Press, New York, USA.

Goldreich, O. (2001b) *Foundations of Cryptography: Basic Applications*, Vol. 2, Cambridge University Press, New York, USA.

Guajardo, J., Mennink, B. and Schoenmakers, B. (2009) 'Modulo reduction for Paillier encryptions and application to secure statistical analysis', in *SPEED '09*, Lausanne, Switzerland.

Henecka, W., K¨ogl, S., Sadeghi, A-R., Schneider, T. and Wehrenberg, I. (2010) 'Tasty: tool for automating secure two-party computations', in *17th ACM Conference on Computer and Communications Security (CCS '10)*, pp.451–462.

Jakobsen, T. (2006) *Secure Multi-party Computation on Integers*, Master thesis, University of Aarhus, Denmark.

Kiltz, E., Leander, G. and Malone-Lee, J. (2005) 'Secure computation of the mean and related statistics', in *Proceedings of Theory of Cryptography Conference*, Vol. 3378 of Lecture Notes in Computer Science, pp.283–302.

Kolesnikov, V., Sadeghi, A-R. and Schneider, T. (2009) 'Improved garbled circuit building blocks and applications to auctions and computing minima', in *CANS*, Vol. 5888 of Lecture Notes in Computer Science, pp.1–20, Springer-Verlag.

Lazzeretti, R. and Barni, M. (2011) 'Division between encrypted integers by means of garbled circuits', in *IEEE International Workshop on Information Forensics and Security*, pp.1–6.

Menezes, A.J., van Oorschot, P.C. and Vanstone, S.A. (1996) *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida, USA.

Naor, M., Pinkas, B. and Summer, R. (1999) 'Privacy preserving auctions and mechanism design', *ACM Conference on Electronic Commerce*, pp.129–139.

Paillier, P. (1999) 'Public-key cryptosystems based on composite degree residuosity classes', in *Proceedings of Eurocrypt*, Vol. 1592 of Lecture Notes in Computer Science, pp.223–238, Springer-Verlag.

Rane, S. and Sun, W. (2010) 'Privacy preserving string comparisons based on Levenshtein distance', *IEEE Workshop on Information Forensics and Security*.

Schoenmakers, B. and Tuyls, P. (2006) 'Efficient binary conversion for Paillier encrypted values', in *Advances in Cryptology – EUROCRYPT,* Vol. 4004 *of Lecture Notes in Computer Science*, pp.522–537, Springer.

Schr¨opfer, A., Kerschbaum, F. and Müller, G. (2011) 'L1 an intermediate language for mixed-protocol secure computation', in *35th IEEE Computer Software and Applications Conference (COMPSAC) Encrypted Integer Division and Secure Comparison*.

Springer-Verlag. Damgård, I., Geisler, M. and Krøigaard, M. (2008) 'Homomorphic encryption and secure comparison', *Journal of Applied Cryptology*, Vol. 1, No. 1, pp.22–31.

Toft, T. (2007) *Primitives and Applications for Multi-party Computation*, PhD thesis, University of Aarhus, Aarhus, Denmark.

Verykios, V.V., Bertino, E., Fovino, I.N., Provenza, L.P., Saygin, Y. and Theodoridis, Y. (2004) 'State-of-the-art in privacy preserving data mining', *ACM SIGMOD Record*, Vol. 33, No. 1, pp.50–57.

Veugen T., Atallah, M.J., Kerschbaum, F. and Du, W. (2003) 'Secure and private sequence comparisons', in *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pp.39–44, ACM Press, Washington, DC.

Veugen, T. (2010) 'Encrypted integer division', in *IEEE Workshop on Information Forensics and Security*.

Notes

1 Only for very large values of d, so $\log_2 d = \log_2 N - 2\sigma - 1$, the three extra decryptions of Atallah, Kerschbaum and Du cost $\frac{9}{8}\log_2 N$ multiplications, whereas an SM will cost $\frac{3}{2}(\log_2 d + \sigma)$ multiplications, which is slightly higher.

Appendix

Correctness proof of Protocol 6

Protocol 6 consists of the three steps previously mentioned in Subsection 4.3. In the first step, the value $x \oslash d'$ is computed as in Protocol 2 to approximate $x \div d'$. Since $z = x + r$, we have, by equation (1), $z \div d' = (x \div d') + (r \div d') + t'$ for some binary value t', so $x \oslash d' = (z \oslash d') - (r \oslash d') = (z \div d') - (r \div d') = (x \div d') + t'$, as in Protocol 2. Therefore, $x \oslash d'$ is bounded by $0 \le x \oslash d' \le ((2d-1) \div d') + 1 \le ((2d-1) \div d') + 1$. This upper bound equals, by equations (1), $2 \cdot (d \div d') + t_d + 1$, where t_d is the comparison result $((2d) \bmod d' < (d \bmod d'))$. Let $\alpha = d \bmod d'$. Then $t_d = ((2\alpha) \bmod d' < \alpha)$, so $t_d = 0$ exactly when the extra constraint $\alpha < \frac{d'}{2}$ holds, in which case $0 \le x \oslash d' \le 2 \cdot (d \div d') + 1 < 2 \cdot ((d \div d') + 1) = 2 \cdot (d \oslash d')$.

In the second step, the bit $t = (x \oslash d') \div (d \oslash d')$ is exactly computed, similar to Protocol 1. And in the final third step, the minimum m is computed from t using a SM. To understand $|m - \min(a, b)| < d'$, we consider three cases:

1 $b - a \ge d'$. In this case $x = d + b - a \ge d - d'$, thus $x \oslash d' \ge x \div d' \ge (d + d') \div d' = d \oslash d'$, so $t = (x \oslash d') \div (d \oslash d') = 1$ which in this case equals the comparison result $(a \le b)$ because $a \le b - d' < b$. Therefore m will be exactly equal to $\min(a, b)$.

2 $b - a \le -d'$. In this case $x \oslash d' \le (x \div d') + 1 \le ((d + d') \div d') + 1 = d \div d' < d \oslash d'$, so $t = (x \oslash d') \div (d \oslash d') = 0$ which in this case also equals the comparison result $(a \le b)$ because $a \ge b + d' > b$. Therefore m will be exactly equal to $\min(a, b)$.

3 $|a - b| < d'$. In this case t will not necessarily equal the comparison result $(a \le b)$, but because $m \in \{a, b\}$, it is guaranteed that $|\text{m-min}(a - b)| < d'$.

Which proves the correctness of Protocol 6.