Improving the DGK comparison protocol

Thijs Veugen [#] [*1]

[#] *Information Security and Privacy Lab, Delft University of Technology*
Delft, The Netherlands

[*] *Technical Sciences, TNO*
Delft, The Netherlands
[1] thijs.veugen@tno.nl

***Abstract*—When processing signals in the encrypted domain, homomorphic encryption can be used to enable linear operations on encrypted data. Comparison of encrypted data however requires an additional protocol between the parties and will be relatively expensive. A well-known and frequently used comparison protocol is by Damgård, Geisler and Krøigaard. We present two ways of improving this comparison protocol. Firstly, we reduce the computational effort of one party by roughly 50%. Secondly, we show how to achieve perfect security towards the other party without additional costs, whereas the original version with encrypted inputs only achieved statistical security. An additional advantage is that larger inputs are allowed.**

I. Introduction

In 2007, Damgård, Geisler and Krøigaard (DGK) invented their secure comparison protocol [1] together with a new homomorphic cryptosystem that together formed an interesting and efficient solution for the so-called millionaire's problem. Their protocol has been used frequently ever since as a subprotocol in applications for signal processing with encrypted data. We mention a few.

In secure face recognition [2] a person is identified by comparing many face related values with a sample value. The same with fingerprints [3]. In secure statistical analysis [4] many sensitive statistical data have to be compared. In secure user clustering [5] user profiles have to be compared with cluster centroids. When generating private recommendations [6], user similarity values have to be compared with a threshold. Finally, also in secure adaptive filtering [7] or secure bioinformatics services [8], the DGK comparison protocol is used.

We finish Section 1 by describing related work followed by relevant background information in the preliminaries. In the second Section, the DGK comparison protocol is introduced and analysed. Our main contribution is described in Section 3 containing several ways of improving the DGK comparison protocol. The paper is finalized by the conclusions.

A. Related work

In 2001 Fischlin [9] presented his protocol using quadratic residues to encrypt bits and securely computing a boolean circuit presenting a comparison of two private numbers. His protocol requires roughly a factor 10 computational overhead compared to DGK and a factor λ communication overhead, where λ is an error probability parameter which should be around 86 to achieve a negligible probability of incorrectness [1]. Later, Blake and Kolesnikov [10], [11] described a protocol for comparing two numbers that are unknown to the parties requiring few communication but an exponentially large plain text and therefore much worse computational complexity than DGK. The protocol by Damgård, Geisler and Krøigaard essentially combines the best of both worlds yielding small computational and communication costs, requiring only one communication round.

In concurrent independent work, Garay, Schoenmakers and Villegas [12] develop a similar solution where the comparison result is to remain secret. They save computation and communication by allowing a number of communication rounds logarithmic in the number of input bits. Their constant round solution however is less efficient than the DGK solution.

Similarly, Schoenmakers and Tuyls [13] in 2004 found a very efficient comparison protocol in terms of computational complexity at the cost of a number of communication rounds linear in the number of input bits.

As with many secure two-party solutions in the semi-honest model, also secure comparison offers a perfect playground for garbled circuits as shown by Kolesnikov, Sadeghi and Schneider [14] in 2009. A disadvantage of garbled circuits based solutions is that they cannot achieve perfect security like DGK does for one party.

B. Preliminaries

The notation $(x \leq y)$ is used to denote the bit that will be one exactly when $x \leq y$, and $\oplus$ denotes the exclusive or of two bits. We use two different homomorphic cryptosystems in this paper to encrypt signals represented by integers.

The first one is the cryptosystem by Damgård, Geisler and Krøigaard (DGK) [1], [15] that is dedicated to small plaintexts and fits nicely within the secure comparison protocol. The public key is (n, g, h, u) and the private key is (p, q, v_p, v_q) such that the cipher text modulus n is the product of two large primes p en q. In the protocols we use K_{DGK} to denote the private key of the DGK crypto system. The plaintext space is $\mathbb{Z}_u$ where u is a small (16 or 32 bit) prime divisor of both

WIFS'2012, December, 2-5, 2012, Tenerife, Spain.
978-1-4673-2287-4/12/$31.00 ©2012 IEEE.

$p-1$ and $q-1$. The additional parameters v_p and v_q are t-bit prime divisors of $p-1$ and $q-1$ respectively, where a reasonable value for parameter t is 160. The numbers g and h are elements of $\mathbb{Z}_n^*$ of order uv_pv_q and v_pv_q respectively. The reasoning behind the values of all these parameters is explained in [15].

We denote a DGK encryption of plaintext $m \in \mathbb{Z}_u$ by $[m]$ which is computed as $[m] = g^m h^r \bmod n$, where r is a fresh random integer of $2t$ bits. A table can be used for decrypting $[m]$ [15] but in the comparison protocol we only want to determine whether $m = 0$ which can be done quite fast through the check $[m]^{v_pv_q} \bmod n = 1$. Since $u < p$ it is even sufficient to check $[m]^{v_pv_q} \bmod p = 1$ which will on average cost $\frac{3}{2}(t+t)/4 = \frac{3}{4}t$ multiplications modulo n.

The second cryptosystem is Paillier [16] with cipher text modulus N^2, N being a product of two large primes. The Paillier encryption of plaintext $m \in \mathbb{Z}_N$ is denoted by $[\![m]\!]$ and computed as $[\![m]\!] = g^m r^N \bmod N^2$, where r is a fresh random integer of size N and the order of $g \in \mathbb{Z}_{N^2}^*$ is a multiple of N. We choose $g = N+1$ because it reduces g^m to $1 + N \cdot m$ modulo N^2 and saves an exponentiation. The private key is denoted by $K_{Paillier}$ in our protocols. More details can be found in the paper [16].

Both cryptosystems are additively homomorphic so $[\![x]\!] \cdot [\![y]\!] = [\![x+y]\!] \bmod N^2$ and $[x] \cdot [y] = [x+y] \bmod n$, a property thas we will use frequently.

We assume the semi-honest model where both parties A and B follow the rules of the protocol, but collect as much information as possible to deduce private information. However, the DGK comparison can be extended to the malicious model with active adversaries [1].

The multiplicative inverse of x modulo n is denoted by x^{-1} and equals the integer y, $0 \le y < n$, such that $x \cdot y = 1 \bmod n$. The multiplicative inverse is efficiently computed by using the Euclidean algorithm [17], and can also be used to negate an encrypted integer: $[-x] \leftarrow [x]^{-1} \bmod n$. To estimate the computational complexity of the different protocols, we use the fact that an involution modulo n with an exponent of e bits will on average take $\frac{3}{2}e$ multiplications modulo n.

Finally, let σ be the statistical security parameter, which value is usually chosen around 80. Integer division is denoted by $\div$. And we assume all random variables, excluding the inputs of the secure multi-party computation protocol, are uniformly chosen.

II. Analysis of DGK comparison

When comparing two integers x and y bitwise, the obvious approach is to scan both bit rows from left (the most significant part) to right searching for the first differing bit. The outcome of the comparison of these differing bits will determine the comparison result of both integers. A similar approach is followed by the DGK protocol. Assume both integers contains ℓ bits denoted by x_i and y_i respectively, so $x = x_{\ell-1} \dots x_1 x_0$, $x_{\ell-1}$ being the most significant bit of x. Then the numbers c_i, $0 \le i < \ell$ are computed which will only be zero when $x_j = y_j$ for each j, $i < j < \ell$ and at the same time $x_i \neq y_i$.

More precisely,

$$c_i = s + x_i - y_i + 3 \sum_{j=i+1}^{\ell-1} (x_i \oplus y_i)$$

Clearly, the sum of exclusive ors will be zero exactly when $x_j = y_j$ for each j, $i < j < \ell$. The variable s, introduced later in [2], can be set to either -1 or 1 depending on the comparison that is performed. For example when $s = -1$, c_i will only be zero when $x_i = 1$ and $y_i = 0$ (and $x_j = y_j$ for each j, $i < j < \ell$) and thus $x > y$. To avoid one of the parties learning the comparison result, one party will set the parameter s and the other party will learn whether $c_i = 0$ or not.

The basic DGK comparison protocol is depicted in Protocol 1. In [1] more variants are described like shared inputs or achieving security against active adversaries. For a formal security proof we also refer to this paper.

Protocol 1 DGK comparison with private inputs

Party	A	B
Input	x	y and K_{DGK}
Output	$\delta_A \in \{0,1\}$	$\delta_B \in \{0,1\}$
Constraints	$\delta_A \oplus \delta_B = (x \le y)$ $0 \le x, y < 2^\ell$	

1) B sends the encrypted bits $[y_i]$, $0 \le i < \ell$ to A.
2) For each i, $0 \le i < \ell$, A computes $[x_i \oplus y_i]$ as follows:
   if $x_i = 0$ then $[x_i \oplus y_i] \leftarrow [y_i]$
   else $[x_i \oplus y_i] \leftarrow [1] \cdot [y_i]^{-1} \bmod n$.
3) A chooses a uniformly random bit δ_A and computes $s = 1 - 2 \cdot \delta_A$.
4) For each i, $0 \le i < \ell$, A computes $[c_i] = [s] \cdot [x_i] \cdot [y_i]^{-1} \cdot (\prod_{j=i+1}^{\ell-1} [x_j \oplus y_j])^3 \bmod n$.
5) A blinds the numbers c_i by raising them to a random exponent r_i of $2t$ bits: $[c_i] \leftarrow [c_i]^{r_i} \bmod n$, and sends them in random order to B.
6) B checks whether one of the numbers c_i is decrypted to zero. If he finds one, $\delta_B \leftarrow 1$, else $\delta_B \leftarrow 0$.

To show that in Protocol 1 indeed $\delta_A \oplus \delta_B = (x \le y)$, we distinguigh two cases:

- If $\delta_A = 0$ then $s = 1$ so $s + x_i - y_i$ is only zero when $x_i = 0$ and $y_i = 1$. Thus when B finds $c_i = 0$ (in which case $\delta_B = 1$), we have $x < y$, and otherwise $x \ge y$.
- If $\delta_A = 1$ then $s = -1$ so $s + x_i - y_i$ is only zero when $x_i = 1$ and $y_i = 0$. Thus when B finds $c_i = 0$, we have $x > y$, and otherwise $x \le y$.

In both cases, $\delta_A \oplus \delta_B = (x \le y)$. An extra measure described in Subsection II-A is needed to provide correctness in case $x = y$.

The value of B's input y is hidden from A by the DGK encryption system. On the other hand, A's input x is perfectly hidden from B (given some extra measures for the case $x = y$ as described in subsection II-A) because δ_A was uniformly chosen and party B only learns δ_B. Therefore, Protocol 1

realizes computational security towards A and perfect security towards B.

The main computational effort for A is in the multiplicative blinding of the numbers c_i which requires on average $\ell \cdot 3t$ multiplications modulo n. The main computational effort for B is the decryption (checks) of the same numbers c_i which requires on average $\ell \cdot \frac{3}{4}t$ multiplications modulo n.

The DGK protocol with private inputs is easily extended to encrypted inputs [18] as depicted in Protocol 2. The correctness and security of Protocol 2 is shown in the same paper [18].

Protocol 2 DGK comparison with encrypted inputs and statistical security

Party	A	B
Input	$[\![x]\!]$ and $[\![y]\!]$	$K_{Paillier}$ and K_{DGK}
Output	$[\![(x \leq y)]\!]$	
Constraints	$0 \leq x, y < 2^\ell$ and $\ell + \sigma < \log_2 N$	

1) A chooses a random number r of $\ell + 1 + \sigma$ bits, and computes $[\![z]\!] \leftarrow [\![x - y + 2^\ell + r]\!] = [\![x]\!] \cdot [\![y]\!]^{-1} \cdot [\![2^\ell + r]\!] \bmod N^2$. A sends $[\![z]\!]$ to B.
2) B decrypts $[\![z]\!]$, and computes $\beta = z \bmod 2^\ell$.
3) A computes $\alpha = r \bmod 2^\ell$.
4) A and B run a DGK comparison protocol with private inputs α and β resulting in outputs δ_A and δ_B such that $\delta_A \oplus \delta_B = (\alpha \leq \beta)$.
5) B computes $z \div 2^\ell$ and sends $[\![z \div 2^\ell]\!]$ and $[\![\delta_B]\!]$ to A.
6) A computes $[\![(\beta < \alpha)]\!]$ as follows:
   if $\delta_A = 1$ then $[\![(\beta < \alpha)]\!] \leftarrow [\![\delta_B]\!]$
   else $[\![(\beta < \alpha)]\!] \leftarrow [\![1]\!] \cdot [\![\delta_B]\!]^{-1} \bmod N^2$.
7) A computes $[\![(x \leq y)]\!] \leftarrow [\![z \div 2^\ell]\!] \cdot ([\![r \div 2^\ell]\!] \cdot [\![(\beta < \alpha)]\!])^{-1} \bmod N^2$.

In Protocol 2 the comparison $(x \leq y)$ is reduced to the private comparison $(\alpha \leq \beta)$ [18]. As in Protocol 1, it realizes computational security towards A. Since the value $x - y$ is statistically hidden in z, the probability $\Pr(x - y|z)$ is not uniform and depends on z and therefore Protocol 2 provides only statistical security towards B. For example when $z = r_{min} - 1$, B will know that $x = 0$ and $y = 2^\ell - 1$.

A. Equality of inputs

When $x \neq y$, none or one of the values c_i will be zero depending on the (uniform) choice of δ_A, so δ_B will be uniformly distributed and independent from the random distributions of inputs x and y. However, when $x = y$ there will never occur a zero in the c_i, irrespective of δ_A, because the part $s + x_i - y_i$ will never equal zero. So some information is leaked towards B in case of equality of inputs. This is due to the introduction of the variable s in [2], but they did not mention the problem of information leakage.

As personally communicated by Tomas Toft, an easy way to overcome this information leakage is to introduce an extra variable c_{-1} that will be zero when $x = y$ with probability $\frac{1}{2}$ and not zero otherwise.

$$c_{-1} = \delta_A + \sum_{i=0}^{\ell-1} x_i \oplus y_i$$

Party B will set $\delta_B \leftarrow 1$ only when one of the variables $c_i = 0$, $-1 \leq i < \ell$, and $\delta_B \leftarrow 0$ otherwise. This also assures that $\delta_A \oplus \delta_B = (x \leq y)$ even in the case of equality.

With this extra measure in Protocol 1, perfect security is achieved towards B. The variable δ_B will be uniformly distributed independent of the random distributions of x and y.

III. Improvements

We present two different ways to improve the DGK comparison algorithm. The first improvement significantly reduces the computational complexity of Protocol 1, and the second improvement provides perfect security towards B for Protocol 2 without substantially reducing the performance.

A. Computational complexity

The computational complexity of Protocol 1 can be reduced in two ways. The first, major improvement is achieved by carefully considering the cases where $c_i = 0$ leading to a reduction in step 5 that requires the highest computational effort within Protocol 1. The observation that $c_i > 0$ when $x_i \neq \delta_A$ independent of the value y leads to the definition of the set $\mathcal{L} = \{i \mid 0 \leq i < \ell \text{ and } x_i = \delta_A\}$. Since $c_i > 0$ whenever $i \notin \mathcal{L}$, these elements $[c_i]$ can be replaced by random non-zero elements and don't need to be multiplicatively blinded in step 5. This is depicted in Protocol 3.

The set $\mathcal{L}$ will on average contain $\ell/2$ elements leading to reduction of the computational complexity in step 5 of 50%. And since this step determines the complexity of party A, Protocol 3 reduces the average computational complexity of party A in Protocol 1 by 50% (see Subsection III-C).

The second, minor improvement is achieved by removing the exponent three in step 4 saving 2ℓ muliplications modulo n in total. This is depicted in step 4 of Protocol 3. Because the first part of c_i is either zero or one, the second part containing the sum no longer requires a factor three.

1) Timing attacks: The computational optimizations described above introduce a practical weakness in the protocol. This is due to the fact that the computational effort depends on the value x so by measuring the time or the power consumption party B could learn information about private input x.

One way to overcome this vulnerability is to use additional timers or dummy execution steps such that the execution time will be constant. This however invalidates the introduced benefits of reduced processing time.

Another solution is to add a buffer of precomputations to be executed. Whenever the situation permits, precomputations can be done and stored for later usage. Precomputations could for example consist of random factors that are needed while encrypting values. By using 'idle' time for precomputing random values, the overall computational complexity and

Protocol 3 Optimized DGK comparison with private inputs

Party	A	B
Input	x	y and K_{DGK}
Output	$\delta_A \in \{0,1\}$	$\delta_B \in \{0,1\}$
Constraints	$\delta_A \oplus \delta_B = (x \leq y)$ $0 \leq x, y < 2^\ell$	

1) B sends the encrypted bits $[y_i]$, $0 \leq i < \ell$ to A.
2) For each i, $0 \leq i < \ell$, A computes $[x_i \oplus y_i]$ as follows:
   if $x_i = 0$ then $[x_i \oplus y_i] \leftarrow [y_i]$
   else $[x_i \oplus y_i] \leftarrow [1] \cdot [y_i]^{-1} \bmod n$.
3) A chooses a uniformly random bit δ_A. Let $\mathcal{L}$ be the set $\{i \mid 0 \leq i < \ell \text{ and } x_i = \delta_A\}$.
4) For each $i \in \mathcal{L}$:
   A computes $[c_i] = \prod_{j=i+1}^{\ell-1} [x_j \oplus y_j] \bmod n$.
   If $\delta_A = 0$ then $[c_i] \leftarrow [1] \cdot [y_i]^{-1} \cdot [c_i] \bmod n$
   else $[c_i] \leftarrow [y_i] \cdot [c_i] \bmod n$.
5) For each $i \in \mathcal{L}$, A blinds the numbers c_i by raising them to a random exponent r_i of $2t$ bits: $[c_i] \leftarrow [c_i]^{r_i} \bmod n$. For the remaining $i \notin \mathcal{L}$ a random non-zero $[c_i] \leftarrow [r_i]$ is generated en encrypted.
   A sends all $[c_i]$ in random order to B.
6) B checks whether one of the numbers c_i is decrypted to zero. If he finds one, $\delta_B \leftarrow 1$, else $\delta_B \leftarrow 0$.

execution time of the protocol will reduce. To show that even the worst-case execution time of our protocol can be reduced by such precomputations, consider the randomization step $[c_i] \leftarrow [c_i]^{r_i} \bmod n$ that has to be performed before A can send the value c_i to party B. The same (and from a security perspective probably preferable) effect is achieved by randomizing c_i through $[c_i] \leftarrow [c_i]^{s_i} \cdot h^{r_i} \bmod n$, where r_i is the same random value of $2t$ bits, but s_i is a considerably smaller random value of size u [1]. The random factors $h^{r_i} \bmod n$ can be easily precomputed in which case the randomization effort of c_i is reduced by a factor $2t/\log_2 u = 10$ when u consists of 32 bits.

However, formally there is no security problem as all our protocols are provably secure in the semi-honest model [1], [18]. This is argued by considering the detailed proof in section 4.2 of the original paper [1]. Since the main difference is in the computation of the $[c_i]$, we especially have to simulate these messages from A to B. Each encrypted nonzero c_i is easily simulated by an encrypted random nonzero element of $\mathbb{Z}_u^*$. Since we use the same randomizations (as described above) as the original protocol, the simulated messages will be statistically indistinguishable from the real protocol messages by the same arguments.

Many cryptographic protocols suffer from potential timing attacks. Also, the comparison protocol is always used as a subprotocol within an application, and it's not always possible to time the execution of a particular comparison protocol.

α_i	0	1	0	1	0	1	0	1
β_i	0	0	1	1	0	0	1	1
d	0	0	0	0	1	1	1	1
$\alpha_i \oplus \beta_i$	0	1	1	0	0	1	1	0
w_i	0	1	1	0	-1	0	0	-1
$\tilde{\alpha}_i \oplus \beta_i$					1	0	0	1

TABLE I
THE VALUE w_i WHEN $\alpha_i \neq \tilde{\alpha}_i$

B. Security properties

In Protocol 2, no carry-over modulo n is allowed in the addition of $x - y + 2^\ell$ and r leading to only statistical security towards B. If r could be chosen from the full range $0 \leq r < N$, the value z would perfectly mask the secret value $x - y$, and perfect security could be achieved towards B.

Protocol 4 shows how to adjust the DGK comparison protocol with encrypted inputs such that perfect security is achieved towards B requiring only a small increase in computational and communication complexity. The difference with Protocol 2 is the modified subprotocol with private inputs.

The idea is that B sends an encrypted bit $[d]$ to A 'informing' A whether a carry-over has occurred in the addition of $x - y + 2^\ell$ and r. A can use this additional encrypted bit to compute numbers c_i, $0 \leq i < \ell$, similar to the original Protocol 1. An additional advantage of allowing carry-overs in Protocol 4 is that the inputs x and y are allowed to be larger than in Protocol 2.

To ensure that bit $d = 1$ exactly when a carry-over has occurred, we require $\ell + 2 < \log_2 N$ such that $0 \leq x - y + 2^\ell < (N-1)/2$. This means that we pay the price of not allowing input values consisting of $\log_2 N - 2$ or $\log_2 N - 1$ bits to ensure that $z - r$ will also be in the first half of the interval $[0, N)$, i.e. $0 \leq z - r < (N-1)/2$. When $0 \leq r < (N-1)/2$, party A will be assured that no carry-over has occurred. Otherwise, when r is in the second half of the interval $[0, N)$, the comparison $z < (N-1)/2$ (which can be performed by B) will inform party A about the carry-over.

Depending on the value of d a different comparison should be executed (see Equation 1). When $d = 0$, $z = x - y + 2^\ell + r$ and the original comparison $\alpha \leq \beta$ should be computed, but in case a carry-over occurred ($d = 1$ and $z = x - y + 2^\ell + r - N$), the comparison $\tilde{\alpha} \leq \beta$ should be performed where the non-negative integer $\tilde{\alpha} = (r - N) \bmod 2^\ell$.

The most important part of the modified subprotocol is in the computation of the encrypted values w_i that should approximate $\alpha_i \oplus \beta_i$ in case no carry-over occurred, and $\tilde{\alpha}_i \oplus \beta_i$ when a carry-over actually did occur. When $\alpha_i = \tilde{\alpha}_i$ this is obviously true. The most interesting case is $\alpha_i \neq \tilde{\alpha}_i$ when $w_i = (\alpha_i \oplus \beta_i) - d$.

As can be deduced from Table I, w_i will be zero in exactly the right cases. That is, $w_i = 0$ when $\alpha_i \oplus \beta_i = 0$ and $d = 0$, but also when $\tilde{\alpha}_i \oplus \beta_i = 0$ and $d = 1$. Furthermore, $w_i \in \{-1, 1\}$ in all other cases.

By multiplying each w_i with a factor 2^i in step 4(f) of the protocol, we can assure that in step 4(h) the sum $\sum_{j=i+1}^{\ell-1} w_j =$

Protocol 4 DGK with encrypted inputs and perfect security

Party	A	B
Input	$[\![x]\!]$ and $[\![y]\!]$	$K_{Paillier}$ and K_{DGK}
Output	$[\![(x \leq y)]\!]$	
Constraints	$0 \leq x, y < 2^\ell$ and $\ell + 2 < \log_2 N$	

1) A chooses a random number r, $0 \leq r < N$, and computes $[\![z]\!] \leftarrow [\![x - y + 2^\ell + r]\!] = [\![x]\!] \cdot [\![y]\!]^{-1} \cdot [\![2^\ell + r]\!] \bmod N^2$. A sends $[\![z]\!]$ to B.
2) B decrypts $[\![z]\!]$, and computes $\beta = z \bmod 2^\ell$.
3) A computes $\alpha = r \bmod 2^\ell$.
4) A and B run a *modified* DGK comparison protocol with private inputs α and β resulting in outputs δ_A and δ_B:
   a) B sends the encrypted bit $[d]$ where $d = (z < (N-1)/2)$ is the bit informing A whether a carry-over has occured.
   b) B sends the encrypted bits $[\beta_i]$, $0 \leq i < \ell$ to A.
   c) A corrects $[d]$ by setting $[d] \leftarrow [0]$ whenever $0 \leq r < (N-1)/2$.
   d) For each i, $0 \leq i < \ell$, A computes $[\alpha_i \oplus \beta_i]$ as follows:
   if $\alpha_i = 0$ then $[\alpha_i \oplus \beta_i] \leftarrow [\beta_i]$
   else $[\alpha_i \oplus \beta_i] \leftarrow [1] \cdot [\beta_i]^{-1} \bmod n$.
   e) A computes $\tilde{\alpha} = (r - N) \bmod 2^\ell$, the corrected value of α in case a carry-over actually did occur and adjusts $[\alpha_i \oplus \beta_i]$ for each i:
   If $\alpha_i = \tilde{\alpha}_i$ then $[w_i] \leftarrow [\alpha_i \oplus \beta_i]$
   else $[w_i] \leftarrow [\alpha_i \oplus \beta_i] \cdot [d]^{-1} \bmod n$
   f) For each i, $0 \leq i < \ell$, A computes $[w_i] \leftarrow [w_i]^{2^i} \bmod n$ such that these values will not interfere each other when added.
   g) A chooses a uniformly random bit δ_A and computes $s = 1 - 2 \cdot \delta_A$.
   h) For each i, $0 \leq i < \ell$, A computes $[c_i] = [s] \cdot [\alpha_i] \cdot [d]^{\tilde{\alpha}_i - \alpha_i} \cdot [\beta_i]^{-1} \cdot (\prod_{j=i+1}^{\ell-1} [w_j])^3 \bmod n$.
   i) A blinds the numbers c_i by raising them to a random exponent r_i of $2t$ bits: $[c_i] \leftarrow [c_i]^{r_i} \bmod n$, and sends them in random order to B.
   j) B checks whether one of the numbers c_i is decrypted to zero. If he finds one, $\delta_B \leftarrow 1$, else $\delta_B \leftarrow 0$.
5) B computes $z \div 2^\ell$ and sends $[\![z \div 2^\ell]\!]$ and $[\![\delta_B]\!]$ to A.
6) A computes $[\![(\beta < \alpha)]\!]$ as follows:
   if $\delta_A = 1$ then $[\![(\beta < \alpha)]\!] \leftarrow [\![\delta_B]\!]$
   else $[\![(\beta < \alpha)]\!] \leftarrow [\![1]\!] \cdot [\![\delta_B]\!]^{-1} \bmod N^2$.
7) A computes $[\![(x < y)]\!] \leftarrow [\![z \div 2^\ell]\!] \cdot ([\![r \div 2^\ell]\!] \cdot [\![(\beta < \alpha)]\!])^{-1} \bmod N^2$.

0 exactly when all individual $w_j = 0$.

The final difference with Protocol 2 is that we use $[\alpha_i] \cdot [d]^{\tilde{\alpha}_i - \alpha_i}$ instead of $[\alpha_i]$ in step 4(h). In effect, when $d = 0$ it will equal $[\alpha_i]$ and when $d = 1$ it will be $[\tilde{\alpha}_i]$. So the right value is used depending on whether a carry-over occurred or not.

Because the absolute value of $s + \alpha_i + d \cdot (\tilde{\alpha}_i - \alpha_i) - \beta_i$ in step 4(h) is bounded by two, the factor three in $3 \sum_{j=i+1}^{\ell-1} w_j$ avoids interference with this value, so c_i will eventually be zero only when both parts are zero.

We conclude that

$$\delta_A \oplus \delta_B = \begin{cases} (\alpha \leq \beta) & \text{, if } d = 0 \\ (\tilde{\alpha} \leq \beta) & \text{, if } d = 1 \end{cases} \quad (1)$$

1) Optimizations: We describe three ways of optimizing the computational complexity of Protocol 4, and in particular its subprotocol of step 4.

First, the exponentiations in step 4(f) require $\sum_{i=0}^{\ell-1} i = \frac{1}{2}(\ell - 1)\ell$ multiplications modulo n which is quite a lot. This can be reduced by carefully analyzing the construction of the w_i. The factor 2^i is needed to avoid interference between the different values when they are added in step 4(h). When $d = 0$ all $w_i = (\alpha_i \oplus \beta_i)$ will be either zero or one so then any positive factor can be used to avoid interference. When $d = 1$, $w_i \in \{-1, 0, 1\}$, and more precisely $w_i = (\alpha_i \oplus \beta_i) \in \{0, 1\}$ when additionally $\alpha_i = \tilde{\alpha}_i$, and $w_i = (\alpha_i \oplus \beta_i) - 1 \in \{-1, 0\}$ otherwise. So when $d = 1$, the sum $\sum_{j=i+1}^{\ell-1} w_j$ can be split into a non-negative part $\sum_{j=i+1, \alpha_j = \tilde{\alpha}_j}^{\ell-1} w_j$ and a non-positive part $\sum_{j=i+1, \alpha_j \neq \tilde{\alpha}_j}^{\ell-1} w_j$. Therefore, a factor ℓ for the w_j in the second part will suffice ensuring that the total sum can only be zero when all individual elements are zero.

This leads to the following optimization in step 4(f):

$$[w_i] \leftarrow [w_i]^\ell \bmod n \text{ only when } \alpha_i \neq \tilde{\alpha}_i$$

Maximally $\ell \frac{3}{2} \log_2 \ell$ multiplications modulo n are required for this optimized step 4(f) which is less than the computational bottleneck of the protocol in step 4(i).

Another reason to introduce this optimization is that DGK encryption requires the plain texts to remain small (16 or 32 bits) [1]. Our modification reduces the absolute value of the numbers c_i in step 4(h) from roughly 2^ℓ to ℓ^2.

The second optimization is similar to the one described in Subsection III-A. The set $\mathcal{L}$ can be defined as $\{0 \leq i < \ell \mid (\alpha_i = \delta_A) \text{ or } (\tilde{\alpha}_i = \delta_A)\}$. When $i \notin \mathcal{L}$, neither α_i nor $\tilde{\alpha}_i$ will equal δ_A, so c_i will never be zero independent of the fact whether a carry-over occurred. On average 25% of the ℓ elements will lie outside $\mathcal{L}$, so the computational complexity of Party A in Protocol 4 (in particular of step 4(i)) will be roughly reduced by a factor 25% (see Subsection III-C).

By considering step 4(c), a final optimization can be deduced. Namely, when party A is certain that no carry-over has occurred, there is no need for executing the steps 4(c) upto 4(i) to compute the values c_i. Instead, the computationally less intensive steps 2) upto 5) from Protocol 3 could be performed. From B's point of view, there is no difference between Protocol 3 and the modified version, it only affects the way that party A computes the numbers c_i. In particular, steps 4(c) upto 4(i) could be optimized as follows:
If $r + 2^{\ell+1} < N$ then A executes steps 2) upto 5) from Protocol 3 (with private inputs α and β instead of x and y), and otherwise A performs steps 4(c) upto 4(i) from Protocol 4.

Since our optimization with set $\mathcal{L}$ improves A's computational complexity by a factor 50% in Protocol 3 and by 25% in Protocol 4, the modification above leads to an average improvement by a factor $\frac{N-2^{\ell+1}}{N} \cdot 50\% + \frac{2^{\ell+1}}{N} \cdot 25\%$ which is very close to 50% for most values of ℓ. A disadvantage of the modification above is that it might lead to additional timing attacks with respect to r as described in Subsection III-A1.

C. Comparison of performance

To determine the value of our improvements the average total number of multiplications modulo n is computed and compared to the original Protocol 1. Since the main computational difference between Protocol 1 and Protocol 2 is B's decryption of z, the computational effort for party A will be comparable for both protocols. We assume that all random factors are precomputed and only depict the effort for party A, because our optimizations only affect party A. Also depicted is the performance of Protocol 4 including the first two optimizations from Subsection III-B1 but excluding the third one.

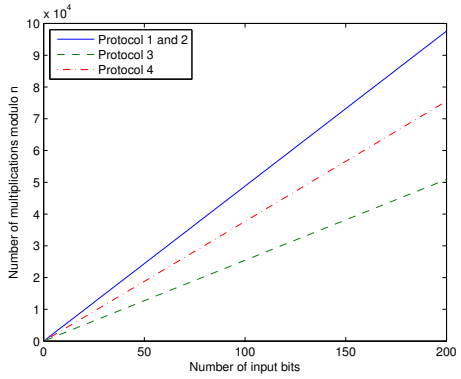

Fig. 1. Average computational complexity for party A

The results depicted in Figure 1 confirm our expectations that Protocol 3 and Protocol 4 reduce the average computational complexity of party A in Protocols 1 and 2 by 50% and 25% respectively. Because in Protocol 1 the computational effort of party B is roughly 25% of A's effort, the average computational complexity of the entire protocol will be reduced by 40% and 20% respectively. If the third optimization from Subsection III-B1 had been included, the performance of Protocols 3 and 4 would have been identical at the cost of introducing extra timing vulnerabilities.

IV. Conclusions

We carefully analyzed the widely used secure comparison protocol by Damgård, Geisler and Krøigaard [1], [15] and presented two improvements. Firstly, we were able to reduce the computational effort of party A by roughly 50%. Secondly, we showed how to achieve perfect security towards party B without additional costs in the variation with encrypted inputs, whereas the original version only achieved statistical security. An additional advantage is that larger inputs are allowed which is particularly interesting when packing [6] is used which allows additional computational and communication advantages.

References

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

[2] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, R. L. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Proceedings of the Privacy Enhancing Technologies Symposium*, Seattle, USA, 2009, pp. 235–253.

[3] M. Barni, T. Bianchi, D. Catalano, M. D. Raimondo, R. D. Labati, and P. Failla, "Privacy-preserving fingercode authentication," in *Workshop on Multimedia and Security*, 2010.

[4] J. Guajardo, B. Mennink, and B. Schoenmakers, "Modulo reduction for Paillier encryptions and application to secure statistical analysis," in *SPEED'09*, Lausanne, Switzerland, sep 2009.

[5] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving user clustering in a social network," in *IEEE International Workshop on Information Forensics and Security*, 2009.

[6] ——, "Generating private recommendations efficiently using homomorphic encryption and data packing," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 1053–1066, 2012.

[7] J. Troncoso-Pastoriza and F. Perez-Gonzalez, "Secure adaptive filtering," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 469 – 485, 2011.

[8] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröeder, "Towards secure bioinformatics services," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, vol. 7035, 2012, pp. 276–283.

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

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

[11] ——, "Conditional encrypted mapping and comparing encrypted numbers," in *Financial Crypto*, vol. 4107. LNCS, 2006.

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

[13] B. Schoenmakers and P. Tuyls, "Practical two-party computation based on the conditional gate," in *ASIACRYPT'04*, ser. Lecture Notes in Computer Science, no. 3329, Advances in Cryptology. Springer, 2004, pp. 119–136.

[14] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, ser. Lecture Notes in Computer Science, vol. 5888. Springer-Verlag, 2009, pp. 1–20.

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

[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of Eurocrypt 1999*, ser. Lecture Notes in Computer Science, vol. 1592. Springer-Verlag, 1999, pp. 223–238. [Online]. Available: citeseer.ist.psu.edu/article/paillier99publickey.html

[17] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied cryptography*. CRC Press, 1996.

[18] T. Veugen, "Encrypted integer division," in *IEEE Workshop on Information Forensics and Security*, Dec 2010.