Encrypted Integer Division

Thijs Veugen
TNO Information and Communication Technology
Delft, The Netherlands
thijs.veugen@tno.nl
and
Multimedia Signal Processing Group, Delft University of Technology
Delft, The Netherlands

***Abstract*—When processing signals 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. We present new solutions for dividing encrypted data, having low computational complexity. Two protocols for computing exact division, and two for approximating the division result.**

I. Introduction

When processing signals 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 e.g. in secure clustering [1], [2] and secure personal recommendation [3]. An import application of integer division is unpacking. When processing signals in the encrypted domain, it's computationally and communicatively interesting to pack several signals into one encryption [4]. It enables simultaneous execution of linear operations on all signals that are packed in one encryption. However, for more complicated operations, the signals have to be unpacked which requires an additional protocol for integer division.

We present new solutions for dividing encrypted data, having low computational complexity. Two for computing exact division, and two for approximating the division result. The most suitable approach depends on the application in mind.

A. Preliminaries

We consider the client-server model where party A, the client, has some encrypted signal $[x]$, 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.

Homomorphic encryption, denoted by $[.]$, is used, to encrypt the signals represented by integers. Any multiplicative homomorphic encryption system could be used, as long as it's semantically secure, e.g. Paillier [5]. The integer N is the modulus of the encryption system which usually equals the product of two large primes. We recall an important property of homomorphic encryption systems, namely that for integers x and y we have $[x] \cdot [y] = [x+y] \bmod N$. The multiplicative inverse of x modulo N is denoted by x^{-1} and equals the integer y, $0 \leq y < N$, such that $x \cdot y = 1 \bmod N$. The multiplicative inverse is most efficiently computed by using the Euclidean algorithm [6], and can also be used to negate an encrypted integer: $[-x] \leftarrow [x]^{-1} \bmod N$. For convenience, we neglect in our notation that the cipher text modulus in Paillier is N^2 and use N instead. To estimate the computational complexity of the different protocols, we use the fact that an involution modulo N with an exponent of n bits will on average take $\frac{3}{2}n$ multiplications modulo N. A Paillier decryption costs around $\frac{3}{2}\log_2 N$ multiplications modulo N, and a Paillier encryption only 1 when the random factor is precomputed [5].

More precisely, A has an encrypted number $[x]$, $0 \leq x < N$, in a modular system with modulus N. A can't decrypt but wants to divide the encrypted number $[x]$ by a number d, $0 < d < N$. I.e. he wants to find the encrypted number $[x \div d]$, such that $x = d \cdot (x \div d) + (x \bmod d)$, where $x \div d \geq 0$ and $0 \leq x \bmod d < d$.

Let σ be the statistical security parameter, which value is usually chosen around 80. We assume all random variables, excluding the inputs of the secure multi-party computation protocol, are uniformly chosen.

B. Related work

The problem of dividing an encrypted number has been studied recently in different settings. Schoenmakers and Tuyls [7] 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.

Jakobsen [8] describes a couple of methods for dividing two secretly shared integers. Also approximations are considered, but all methods use some kind of binary search.

Damgård et al. [9] present secure protocols for modulo reduction of secretly shared integers within constant rounds. The modulus is either public or secretly shared. This modulo reduction could consequently be used to compute the division.

Bunn and Ostrovsky [1] describe a protocol for securely clustering two databases. They propose a subprotocol that securely computes the division of two shared integers, using ideas of the long division.

Toft [10] shows how to reduce a secretly shared integer with respect to a known modulus, thereby generalizing 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 a.o.

While processing homomorphically encrypted data, packing is used to reduce communication and computation complexity. Bianchi et al. [4], [11] show how to unpack an encrypted integer that presents 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 modulo reduction. A similar protocol for reducing an encrypted number with respect to a known modulus was simultaneously found by Guajardo et al. [12]. This protocol is further improved in our subsection III-A by avoiding an intermediate modular result.

II. Exact division

We consider two different solutions for the exact computation of $[x \div d]$ from $[x]$, given the public divisor d, $0 < d < N$. Both solution require a secure comparison protocol as a subprotocol. Any comparison protocol with encrypted output could be used here, as long as it's secure in the semi-honest model. An efficient comparison protocol is described in Section 5.2 of [13] for the malicious model with threshold decryption. Optimizing this for our client-server model yields a comparison protocol requiring on average $4(\ell - 1)$ multiplications modulo N when the inputs consist of ℓ bits, and all random (encryption) factors are precomputed. When analyzing the computational complexity of our protocols, we assume this comparison protocol is used, which uses ℓ communication rounds. However, the gain of our protocols w.r.t. known solutions will increase when computationally more intensive (but possibly having lower communication complexity) comparison protocols are used.

In the first solution, the interval $[0, N)$ is divided into d subintervals, each corresponding to an index i, $0 \leq i < d$. By using a binary search, we can determine the index i of the subinterval that contains the value $(x \cdot d^{-1}) \bmod N$. We'll show that this index can be used to compute $[x \bmod d]$, and consequently $[x \div d]$.

The second solution is more efficient, and uses blinding and only one comparison to compute $[x \div d]$.

A. Binary search

Although more efficient solutions exist, we present this approach because it's a novel idea which can also be used when the divisor is privately known to B.

When considering the d numbers $x_i = (i \cdot d^{-1}) \bmod N$, $0 \leq i < d$, we know that these d numbers will divide the plain text range $[0, N)$ into d subintervals. Let x be an integer in the plain text range, i.e. $0 \leq x < N$. Since $x = d \cdot (x \div d) + (x \bmod d)$, we derive that $x \cdot d^{-1} = (x \div d) + (x \bmod d) \cdot d^{-1}$ modulo N. So the number $x \cdot d^{-1}$ will be contained in the subinterval that starts with x_i, exactly when $x \bmod d = i$. It also follows that the length of the subinterval that is headed by x_i equals $(N \div d) + 1$ when $i < (N \bmod d)$, and $N \div d$ otherwise. It is easily verified that the sum of the length of all subintervals indeed equals N.

Unfortunately, the numbers x_i are not ordered, i.e. not necessarily $x_i < x_j$ whenever $i < j$. Consider the ordered list of x_i's, and let y_i, $0 \leq y_i < d$, be the index of the i^{th} interval, $0 \leq i < d$. So $y_i = j$ when the i^{th} interval is headed by x_j. Consequently, $y_0 = 0$, because $x_0 = 0$ will definitely head the first interval.

Theorem 1. Define $k = d - (N \bmod d)$, then for all i, $0 \leq i < d$:

$$y_i = (i \cdot k) \bmod d$$

Proof: The proof is by natural induction on i. Obviously, $y_0 = 0$. Suppose $y_i = (i \cdot k) \bmod d$ for some i, $0 \leq i < d - 1$. We consider two cases.

1) $y_i < (N \bmod d)$. In this case the i^{th} interval has length $(N \div d) + 1$, so the $(i+1)^{th}$ interval will be headed by $x_{y_{i+1}} = x_{y_i} + (N \div d) + 1$, which equals $(y_i \cdot d^{-1}) \bmod N + (N \div d) + 1$. Since $k = d - (N \bmod d)$, we derive $(k \cdot d^{-1}) \bmod N = 1 + (N \div d)$, so $x_{y_{i+1}} = ((y_i + k) \cdot d^{-1}) \bmod N$, and thus $y_{i+1} = (y_i + k) \bmod N$. Because in this first case $y_i < (N \bmod d)$, we have $y_i + k < d$, so $y_{i+1} = y_i + k = (y_i + k) \bmod d$. By induction, $y_{i+1} = ((i+1) \cdot k) \bmod d$.
2) $y_i \geq (N \bmod d)$. In this case the i^{th} interval has length $N \div d$, so the $(i+1)^{th}$ interval will be headed by $x_{y_{i+1}} = x_{y_i} + (N \div d)$, which equals $(y_i \cdot d^{-1}) \bmod N + (N \div d)$. By definition of k, this is equal to $((y_i + k - d) \cdot d^{-1}) \bmod N$, and therefore $y_{i+1} = (y_i + k - d) \bmod N$. Because in this second case $y_i \geq (N \bmod d)$, we have $y_i + k \geq d$, and thus $y_{i+1} = y_i + k - d = (y_i + k) \bmod d$. By induction, $y_{i+1} = ((i+1) \cdot k) \bmod d$.

In both cases, $y_{i+1} = ((i+1) \cdot k) \bmod d$.

(End of Proof)

Note that k and d have no common divisors, otherwise we found a divisor of N, so the numbers y_i will be different for all i, $0 \leq i < d$.

Theorem 1 is illustrated by an example in Table I. We choose $N = 3 * 5 = 15$. The divisor is $d = 7$. So $N \bmod d = 1$, $k = d - (N \bmod d) = 6$, and $N \div d = 2$. The multiplicative inverse of d is in this example equal to 13. It is easily verified that indeed $y_i = (i * k) \bmod d$.

TABLE I
EXAMPLE WITH $N = 15$ AND $d = 7$

i	$(i * d^{-1}) \bmod N$	x_i	y_i
0	0	$x_0 = 0$	$y_0 = 0$
7	1		
14	2		
6	3	$x_6 = 3$	$y_1 = 6$
13	4		
5	5	$x_5 = 5$	$y_2 = 5$
12	6		
4	7	$x_4 = 7$	$y_3 = 4$
11	8		
3	9	$x_3 = 9$	$y_4 = 3$
10	10		
2	11	$x_2 = 11$	$y_5 = 2$
9	12		
1	13	$x_1 = 13$	$y_6 = 1$
8	14		

Theorem 1 leads to Protocol 1 for computing $[x \div d]$ from $[x]$ and d, by considering the ordered list x_{y_i}, $0 \leq i < d$. The correctness of the computation of $x \div d$ follows from $c = x \cdot d^{-1} = (x \div d) + (x \bmod d) \cdot d^{-1} = (x \div d) + x_{y_i}$ modulo N.

Protocol 1 Integer division by binary search

Input A	$[x]$, and d
Input B	d, and K
Output A	$[x \div d]$
Constraints	$0 \leq x < N$ and $0 < d < N$

1) Party A computes $[c] = [x \cdot d^{-1}]$ by raising $[x]$ to the power d^{-1} modulo N.
2) A and B perform $\log_2 d$ secure comparisons, each time comparing $[c]$ with some $[x_j]$, to find the encrypted starting value $[s]$, $s = x_{y_i}$, of the interval (with index i) that contains c.
3) Party A computes $[x \div d] = [c - s] = [c] \cdot [s]^{-1} \bmod N$.

Since the index i of the interval that contains c reveals the value of $y_i = x \bmod d$, this index should remain unknown to both parties. So the result of each comparison should be encrypted. The techniques to perform the secure comparisons are secure indexing and secure binary search with closed result, and are described in [10].

This solution can be extended to the case where party A is not allowed to know d, but only its size $\log_2 d$. The value d (actually d^{-1}) is only used by party A in the computation of $[c]$, but the following modification facilitates the computation of $[c]$ by A without knowing d:

1) Party A computes $[c] = [x \cdot d^{-1}]$ as follows:
   a) Party A chooses a random r, $0 < r < N$, to blind x, and computes $[a] = [x \cdot r]$ by raising $[x]$ to the power r modulo N. A sends $[a]$ to B.
   b) Party B computes $[b] = [a \cdot d^{-1}]$, either by raising $[a]$ to the power d^{-1} modulo N, or through decryption. B sends $[b]$ to A.
   c) Party A computes $[c] = [x \cdot d^{-1}]$, by raising $[b]$ to the power r^{-1} modulo N.

The extended protocol leads to a solution of the following problem:

Input A	$[x]$
Input B	d, and K
Output A	$[x \div d]$, and $\log_2 d$
Constraints	$0 \leq x < N$, and $0 < d < N$

One could try to further extend this solution by refusing B to learn the value d, and having it as an encrypted input $[d]$ for A, but this would hinder B in the computation of the ordered list x_{y_i}, and lead to a complex and inefficient solution.

The complexity of Protocol 1 is mainly determined by the $\log_2 d$ secure comparisons. The computational complexity of the secure comparison protocol depends on the size of its inputs, which is $\log_2 N$ bits in this case. Furthermore, secure indexing is needed which requires a number of secure multiplications linear in d [10]. The number of communication rounds is dominated by the $\log_2 d$ iterations of the binary search.

The reason for A learning $\log_2 d$ is that its equal to the number of performed secure comparisons. This could be avoided by always performing $\log_2 N$ secure comparisons (of which some will be meaningless), at the cost of an increased complexity.

B. Blinding x

A more 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) size $\log_2 x + \sigma$, where σ, usually in the order of 80, is the statistical security parameter. It leads to Protocol 2.

Protocol 2 Integer division by blinding

Input A	$[x]$, and d
Input B	d, and K
Output A	$[x \div d]$
Constraints	$0 \leq x < N \cdot 2^{-\sigma}$, and $0 < d < N$

1) A chooses a random number r of size $\log_2 N - 1$, and computes $[z] = [x + r] = [x] \cdot [r] \bmod N$. A sends $[z]$ to B.
2) B decrypts $[z]$, and computes $z \bmod d$.
3) A and B perform a secure 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 computes $c = z \div d$, encrypts it, and sends $[c]$ to A.
5) A computes $[x \div d] = [(z \div d) - (r \div d) - t] = [c] \cdot ([r \div d] \cdot [t])^{-1} \bmod N$.

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 $(z \bmod d) = (x \bmod d) + (r \bmod d) \geq (r \bmod d)$, i.e. $t = 0$.

Since the addition of x and r is not allowed to initiate a carry-over modulo N, the above solution 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 an extra comparison protocol ($z < r$) to determine whether a carry-over has occurred (similar to the approach described in Protocol 4).

Protocol 2 requires only one execution of the secure comparison protocol, with inputs of $\log_2 d$ bits. The remaining steps only require one encryption, three modular multiplications, and one inversion by A, and one decryption, and one encryption by B. So the total number of multiplications modulo N for Protocol 2 is $\frac{3}{2}\log_2 N$ (for the decryption) $+ 4\log_2 d$ (for the comparison)$+2$. This outperforms the similar division protocols of [4], [11], [12], 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 $\frac{3}{2}\log_2 N$ modular multiplications. The gain in computational complexity varies from 21% upto 50%, depending on the size of d.

The number of communication rounds for Protocol 2 is constant, whenever the used comparison protocol has constant rounds complexity.

III. Approximate division

Instead of exactly computing (the encrypted) $x \div d$, it might be enough for some applications, to approximate $x \div d$. This approach is interesting since it leads to more efficient solutions that don't require a secure comparison protocol as a subprotocol. The gain in modular multiplications will be small when d is small, but can increase upto 73% for large values of d. It also reduces the communication complexity and the number of communication rounds.

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 size $\log_2 d$.

A. Approximate division with public divisor

This approach is derived from the blinding approach described in subsection II-B. There, the comparison result t is used to slightly correct the division result. The analysis in subsection II-B shows that $xd = (z \div d) - (r \div d)$ is a good approximation of $x \div d$, since either $xd = x \div d$ or $xd = (x \div d) - 1$, depending on the outcome of the comparison. This leads to Protocol 3.

In order to avoid limitations on the length of x, the protocol could, similarly as in subsection II-B, be extended to Protocol 4.

The comparison result t will inform A whether a carry-over has occurred during the computation of z, in which case $z = x + r - N$, and thus $x \div d \approx (z \div d) + ((N-r) \div d)$. When no carry-over has occurred, i.e. $t = 0$, then $z = x + r$, so $x \div d \approx (z \div d) - (r \div d)$. The computation of rd by A assures that $rd = -(r \div d)$ when $t = 0$, and $rd = (N - r) \div d$ otherwise.

Protocol 3 Approximate division with public divisor and input constraints

Input A	$[x]$, and d
Input B	d, and K
Output A	$[xd]$
Constraints	$0 \leq x < N \cdot 2^{-\sigma}$, and $0 < d < N$ $xd \in \{x \div d, (x \div d) + 1\}$

1) A chooses a random number r of size $\log_2 N - 1$, and computes $[z] = [x + r] = [x] \cdot [r] \bmod N$. A sends $[z]$ to B.
2) B decrypts $[z]$, computes $c = z \div d$, encrypts it, and sends $[c]$ to A.
3) A computes $[xd] = [c] \cdot [-(r \div d)] \bmod N$.

Protocol 4 Approximate division with public divisor without input constraints

Input A	$[x]$, and d
Input B	d, and K
Output A	$[xd]$
Constraints	$0 \leq x < N$, and $0 < d < N$ $xd \in \{x \div d, (x \div d) + 1\}$

1) A chooses a random number r, $0 \leq r < N$, and computes $[z] = [x + r] = [x] \cdot [r] \bmod N$. A sends $[z]$ to B.
2) B decrypts $[z]$, computes $c = z \div d$, encrypts it, and sends $[c]$ to A.
3) A and B perform a secure comparison protocol. The input of A is r, the input of B is z, and the output for A will be the encrypted bit $[t]$ such that $\{t = 1\} \equiv \{z < r\}$.
4) A computes $[rd] = [t]^{((N-r) \div d) + (r \div d)} \cdot [-(r \div d)] \bmod N$.
5) A computes $[xd] = [c] \cdot [rd] \bmod N$.

Therefore, $xd = c + rd$ will be a good approximation for $x \div d$. In fact, either $xd = x \div d$ or $xd = (x \div d) + 1$.

The main drawback of this extended protocol is the extra comparison protocol, which absence was the big advantage of going for an approximation instead of an exact division result. Although the security towards B is increased (see Section IV), it doesn't seem worthwhile in practice, but nicely illustrates the idea of fully blinding.

B. Approximate division with private divisor

Instead of having d publicly known, Protocol 3 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 5.

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 is of size $\log_2 d$, we derive $0 \leq r_m < 2d$, so $(r \div d) \leq r_d \leq (r \div d) + 1$. Therefore

Protocol 5 Approximate division with private divisor

Input A	$[x]$
Input B	d, and K
Output A	$\log_2 d$, and $[xd]$
Constraints	$0 \leq x < N \cdot 2^{-\sigma}$, and $0 < d < N$ $x \div d \leq xd \leq (x \div d) + 2$

1) B encrypts d, and sends $[d]$ and $\log_2 d$ to A.
2) A chooses random numbers r_d and r_m of sizes $\log_2 N - 1 - \log_2 d$ and $\log_2 d$ respectively. A computes $[r] = [r_d \cdot d + r_m] = [d]^{r_d} \cdot [r_m] \bmod N$.
3) A computes $[z] = [x + r] = [x] \cdot [r] \bmod N$ and sends it to B.
4) B decrypts $[z]$, computes $c = z \div d$, encrypts it, and sends $[c]$ to A.
5) A computes $[xd] = [c - r_d] = [c] \cdot [-r_d] \bmod N$.

$xd = (z \div d) - r_d$ is a good approximation of $x \div d$, in fact $(x \div d) \leq xd \leq (x \div d) + 2$.

Similar to Protocol 4, the limitations on the size of x can be eliminated at the cost of an extra comparison protocol.

Protocol 5 is more intensive than Protocol 3, 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 are constant rounds.

IV. Security proof

We have to show that our division 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.

More formally, in order to show that our protocols privately compute (or approximate) the division of two integers in the semi-honest model, we have to show that whatever can be computed by A or B from their view of a protocol execution, can be computed from their input and output (see Definition 7.2.1 in Goldreich [14]). Since we use the comparison protocol as a building block, we can present it as an oracle in our proofs and use Goldreich's Composition Theorem [14] to deduce that our protocols privately compute the division of two integers whenever the building block privately computes the comparison of two integers.

We work out the proof of Protocol 2, the other security proofs are analogue. In Protocol 2, the view of A consists of its private numbers $[x]$, and d, its random number r (of size $\log_2 N - 1$), its output $[x \div d]$, and all intermediate messages received from B: the encrypted comparison bit $[t]$, and the encrypted number $[c]$. Summarizing, the view of A equals

$$V_A = ([x], d, r, [x \div d], [t], [c])$$

It suffices to show that there exists a probabilistic polynomial-time algorithm S_A such that $S_A([x], d, [x \div d])$ is computationally indistinguishable from V_A [14]. Since the encryption algorithm is semantically secure, every pair of encryptions is computationally indistinguishable, so by letting S_A randomly generate an encrypted bit $[t_R]$, an encrypted integer $[c_R]$ of size $\log_2 d$, and a random number r_R of size $\log_2 N - 1$, 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. Summarizing, the view of B is equivalent to

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

Again, we have to show that there exists a probabilistic polynomial-time algorithm S_B such that $S_B(d, K)$ is computationally indistinguishable from V_B. This is easily satisfied by letting S_B randomly generate an integer z_R of $\log_2 N - 1$ bits. It follows that $\Pr(z) = \sum_r \Pr(z|r) \cdot \Pr(r) = 2^{-(\log_2 N - 1)} \sum_r \Pr(x = z - r) \leq 2^{-(\log_2 N - 1)}$, 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.

We conclude that Protocol 2 privately computes $[x \div d]$ 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 subsection II-B, where x and r can cover the entire plain text interval, this could be extended to perfect security towards B, so even with unbounded computational power.

V. Conclusions

We described four new protocols for dividing an encrypted number, each with their own merits. Two protocols approximate the division result, one with known and one with unknown divisor, both of which have a very low computational complexity. The other protocols compute the exact division result at the cost of an extra comparison protocol, Protocol 2 thereby improving the computational complexity of previous results. Suggestions where given for extending the protocols in order to avoid limitations on the size of the encrypted number, and improving the security properties.

References

[1] Paul Bunn and Rafail Ostrovsky, "Secure two-party k-means clustering," in *CCS'07*, USA, 2007.

[2] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L. Lagendijk, "Privacy-preserving user clustering in a social network," in *IEEE International Workshop on Information Forensics and Security*, 2009.

[3] Z. Erkin, M.T. Beye, Thijs Veugen, and R.L. Lagendijk, "Privacy enhanced recommender system," in *Thirty-first Symposium on Information Theory in the Benelux*, Rotterdam, 2010, pp. 35–42.

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

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

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

[7] Berry Schoenmakers and Pim Tuyls, "Efficient binary conversion for paillier encrypted values," in *Advances in Cryptology - EUROCRYPT 2006*. 2006, vol. 4004 of *Lecture Notes in Computer Science*, pp. 522–537, Springer.

[8] Thomas Jakobsen, "Secure multi-party computation on integers," Master Thesis University of Aarhus, Denmark, 2006.

[9] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Proceedings of the third Theory of Cryptography Conference, TCC 2006*. 2006, vol. 3876 of *Lecture Notes in Computer Science*, pp. 285–304, Springer-Verlag.

[10] Tomas Toft, *Primitives and Applications for Multi-party Computation*, Ph.D. thesis, University of Aarhus, Aarhus, Denmark, 2007.

[11] Tiziano Bianchi, Thijs Veugen, Alessandro Piva, and Mauro Barni, "Processing in the encrypted domain using a composite signal representation," in *SPEED'09*, Lausanne, Switzerland, Sept. 2009.

[12] Jorge Guajardo, Bart Mennink, and Berry Schoenmakers, "Modulo reduction for paillier encryptions and application to secure statistical analysis," in *SPEED'09*, Lausanne, Switzerland, Sept. 2009.

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

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