Correction to ”Improving the DGK comparison protocol”

Thijs Veugen [#] [*1]

[#] *Department of Cryptology, Centrum Wiskunde & Informatica*
Amsterdam, The Netherlands

[*] *Unit ICT, TNO*
The Hague, The Netherlands
[1] thijs.veugen@tno.nl

***Abstract*—At the IEEE Workshop on Information Forensics and Security in 2012, Veugen introduced two ways of improving a well-known secure comparison protocol by Damgård, Geisler and Krøigaard, which uses additively homomorphic encryption. The first new protocol reduced the computational effort of one party by roughly 50%. The second one showed how to achieve perfect security towards one party without additional costs, whereas the original version with encrypted inputs only achieved statistical security. However, the second protocol contained a mistake, leading to incorrect outputs in some cases. We show how to correct this mistake, without increasing its computational complexity.**

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 sub-protocol 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.

In the remainder of this section, we mention related work, and repeat the preliminaries from [9]. In the second section, the DGK comparison protocol is introduced and analysed, as in [9]. In Section 3 we show how to improve the security properties of the DGK comparison protocol, and explain how to correct the flaws from [9]. The paper is finalized by the conclusions.

A. Related work

In 2017, Baptiste Vinh Mau and Koji Nuida [10] noted the error in [9], and described a way to correct the flaw. Unfortunately, they introduced an additional secure multiplication protocol that securely multiplies two additively-homomorphic encrypted values, leading to a substantial increase of the computational (and communication) complexity of Veugen's approach. We propose an alternative solution without needing a secure multiplication protocol.

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], [11] 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 $p-1$ 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 [11].

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 $[m]$ [11] 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 [12] 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 [12].

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, $0 \leq y < n$, such that $x \cdot y = 1 \bmod n$. The multiplicative inverse is efficiently computed by using the Euclidean algorithm [13], 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 $x = x_{\ell-1} \ldots x_1 x_0$, $x_{\ell-1}$ being the most significant bit of x. Then the numbers c_i, $0 \leq i < \ell$ 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.

To show that in Protocol 1 indeed $\delta_A \oplus \delta_B = (x \leq y)$, 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 $x \geq y$.

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	$\delta_A \oplus \delta_B = (x \leq y)$ $0 \leq x, y < 2^\ell$	

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 and computes $s = 1 - 2 \cdot \delta_A$.
4) For each i, $0 \leq i < \ell$, 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 non-zero exponent $r_i \in \{1, \ldots, u-1\}$, and refreshing the randomness with a second exponent r'_i of $2t$ bits: $[c_i] \leftarrow [c_i]^{r_i} \cdot h^{r'_i} \bmod n$, 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 $\delta_B \leftarrow 0$.

- 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 $x \leq y$.

In both cases, $\delta_A \oplus \delta_B = (x \leq y)$. 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 [14] as depicted in Protocol 2. The correctness and security of Protocol 2 is shown in the same paper [14].

In Protocol 2 the comparison $(x \leq y)$ is reduced to the private comparison $(\alpha \leq \beta)$ [14]. 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 $y = 2^\ell - 1$.

Protocol 2 DGK comparison with encrypted inputs and statistical security

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

1) A chooses a random number r of $\ell + 1 + \sigma$ bits, and computes $[\![z]\!] \leftarrow [\![y - x + 2^\ell + r]\!] = [\![y]\!] \cdot [\![x]\!]^{-1} \cdot [\![2^\ell + r]\!] \bmod N^2$. 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 $\delta_A \oplus \delta_B = (\alpha \le \beta)$.
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 $[\![(x \le y)]\!] \leftarrow [\![z \div 2^\ell]\!] \cdot ([\![r \div 2^\ell]\!] \cdot [\![(\beta < \alpha)]\!])^{-1} \bmod N^2$.

A. Equality of inputs

When $x \ne y$, 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 $c_i = 0$, $-1 \le i < \ell$, and $\delta_B \leftarrow 0$ otherwise. This also assures that $\delta_A \oplus \delta_B = (x \le y)$ 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. Improving the security

We show how to provide perfect security towards B for Protocol 2 without substantially reducing the performance.

A. The solution from [9]

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 $0 \le r < N$, the value z would perfectly mask the secret value $x - y$, and perfect security could be achieved towards B.

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 $\alpha_i \ne \tilde{\alpha}_i$

Protocol 3 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, $0 \le i < \ell$, similar to the original Protocol 1. An additional advantage of allowing carry-overs in Protocol 3 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 $0 \le x - y + 2^\ell < (N-1)/2$. 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. $0 \le z - r < (N-1)/2$. When $0 \le r < (N-1)/2$, 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 $\alpha \le \beta$ should be computed, but in case a carry-over occurred ($d = 1$ and $z = x - y + 2^\ell + r - N$), the comparison $\tilde{\alpha} \le \beta$ 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 $\alpha_i \ne \tilde{\alpha}_i$ 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 $\sum_{j=i+1}^{\ell-1} w_j = 0$ 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

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

For further optimizing the computational complexity of Protocol 3, and in particular its subprotocol of step 4, see [9].

B. Incorrect output

As pointed out in [10], in Protocol 3 the value $(\beta < \alpha)$ is correctly computed, even in case of overflow during the computation of z. The problem is the value $z \div 2^\ell$ used in step 7, which is sometimes too small, because a large r led to an overflow in the computation of z in step 1, thereby reducing the value of z by N.

To correct this overflow problem, we need to compute $z \div 2^\ell$, in case $d = 0$, so no overflow occurred, and we need $(z + N) \div 2^\ell$, in case there was an overflow and $d = 1$. The solution of [10] was to securely multiply $[\![z]\!]$ and $[\![((z+N) \div 2^\ell) - (z \div 2^\ell)]\!]$, and add this encrypted product to $[\![(x < y)]\!]$ in step 7. However, this intensive secure multiplication can be avoided as follows:

- In step 5, party B not only computes $\zeta_1 = z \div 2^\ell$, but also ζ_2, where $\zeta_2 = (z + N) \div 2^\ell$, if $z < (N-1)/2$, and $\zeta_2 = z \div 2^\ell$, otherwise. Party B encrypts both ζ_1 and ζ_2, and sends them to A.
- In step 7, party A uses $[\![\zeta]\!]$ instead of $[\![z \div 2^\ell]\!]$, where $\zeta = \zeta_1$, if $r < (N-1)/2$, and $\zeta = \zeta_2$, otherwise.

In this way, the parties can ensure that $\zeta = z \div 2^\ell$ in case no overflow occurred (r small or z large), and $\zeta = (z + N) \div 2^\ell$ in case an overflow did occur (r large and z small).

IV. Conclusions

In [9] the widely used secure comparison protocol by Damgård, Geisler and Krøigaard [1], [11] was carefully analyzed, and two improvements were presented. The second improvement achieved perfect security towards party B without additional costs in the variation with encrypted inputs, whereas the original version only achieved statistical security. As pointed out in [10], the output of this improved protocol was not computed correctly. We showed how to correct this flaw without increasing the computational complexity of the protocol.

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] T. Veugen, "Improving the dgk comparison protocol," in *IEEE Workshop on Information Security and Forensics*, 2012, pp. 49–54.

[10] B. V. Mau and K. Nuida, "Correction of a secure comparison protocol for encrypted integers in IEEE WIFS 2012," in *IWSEC 2017*, ser. LNCS, S. Obana and K. Chida, Eds., vol. 10418, 2017, pp. 181–191.

[11] 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.

[12] 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

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

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

Protocol 3 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 $[\![z]\!] \leftarrow [\![y - x + 2^{\ell} + r]\!] = [\![y]\!] \cdot [\![x]\!]^{-1} \cdot [\![2^{\ell} + r]\!] \bmod N^2$. 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 non-zero exponent $r_i \in \{1, \ldots, u-1\}$, and refreshing the randomness with a second exponent r'_i of $2t$ bits: $[c_i] \leftarrow [c_i]^{r_i} \cdot h^{r'_i} \bmod n$, 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 $[\![(x \leq y)]\!] \leftarrow [\![z \div 2^{\ell}]\!] \cdot ([\![r \div 2^{\ell}]\!] \cdot [\![(\beta < \alpha)]\!])^{-1} \bmod N^2$.