Journal of King Saud University – Computer and Information Sciences (2016) **28**, 27–36

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Privacy preserving cloud computation using Domingo-Ferrer scheme

Abdulatif Alabdulatif [a,b,*], **Mohammed Kaosar** [c]

[a] *School of Computer Science, RMIT University, Melbourne, Australia*
[b] *The School of Computer Science, Qassim University, Saudi Arabia*
[c] *Computer Science Department, Effat University, Saudi Arabia*

Received 21 October 2014; revised 20 October 2015; accepted 22 October 2015
Available online 6 November 2015

KEYWORDS

Homomorphic encryption;
Arithmetic operations;
Maximum/minimum function;
Cloud computing;
Cloud-based applications

Abstract Homomorphic encryption system (HES) schemes are anticipated to play a significant role in cloud-based applications. Moving to cloud-based storage and analytic services securely are two of the most important advantages of HES. Several HES schemes have been recently proposed. However, the majority of them either have limited capabilities or are impractical in real-world applications. Various HES schemes provide the ability to perform computations for statistical analysis (e.g. average, mean and variance) on encrypted data. Domingo-Ferrer is one scheme that has privacy homomorphism properties to perform the basic mathematical operations (addition, subtraction and multiplication) in a convenient and secure way. However, it works only in the positive numbers' range which is considered as a limitation because several applications require both positive and negative ranges in which to work, especially those that have to implement analytical services in cloud computing. In this paper, we extend Domingo-Ferrer's scheme to be able to perform arithmetic operations for both positive and negative numbers. We also propose using a lightweight data aggregation function to compute both maximum and minimum values of the aggregated data that works for both positive and negative numbers.

1. Introduction

The essential idea behind the efforts to involve homomorphic encryption system (HES) techniques in practical applications is to employ the advantages of cloud computing services and resources, as these techniques provide a convenient and secure environment for uploading private information to a cloud. HES has existed since the inception of public key cryptography. Examples of HES include Rivest et al. (1978), ElGamal (1985), Benaloh (1994) and Paillier (1999), to name a few. However, these are somewhat homomorphic encryption systems (SHES), meaning they only support either addition or

* Corresponding author at: School of Computer Science, RMIT University, Melbourne, Australia.
E-mail addresses: abdulatif.alabdulatif@rmit.edu.au (A. Alabdulatif), mkaosar@effatuniversity.edu.sa (M. Kaosar).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2015.10.001

multiplication, not both. A number of these schemes fail to be appropriate for practical applications because they are often inefficient and are unable to perform many required arithmetic operations to build useful applications in cloud environments.

A fully homomorphic encryption system (FHES) would support several operations simultaneously. This was a problem until the breakthrough result of Gentry (2009), which is based on the properties of ideal lattices (Bayer-Fluckiger, 2002). This scheme is still quite impractical for real-life applications because of its limitations in arithmetic operations, time consumption and the amount of resources that are required for computations. Shortly after, a FHES was introduced that used only elementary arithmetic operations (Van Dijk et al., 2010). While this modified scheme reduced the complexity of a fully homomorphic encryption process by allowing it to be described in simple terms, it is still complex enough to be useful in any real-life applications. The aforementioned FHES schemes and other similar schemes, such as Gentry (2009, 2010), have a common drawback, which includes computational overhead in terms of efficiency and execution time. This led to the conclusion that these schemes are ineffective for real-life applications, especially in a cloud-based environment.

The Domingo-Ferrer encryption scheme (Domingo-Ferrer, 2002) is considered a lightweight scheme that has the ability to perform various arithmetic computations in a secure manner based on homomorphic properties, and it can be a possible candidate in various practical cloud-based applications. We believe that the Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme (Domingo-Ferrer, 2002) is one of the most applicable HESs that can perform main basic arithmetic operations, which include addition, subtraction and multiplication. This is done in appropriate and secure ways such as through statistical analysis services in wire and wireless sensor networks (WSN), where aggregated data are analysed using aggregation functions. Indeed, it has a convenient encryption/decryption mechanism, which helps with use in various cloud-based applications (see Fig. 1).

We are working to adapt Domingo-Ferrer's scheme (Domingo-Ferrer, 2002) to be able to operate within practical applications in cloud-based environments by improving their capabilities to encompass a wider range of arithmetic operations, which leads to increased opportunities to move many of the existing applications to the cloud. In this paper, we highlight and address the following issues:

- An ability to involve a negative number's range in Domingo-Ferrer's scheme (Domingo-Ferrer, 2002). We extend Domingo-Ferrer's scheme to be able to perform arithmetic operations in both positive and negative numbers. We reorganise encryption/decryption parameters in a way that helps to carry out numbers' signs. This contribution allows the deployment of many applications that require both positive and negative ranges in cloud-based environments in a secure manner. In real-life situations, we most often think of negative numbers when we speak of real-world applications, such as military navigation systems, human health monitoring systems and many others. We have to consider how they can be manipulated in a secure and efficient way, especially those applications that involve sensitive and private data. A negative number's range on encrypted data with homomorphism properties can contribute to securing many aggregation functions that use a negative number range as an essential part of their computations.
- According to the previous contribution, we introduce an aggregation function to compute maximum and minimum values among aggregated data based on both positive and negative number ranges. This aggregation function is based on Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme. We improve an idea that is shown in Ertaul and Kedlaya (2007). This is based on Domingo-Ferrer's scheme of finding maximum and minimum values among aggregated values through an ability to combine the two processes to find maximum and minimum in a single process rather than complete them separately. This is a result of taking advantage of both positive and negative number ranges instead of working in a positive numbers range only. This aggregation function is compatible to be applied to the cloud because of its ability to find maximum and minimum values among a set of values in their encrypted form without the need to reveal any information during the implementation of this function.

We revisit previous work and background concepts in Section 2. The proposed extended scheme is described in Section 3. In Section 4, we illustrate arithmetic and logical operations and their specifications based on the proposed scheme. We present implementation details and performance analysis in Section 5. Finally, we conclude the paper in Section 6.

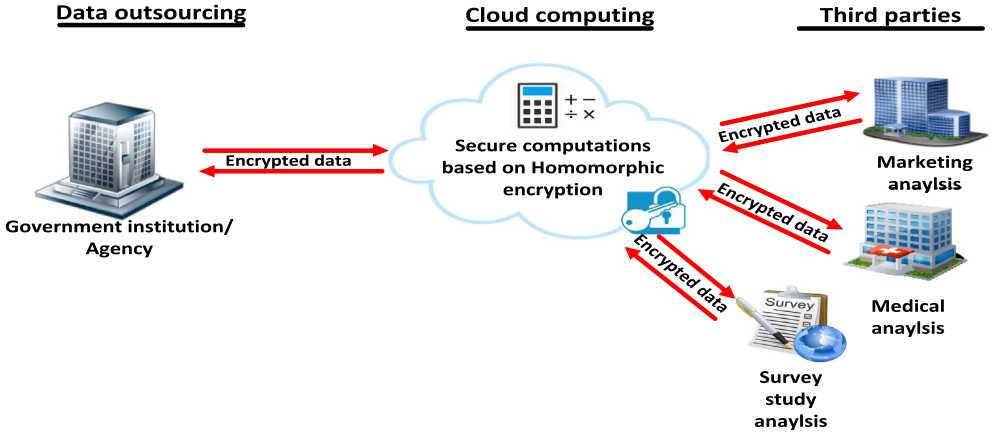

Figure 1 Numerous applications can delegate their data processing to the cloud based on HES.

2. Background

In this section, we provide an overview of secure computation schemes and describe specifications of Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme.

2.1. Secure computation operations

Secure computation allows a single or multiparty to execute specific functions on their inputs and return the result without revealing any private information to other parties. In practicality, several applications must have single or multiparty computations, especially those applications that perform statistical analysis computations shared by distributed multiparties, and the main concern with these applications is preserving data privacy. Secure multiparty computation (SMC) (Prabhakaran and Sahai, 2013), considered an important part of cryptography concepts, is directly involved in these types of applications. Atallah and Blanton (2013) illustrate an overveiw of some techniques for computation outsourcing. However, these techniques have some limitations in terms of computation capabilities that fail to achieve secure analysis services requirements. In general, there are two common techniques for multiparty computations: garbled circuits, such as Yao (1986), and secret sharing, such as Chaum et al. (1988) and Ben-Or et al. (1988). These techniques are involved in many existing multiparty computation protocols. However, we focus on SMCs that are based on HES techniques.

Recently, several FHES schemes, such as Gentry (2009), Van Dijk et al. (2010) and López-Alt et al. (2012), and SHES schemes, such as Damgård et al. (2012), have been introduced as efficient solutions compared to other techniques in the sense of less interaction required between different parties. However, they have either of the following two disadvantages: a lack of efficiency caused by overhead computations in both encryption and decryption processes, which make them impractical for real-world applications, or an inability to perform various functions on both positive and negative numbers that are required as part of many real-world applications, especially in statistical analysis applications.

Upon what has been previously mentioned, SMCs should maintain two main principles: an efficiency in both encryption and decryption processes and an ability to execute various functions as required. We believe that Domingo-Ferrer's scheme (Domingo-Ferrer, 2002) is one of the most efficient schemes to perform at least main basic operations (addition, subtraction and multiplication) based on homomorphism properties. These operations give an opportunity to execute more complex operations based on targeted application requirements.

2.2. Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme

Domingo-Ferrer introduced in Domingo-Ferrer (2002) an HES that has the ability to perform main basic arithmetic operations (addition, subtraction and multiplications) in a secure and convenient manner. Domingo-Ferrer's scheme is classified as a symmetric-key encryption scheme and is proven secure against chosen ciphertext attacks (Wagner, 2003). This scheme does not have the ability to carry numbers' signs meaning that it cannot distinguish between positive and negative numbers' ranges, which is considered an important issue, especially in the case of applying this scheme to applications that use a negative number range as a part of their functionalities. A brief description of Domingo-Ferrer's scheme in (First appendix).

From our observation, we conclude that this scheme can perform operations on encrypted data without considering sign, which limits the scheme's abilities because most real-world applications have to carry a value's sign in their implementation. Accordingly, we attempt to determine a solution for this issue by performing experiments on encryption/decryption parameters, including adjusting these parameters in such a way that it would support carrying a value's sign. This allows for an extension of this scheme to be appropriate for many real-life applications.

3. Proposed secure computations

In the section, we describe our proposed aim to extend Domingo-Ferrer's scheme (Domingo-Ferrer, 2002) to encompass the negative numbers' range through the ability to carry a number's signs.

The essential contribution of this paper relies on restructuring the parameters of Domingo-Ferrer's scheme in such a way that it is collaborative to carry the sign of number. The new configuration assists in performing self or in performing computations in a remote, untrusted third party with the advantage of having both positive and negative number ranges. In this paper, we use similar notations as shown in Domingo-Ferrer (2002), for more explicit and easy observations.

3.1. Keys and parameters

Public parameters m and d are two public parameters where the former is a large integer with many divisors, the latter is a small integer and $d > 2$.

Private parameters m' and r. The former is a small divisor of m and $m' > 1$. m' should be large enough to involve the legitimate computation range to avoid overflow, as received values may fall outside that range. The latter is $r \in Zm$, as long as r^{-1} modulus m exists. We have an additional secret parameter l, which determines the legible range that allows us to carry numbers' signs. We choose l based on the m parameter, as follows:

$$l = \begin{cases} \left[-\left(\left(\frac{m'}{2}\right)-1\right) \quad \frac{m'}{2}\right] & \text{if } m' \text{ is even} \\ \left[-\left(\left(\frac{m'-1}{2}\right)-1\right) \quad \left(\frac{m'-1}{2}\right)\right] & \text{if } m' \text{ is odd} \end{cases}$$

For instance, we can choose $m = 14$, which is considered an even number the legitimate range $l = [-6, 7]$. The parameter l is involved in the encryption process in Eq. (1). This allows us to generate random numbers in a specific range l, which helps to distinguish between negative and positive numbers. Precisely, each received value x is converted to different random small values based on Eq. (1), and the legitimate range l is used to keep the scope of the generated random values in a manner that allows tracking of both positive and negative numbers' ranges. In Fig. 2, we illustrate that our scheme has the ability to carry out numbers' signs or negative numbers precisely, which is not applicable based on Domingo-Ferrer's scheme.

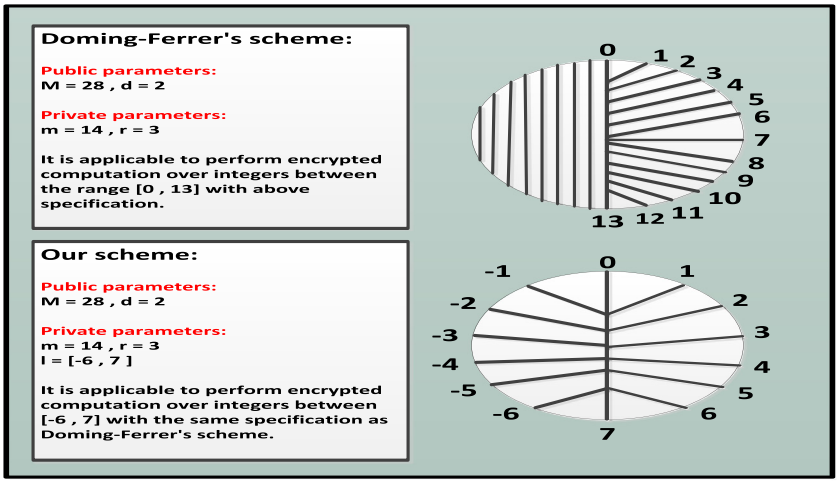

Figure 2 An example of how the scope of our proposed scheme is different from that of Domingo-Ferrer's scheme under a given specification.

3.2. Encryption/decryption processes

We build up encryption/decryption processes in such a way that our target of carrying numbers' signs is achieved. In the following demonstration, we assume that the public parameter $d = 2$.

3.2.1. Encryption process

The encryption process is performed through randomly splitting a selected value $a \in Zm'$ into small secret values $(a_1, a_2 \ldots, a_d)$, where d determines a tuple size for each splitted value as previously mentioned. The splitting process must satisfy the following:

$$a = \sum_{i=1}^{2} a_i \bmod m' \qquad \text{where } a_i \in Zm \tag{1}$$

based on that, we must generate two random values a and b to satisfy Eq. (4) as follows:

1. Choose a random value a_1 from the legitimate range l.
2. Choose a random value a_2 based on three values, which are a_1, a received value a and a secret parameter m, as shown in the following equation:

$$a_2 = m + a - a_1 \tag{2}$$

Eq. (5) is extensible based on the value of the parameter d, which determines the number of random values that are required to be generated. After that, the actual encryption process is performed as follows:

$$E_k(a) = (a_1 \times r^1 \bmod m, a_2 \times r^2 \bmod m) \tag{3}$$

At the end of the encryption process, a tuple of encrypted values is used to represent its plaintext value in encrypted form, and it can be used to perform arithmetic operations with other encrypted tuples. In Algorithm 1, we illustrate the encryption process procedure in more detail.

3.2.2. Decryption process

The decryption process is performed through calculating a scalar product of the, i-th element in a tuple by $r^{-1} \bmod m$ to retrieve $a_i \bmod m$, as follows:

$$(a_1 r^{-1} \bmod m, a_2 r^{-2} \bmod m) \tag{4}$$

Algorithm 1. Encryption process

Require: m, d, m', r, a tuple z and
$a \leftarrow$ a received value
Ensure: $d \leftarrow 2,\ m' > 1,\ r^{-1} \bmod m \text{ exists},\ (0,0) \leftarrow z$
if m' is even **then**
$l[\,] = \left[-\left(\left(\frac{m'}{2}\right) - 1\right) \quad \frac{m'}{2}\right]$
$a_1 = l[i]$ // Pick a random element i of an array $l[\,]$
$a_2 = m + a - a_1$
$z = E_k(a) = (a_1 r \bmod m, a_2 r^2 \bmod) = (z_1,\ z_2)$
else {m' is odd}
$l[\,] = \left[-\left(\left(\frac{m'-1}{2}\right) - 1\right), \quad \left(\frac{m'-1}{2}\right)\right]$
$a_1 = l[i]$ // Pick a random element i of an array $l[\,]$
$a_2 = m + a - a_1$
$z = E_k(a) = (a_1 r \bmod m, a_2 r^2 \bmod) = (z_1,\ z_2)$
//(z_1, z_2) an encrypted result
end if

Then, all elements in a resulted tuple are added through recalling Eq. (4) to retrieve an original value of a, as follows:

$$a = \sum_{i=1}^{2} a_i \bmod m' \tag{1}$$

All arithmetic operations on encrypted data are carried out over $(Zm)^d$ at an unclassified level. Algorithm 2 represents the decryption process, which is essentially the reverse of the encryption process.

Algorithm 2. Decryption process

Require: $m,\ m',\ r^{-1}$, an encrypted tuple a and a dimension of the tuple $a = d$
Ensure: $r^{-1}\ mod\ m\ exists,\ (a_1, \ldots, a_d) \leftarrow a,$
$((0)_1, \ldots, (0)_d) \leftarrow z \quad sum \leftarrow 0,\ result \leftarrow 0$
$z = (a_1 \times r^{-1}\ mod\ m, a_2 \times r^{-2}\ mod\ m, \ldots, a_d \times r^{-d}\ mod\ m)$
$z = (z_1,\ z_2,\ \ldots,\ z_d)$
$sum = \sum_{i-1}^{d} z_i$
$result = sum\ mod\ m'$ // a plaintext result
// The result is mapped based on l range to get the exact result.

We describe a numerical example to illustrate the idea behind our proposed scheme in more detail though concentrating on the computations that show how negative numbers' ranges can be processed based on the proposed scheme. In this example, we choose public and private parameters, as follows:

Public parameters: m is the public modulus and it is chosen to be $m = 28$. We choose $d = 2$.

private parameters: We choose $r = 3$ and $m' = 14$ to be the secret key. We make sure r^{-1} mod m exists, which is $r^{-1} = 19$ in this case. Since we chose $m? = 14$, which is considered an even number, the legitimate range will be $l = [-6, 7]$. The formal interpretation of the legitimate range l in this example is considered as dividing the decrypted results into two ranges of numbers, positive and negative, as follows:

1. If the final decrypted result is in the range between [0,7], the final plaintext result remains the same in the sense that any number in this range with modulus m' has the same value.
2. If the final decrypted result is in the range between [-6 , -1], the final plaintext result with modulus m' is interpreted as follows:

$$(13) \rightarrow (-1), \quad (12) \rightarrow (-2), \quad (11) \rightarrow (-3)$$
$$(10) \rightarrow (-4), \quad (9) \rightarrow (-5), \quad (8) \rightarrow (-6)$$

Fig. 3 shows how a negative number range is deduced based on a legitimate range *l*that is considered a part of the security parameters.

We perform two addition operations combined with a single multiplication operation in the following formula, $(x_1 + x_2)x_3$ where $x_1 = 1,\ x_2 = -4$ and $x_3 = 2$. The first stage is to encrypt all the plaintext values by recalling both Eqs. (1) and (2), respectively, for each single plaintext value.

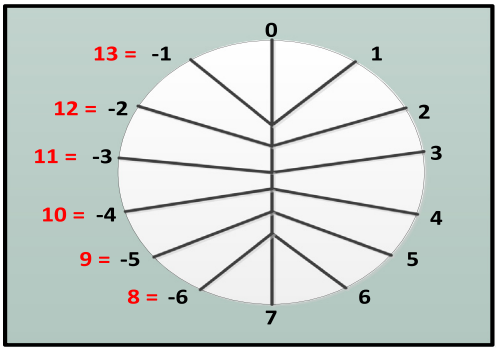

Figure 3 A transformation process for a negative number range based on the example specifications.

Generate random value of $x_1 = 1 \longrightarrow (-6, 21)$
Generate random value of $x_2 = -4 \longrightarrow (4, 6)$
Generate random value of $x_3 = 2 \longrightarrow (3, 13)$

The next step is encrypting all plaintext values by utilising Eq. (2) based on random values that are generated through Eq. (1).

$$E(x_1) = ((-6 \times 3)\ mod\ 28 + (21 \times 9)\ mod\ 28) = (-18, 21)$$
$$E(x_2) = ((4 \times 3)\ mod\ 28 + (6 \times 9)\ mod\ 28) = (12, 26)$$
$$E(x_3) = ((3 \times 3)\ mod\ 28 + (13 \times 9)\ mod\ 28) = (9, 5)$$

All previous steps are processed in a secure environment, which is considered as a classified level. Now, we implement the formula $(x_1 + x_2)x_4$ on encrypted data at an unclassified level, such as a cloud-based environment. We add $(x_1 + x_2)$ to the encrypted forms, as follows:

$$\sum_{i=1}^{2} E(x_i) = (-18 + 12\ mod\ 28, 21 + 26\ mod\ 28) = (-6, 19)$$

Then, we multiply $E(x_3)$ with the addition results $(-6, 19)$,

$$\begin{aligned}(-6,19) \times E(x_4) &= (-6,19) \times (9,5) \\ &= (0, -6 \times 9\ mod\ 28, (-6 \times 5 + 19 \times 9)\ mod\ 28, 19 \times 5\ mod\ 28) \\ &= (0, -26, 1, 11)\end{aligned}$$

Finally, the result is returned to a classified level to perform the decryption process based on Eq. (3) to obtain the plaintext result, as follows:

$$\begin{aligned}&(0 \times r^{-1}\ mod\ m, -26 \times r^{-2}\ mod\ m, 1 \times r^{-3}\ mod\ m, 11 \times r^{-4}\ mod\ m) \\ &= (0 \times 19\ mod\ 28, -26 \times 19^2\ mod\ 28, 1 \times 19^3\ mod\ 28, \\ &\quad 11 \times 19^4\ mod\ 28) \\ &= (0, -6, 27, 15)\end{aligned}$$

The final step in the decryption process is adding all elements in the resulted tuple over Zm' to get $36\ mod\ 14 = 8$. In the same classified level, the final result transformed to the new positive and negative ranges based on the legitimate range l (see Fig. 2). Therefore, the final result is (-6), which is the correct result if we perform the same operations on plaintext values.

We demonstrate this example to state that our proposed scheme has the ability to extend Domingo-Ferrer's scheme through carrying out numbers' signs. Accordingly, we propose an aggregation function to compute maximum and minimum values among aggregated data based on our contribution that relies on both positive and negative numbers' ranges. This function combines the two processes to find maximum and minimum values in a single process rather than completing them separately, as in Ertaul and Kedlaya (2007).

4. Arithmetic and logical operations

In this section, we describe the specifications of arithmetic and logical operations, based on the proposed scheme, through algorithms that show the operational transactions in detail.

We utilise our scheme to build aggregation maximum/minimum function for the aggregated data. This function relies on additive homomorphic property to find both maximum and minimum values among aggregated data. We employed our contribution to reduce the number of operations that are required to perform this function in Ertaul and Kedlaya (2007) by obtaining the benefits of a negative number's range.

4.1. Addition and subtraction operations

As previously illustrated, our scheme has the ability to perform both addition and subtraction operations in such a way that can carry numbers' signs. This involves both positive and negative numbers' ranges. Algorithm 3 represents the general steps to perform a specific encrypted computation for addition and subtraction.

Algorithm 3. Addition and subtraction operations

Require: m, (an addition or subtraction operation $\otimes$),
a and b (encrypted values), a tuple z
Ensure: $(a_1, a_2) \leftarrow a\ (b_1, b_2) \leftarrow b,\ (0,0) \leftarrow z$
$z = (a_1 \otimes b_1\ mod\ m,\ a_2 \otimes b_2\ mod\ m)$
$z = (z_1,\ z_2)$ // $z_n \leftarrow (a_n \otimes b_n)$
// a tuple z is an encrypted operation result

4.2. Multiplication operation

We designed the multiplication operation in our scheme to have the ability to work in a negative numbers' range. In Algorithm 4, encrypted multiplication steps are shown in detail.

Algorithm 4. Multiplication operation

Require: m, (a multiplication operation $\otimes$),
a and b (encrypted values), a tuple z
Ensure: $(a_1, a_2) \leftarrow a\ , (b_1, b_2) \leftarrow b, (0,0,0,0) \leftarrow z$
$z = [0,\ a_1 \times b_1\ mod\ m,\ (a_1 \times b_2 + a_2 \times b_1)\ mod\ m,$
$a_2 \times b_2\ mod\ m]$ // carried as polynomials
$z = [0,\ z_1,\ z_2,\ z_3,\ z_4]$
// a tuple z is an encrypted operation result

The encrypted division operations are carried out in a rational format, such that $E(a)/E(b)$ because the polynomials are not considered as a field but a ring.

4.3. Maximum/minimum function

A proposed maximum/minimum function applies to an existing scheme (Yokoo and Suzuki, 2002) as a part of its operations over aggregated data.We propose necessary modifications to this technique to accommodate negative numbers. Let's say there are two parameters: a weight w and dimension n that are chosen such that $(1 \le w \le n)$ and n is a large enough value to represent the largest value that can be received from outside sources in this scheme. We utilise a new parameter $-n$ to represent a negative dimension of this scheme such that $(-n \le w \le 0)$ and $(0 \le w \le n)$. Each received weight w is classified and encrypted as follows:

$$w = \begin{cases} Case\ 1: \text{The weight } w \text{ is a positive value} \\ Case\ 2: \text{The weight } w \text{ is zero} \\ Case\ 3: \text{The weight} w \text{ is a negative value} \end{cases}$$

Case 1: The weight w is located in a positive numbers' range. Therefore, the encryption process is as follows:

$$E(w) = \underbrace{E(0)}_{0}, \underbrace{E(z), \ldots, E(z)}_{w}, \underbrace{E(0), \ldots, E(0)}_{n-w} \tag{5}$$

where $E(w)$ is an encrypted value of received weight, $E(z)$ is a random encrypted value z that is not equal to 0 and $E(0)$ is the encrypted value of 0. These parameters are uniform in all the mentioned three cases. In this case, $E(0)$ is assigned to all numbers in the negative range from -1 up to $-n$.

Case 2: The weight w is zero. Therefore, the encryption process is as follows:

$$E(w) = \underbrace{E(0), \ldots, E(0)}_{Negative\ range}, \underbrace{E(0)}_{0}, \underbrace{E(0), \ldots, E(0)}_{positive\ range} \tag{6}$$

In this case, $E(0)$ is assigned to numbers in both positive and negative ranges from -1 up to $-n$ and from 1 up to n.

Case 3: The weight w is located in a negative numbers' range. Therefore, the encryption process is as follows:

$$E(w) = \underbrace{E(0), \ldots, E(0)}_{-n-w}, \underbrace{E(z), \ldots, E(z)}_{w}, \underbrace{E(0)}_{0} \tag{7}$$

In this case, $E(0)$ is assigned to all numbers in the positive range from 1 up to n.

Each received weight is transformed into a set of encrypted random non-zero and zero values that are represented in a specific range $(-n, n)$ based on previous mention classification. Domingo-Ferrer's scheme Domingo-Ferrer, 2002 is employed to perform the encryption process for random values $E(z)$ to be used for the transformation process because it has the ability to achieve an additive field operation on encrypted data. A random value z is chosen such that $z \times r < m'$, where r is a number of the expected weights that are passed to the function and $z \neq 0$. The next stage after passing all received weights to question (5)(6)(7) based on their classification is to perform additive field operations to find both maximum and minimum values among all weights. We calculate both maximum and minimum values in a single process based on the following formula:

$$a = \sum_{i=1}^{i=r} (E_{i,j}(x_j), E_{i,j+1}(x_{j+1}), \ldots, E_{i,n}(x_n)) \tag{8}$$

where r is the number of passed weights to the function and $j = -n$. Eq. (8) covers both negative and positive ranges for each transformed weight in a set r. Each encrypted value $E_{i,j}(x_j)$ is either random value $\neq 0$ or 0.

The final stage after performing additive homomorphic encryption is the decryption process in a classified secure level from a positive range $(E_{i,n}(x_n))$ to a negative range $(E_{i,-n}(x_{-n}))$, from right to left, to find a maximum value of a set r and in the reverse mean from the negative range $(E_{i,-n}(x_{-n}))$ to $(E_{i,n}(x_n))$, from left to right is used to find a minimum value of a set r. The decryption process is performed in both ranges in parallel mean, and it continues until non-zero values are detected during this process. Each encrypted value

has a corresponding value j that represents an element order in a set r. The given parameter j is utilised to determine a maximum or minimum value based on the first non-zero values that are detected during the decryption process of both negative and positive ranges.

We accentuate the main steps to perform maximum/minimum function in Algorithms 5–7. This function consists of two main stages: generating random encrypted values which occur in a classified secure level, and encrypting computations, which occur in an unclassified public level.

We assume that random encrypted values are generated based on algorithms in the previous section. Algorithm 5 shows the distribution process of random encrypted data based on a range of received data.

Algorithm 5. Distribution of random encrypted values process

Require: a(a received value) , $[-n, n]$ (legitimate range), $R[\,]$(random encrypted values) , $t[\,]$
Ensure: $t[\,] \leftarrow \sum_{i=-n}^{n} t[x_i] = 0$
  if $a > 0$ **then**
    $\sum_{i=1}^{a} t[x_i] = R[i]$ // i is an element in $R[\,]$
    $\sum_{i=a+1}^{n} t[x_i] = 0$// the rest of positive range set to zero
    $\sum_{i=-n}^{0} t[x_i] = 0$ // all negative range set to zero
  else if $a < 0$ **then**
    $\sum_{i=-1}^{a} t[x_i] = R[i]$ // i is an element in $R[\,]$
    $\sum_{i=a-1}^{-n} t[x_i] = 0$// the rest of negative range set to zero
    $\sum_{i=0}^{n} t[x_i] = 0$ // All positive range set to zero
  else {a = 0}
    $t[0] = 0$// i is an element in $R[\,]$
    $\sum_{i=1}^{n} t[x_i] = 0$// the positive range set to zero
    $\sum_{i=-n}^{-1} t[x_i] = 0$ // All negative range set to zero
  end if

In Algorithms 6 and 7, additive field operations are performed for all arrays where we assume 5 arrays in this case for simplicity. They keep random encrypted and zero values and obtain a new array after performing additive field operations. The resulting array is decrypted in a classified secure level to obtain maximum and minimum values.

Algorithm 6. Find maximum value

Require: $T_{i,5}[\,]$ (a set of arrays), $[1, n]$ (legitimate range) $Result[\,]$, x, $count_1$, $count_2$, Max, Min
Ensure: $\sum_{i=1}^{n} Result[x_i] = 0, x \leftarrow 0, count_1 \leftarrow n, Max \leftarrow 0$
  for j := 1 to n step 1 **do**
    for i = 1 to 5 step 1 **do**
      $Result_j[] \leftarrow Result_j[] + T_{i,j}[]$
    end for
  end for
  for j := n to 1 step -1 **do**
    $x =\leftarrow decrypt(Result[j])$
    if x = 0 **then**
      $count_1 = count_1 - 1$
    else {}
      $break$;
    end if
  end for
  $Max \leftarrow count_1$

Algorithm 7. Find minimum value

Require: $T_{i,5}[\,]$ (a set of arrays), $[-n, 0]$ (legitimate range), $Result[\,]$, x, $count_2$, Min
Ensure: $\sum_{i=-n}^{0} Result[x_i] = 0, x \leftarrow 0, count_2 \leftarrow -n, Min \leftarrow 0$
  for j := -n to 0 step 1 **do**
    for i = 1 to 5 step 1 **do**
      $Result_j[] \leftarrow Result_j[] + T_{i,j}[]$
    end for
  end for
  for j := -n to 1 step 1 **do**
    $x =\leftarrow decrypt(Result[j])$
    if x = 0 **then**
      $count_2 = count_2 + 1$
    else {}
      $break$;
    end if
  end for
  $Min \leftarrow count_1$

We describe a numerical example to disclose the idea behind maximum/minimum function in more detail. We apply the same parameters of the previous example in encryption/decryption processes, and we assume the expected range $(-n, n)$ is $(-6, 6)$ and $r = 4$ such that we have $(r_1, r_2, r_3, r_4) = (3, -2, 1, 0)$, based on each weight r_i being processed, as follows:

$r_1 = 3$ is considered in a positive numbers' range. We transform $r1$ by calling Eq. (5):

$$E(3) = \underbrace{E_{1,0}(0)}_{0}, \underbrace{E_{1,1}(z_1), E_{1,2}(z_2), E_{1,3}(z_3)}_{3}, \underbrace{E_{1,4}(0), \ldots, E(0)_{1,6}}_{6-3}$$

All elements in the negative numbers' range $(-6, -1)$ are set to encrypted values of zero $(E_{1,j}(0))$.

$r_2 = -2$ is considered to be in a negative numbers' range. We transform $r2$ by calling Eq. (7):

$$E(-2) = \underbrace{E_{2,-6}(0), \ldots, E(0)_{2,-3}}_{-6-(-2)} \underbrace{E_{2,-2}(z_{-2}), E_{2,-1}(z_{-1})}_{-2}, \underbrace{E_{2,0}(0)}_{0}$$

All elements in the positive numbers' range $(1, 6)$ are set to encrypted values of zero $(E_{2,j}(0))$.

$r_3 = 1$ is considered to be in a positive numbers' range. We transform $r3$ by calling Eq. (5):

$$E(1) = \underbrace{E_{3,0}(0)}_{0}, \underbrace{E_{3,1}(z_1)}_{1}, \underbrace{E_{3,2}(0), E_{3,3}(0), \ldots, E(0)_{3,6}}_{6-1}$$

All elements in the negative numbers' range $(-6, -1)$ are set to encrypted values of zero $(E_{3,j}(0))$.

$r_4 = 0$ is considered to be zero. We transform $r4$ by calling Eq. (6):

$$E(0) = \underbrace{E_{4,-6}(0), \ldots, E_{4,-1}(0)}_{Negative\ range}, \underbrace{E_{4,0}(0)}_{0}, \underbrace{E_{4,1}(0), \ldots, E_{4,6}(0)}_{positive\ range}$$

All elements in both the positive and negative numbers' ranges $(1, 6)$ and $(-6, -1)$ are set to encrypted values of zero $(E_{4,j}(0))$.

We generate random values $(z_1, z_2, z_3, z_4, z_5)$ to encrypt them under Domingo-Ferrer's scheme with the same previous specifications. The random values are used for the transformation

process as part of the maximum/minimum function stages. We assume the values of $(z_1, z_2, z_3, z_4, z_5)$ are $(3, 1, 2, 0, 0)$, respectively. The encryption process for these random values is performed as follows:

$$E(z_1) = E(3) = E(-1, 18) = E(-3, 22)$$
$$E(z_2) = E(1) = E(5, 10) = E(15, 6)$$
$$E(z_3) = E(2) = E(6, 10) = E(18, 6)$$
$$E(z_4) = E(0) = E(-6, 20) = E(-18, 12)$$
$$E(z_5) = E(0) = E(1, 13) = E(3, 5)$$

We can encrypt extra zero values z_i based on Domingo-Ferrer's scheme to make both transformation and additive homomorphic operations more random and secure. Next, these encrypted values are utilised to fill non-zero values $E(z)$ randomly during the transformation process. The first value $r_1 = 3$ is transformed as follows:

$$S_1 = E(3) = \underbrace{E(0)}_{0}, \underbrace{E(2), E(1), E(3)}_{3}, \underbrace{E(0), E(0), E(0)}_{6-3}$$

$$S_1 = \underbrace{E(0)}_{0}, \underbrace{E(18, 6), E(15, 6), E(-3, 22)}_{3}, \underbrace{E(0), E(0), E(0)}_{6-3}$$

Each $E(0)$ is filled by either z_4 or z_5 randomly, and all elements in the negative numbers' range $(-6, -1)$ are set to encrypted values of zero $(E(0))$. The next received value is $r_2 = -2$,

$$S_2 = E(-2) = \underbrace{E(0), E(0), E(0), E(0)}_{-6-(-2)}, \underbrace{E(1), E(3)}_{-2}, \underbrace{E(0)}_{0}$$

$$S_2 = \underbrace{E(0), E(0), E(0), E(0)}_{-6-(-2)}, \underbrace{E(15, 6), E(-3, 22)}_{-2}, \underbrace{E(0)}_{0}$$

Each $E(0)$ is randomly filled by either z_4 or z_5 and all elements in the positive numbers' range $(1, 6)$ are set to encrypted values of zero $(E(0))$. The next received value is $r_3 = 1$,

$$S_3 = E(1) = \underbrace{E(0)}_{0}, \underbrace{E(2)}_{1}, \underbrace{E(0), E(0), E(0), E(0), E(0)}_{6-1}$$

$$S_3 = \underbrace{E(0)}_{0}, \underbrace{E(18, 6), E(15, 6), E(-3, 22)}_{3}, \underbrace{E(0), E(0), E(0)}_{6-3}$$

Each $E(0)$ is randomly filled by either z_4 or z_5 and all elements in the negative numbers' range $(-6, -1)$ are set to encrypted values of zero $(E(0))$. The last received value is $r_4 = 0$,

$$S_4 = E(0) = \underbrace{E(0), E(0), E(0), E(0), E(0), E(0)}_{Negative\ range(-6-0)}, \underbrace{E(0)}_{0}, \underbrace{E(0), E(0), E(0), E(0), E(0), E(0)}_{positive\ range(6-0)}$$

Each $E(0)$ is randomly filled by either z_4 or z_5. The next stage is performing additive field operations over (S_1, S_2, S_3, S_4). After performing the additive operation, we have a set of tuples in both the positive and negative ranges, shown as follows:

First, the negative range of the additive operation result is:

$$Result_1 = \underbrace{E(0), E(0), E(0), E(0)}_{negative\ range}, \underbrace{E(N_{-2}), E(N_{-1})}_{negativerange}, \underbrace{E(0)}_{0}$$

All added tuples' results in the negative range are zero values, except N_{-2} and N_{-1} because they are already filled with non-zero values in S_1 and added to zero values in S_2, S_3 and S_4.

Second, the positive range of the additive operation is:

$$Result_2 = \underbrace{E(0)}_{0}, \underbrace{E(N_1), E(N_2), E(N_3)}_{positive\ range}, \underbrace{E(0), E(0), E(0)}_{positive\ range}$$

All added tuples in the positive range are zero values except N_1, N_2, N_3 because they are already filled with non-zero values in S_1 and S_3, and added to zero values in S_2 and S_4.

The final stage is performed in a classified secure level, which includes the decryption process from both directions from left to right in $Result_1$, which is considered a negative range decryption and from right to left in $Result_2$, which is considered a positive range decryption. In $Result_1$, all tuples are decrypted from $j = -6$ up to $j = -3$ to get zero values. At $j = -2$, the decrypted result is a non-zero value, therefore it is considered as a minimum value of all received values. On the other hand, in $Result_2$, all tuples are decrypted from $j = 6$ up to $j = 4$ to get zero values. At $j = 3$, the decrypted result is a non-zero value, so this is considered a maximum value of all received values. By comparing our proposed function with an existing function Ertaul and Kedlaya, 2007, we can conclude that our function reduces the redundancy that happens when the function Ertaul and Kedlaya, 2007 performs separate operations to find maximum and minimum values through combining these in a single operation by using the negative numbers' range as a part of finding maximum and minimum values rather than working in just a positive numbers' range.

5. Implementation and performance analysis

In this section, we describe implementations of the proposed scheme and its operations. Our code runs on an Intel Core i5 Processor 2.40 GHz and 4 GB RAM. Java programming language (Gosling et al., 2005) is used as a convenient framework. Our implementation has the following three main processes: (1) an encryption process, which is involved in generating random values and transformation to tuples followed by an actual encryption process; (2) performing encrypted operations that mainly include addition, subtraction and multiplication and (3) a decryption process which is the final stage after finishing the computation process stage.

In this paper, we implemented a performance analysis by identifying the amount of time consumption for both encryption and decryption processes and operational calculations based on the scenario of having self-computation process applications that would be adapted in cloud-based environments. Table 1 illustrates execution time for the different numerical operations of self-computation processes, and all results are shown in millisecond unit.

Table 1 Execution time of self-computation processes.

Process type	Encryption	Addition	Subtraction	Multiplication	Decryption
Execution time	9.195×10^{-2}	7.22×10^{-2}	9.66×10^{-2}	7.27×10^{-2}	6.97×10^{-2}

Table 2 Execution time comparison between Gentry and proposed protocol based on Domingo-Ferrer schemes (unit: milliseconds).

Cryptosystem	Encryption	Decryption	Addition	Multiplication	Size of data
Gentry's scheme	8.39×10^{-3}	7.64×10^{-4}	4.97×10^{-5}	7.72×10^{-4}	Per bit
Proposed protocol based on (Domingo-Ferrer's scheme)	9.195×10^{-2}	6.97×10^{-2}	7.22×10^{-2}	7.27×10^{-2}	Per value (integer or float-point number)

It is well-known that there is a trade-off between security and flexibility in real-world systems. In our protocol, we adapt a practical homomorphic scheme with other secure mechanisms that ensure the privacy of consumers' data and performance elasticity of provided services. Table 2 illustrates a performance of execution time for different operations of proposed protocol based on Domingo-Ferrer's scheme compared with a well-known homomorphic encryption called Gentry's scheme Gentry, 2009.

6. Conclusion

In this paper, we proposed some techniques to perform various arithmetic and comparison operations to ensure secure computation in several applications running in cloud environments using Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme. This led to the advantage of moving applications that require secure computations on both positive and negative ranges in cloud computing. Moreover, this contribution can be applicable to extend another Domingo-Ferrer's scheme named a new privacy homomorphism and application Ferrer, 1996. According to that, we designed an aggregation maximum/minimum function based on Domingo-Ferrer's scheme that improves the existing scheme (Ertaul and Kedlaya, 2007) by extending this function to work in the negative number range and improving the efficiency by combining the processes of finding maximum and minimum values in a single process. The performance of the proposed scheme is quite satisfactory and efficient enough to use in lightweight applications, and it is convenient to be applied to cloud-based applications.

Appendix A. Domingo-Ferrer's additive and multiplicative privacy homomorphism scheme

In Domingo-Ferrer's scheme, m and d are two public parameters where the former is a large integer with many divisors and the latter is a small integer and $d > 2$. Since each single encrypted value is represented by a tuple, which is a set of elements, the parameter d represents the number of elements in each tuple. The secret parameters are m' and r. The former is a small divisor of $m, m' > 1$ and the latter is $r \in Zm$, as long as r^{-1} modulus m exists. In short, we can consider the secret key in this scheme as $k = (m', r)$. All encryption and decryption processes are performed in a classified level that is considered a secure part of this scheme; meanwhile, all arithmetic operations on encrypted data are performed in an unclassified level, which is considered a public environment, such as a cloud-based environment.

The encryption process is performed through randomly splitting the selected value $a \in Zm'$ into small secret values $(a_1, a_2 \ldots, a_d)$, where d determines a tuple size for each splitted value, as previously mentioned. The splitting process must satisfy the following:

$$a = \sum_{i=1}^{d} a_i \bmod m' \qquad \text{where } a_i \in Zm \tag{.1}$$

After that,

$$E_k(a) = (a_1 r^1 \bmod m, a_2 r^2 \bmod m, \ldots, a_d r^d \bmod m) \tag{.2}$$

At the end of an encryption process, a tuple of encrypted values is used to represent its plaintext value in encrypted form, and it can be used to perform arithmetic operations with other encrypted tuples.

The decryption process is performed through calculating a scalar product of the, $i - th$ element in a tuple by $r^{-1} \bmod m$ to retrieve $a_i \bmod m$ as follows:

$$(a_1 r^{-1} \bmod m, a_2 r^{-2} \bmod m, \ldots, a_d r^{-d} \bmod m) \tag{.3}$$

Then, all elements in a resulted tuple are added through recalling Eq. (1). to retrieve an original value of a as follows:

$$a = \sum_{i=1}^{d} a_i \bmod m' \tag{.1}$$

All arithmetic operations on encrypted data are carried out over $(Zm)^d$ at an unclassified level. We demonstrate a simple numerical example of this scheme below to clarify both encryption/decryption processes and arithmetic computations on encrypted data, as well. In this example, we choose public and private parameters, as follows:

Public parameters: m is the public modulus and it is chosen to be $m = 28$. We choose $d = 2$, which represents a number of elements in a single tuple. For simplicity, it is recommended to be $d > 2$ in worst case for security reasons.

Private parameters: We choose $r = 3$ and $m' = 7$ to be the secret key. We make sure r^{-1} mod m exists, which is $r^{-1} = 19$ in this case.

We use a formula to perform two addition operations combined with a single multiplication operation, namely $(x_1 + x_2)x_3$ where $x_1 = 2, x_2 = 1$ and $x_3 = 0$. The first stage

is to encrypt all these plaintext values by recalling both Eqs. (1) and (2) respectively, for each a single plaintext value.

Generate random value of $x_1 = 2 \longrightarrow (0, 9)$
Generate random value of $x_2 = 1 \longrightarrow (1, 7)$
Generate random value of $x_3 = 0 \longrightarrow (2, 5)$

The next step is encrypting all plaintext values by utilising Eq. (2) based on random values that are generated through Eq. (1).

$$E(x_1) = ((0 \times 3)\ mod\ 28,\ (9 \times 9)\ mod\ 28) = (0, 25)$$
$$E(x_2) = ((1 \times 3)\ mod\ 28,\ (7 \times 9)\ mod\ 28) = (3, 7)$$
$$E(x_3) = ((2 \times 3)\ mod\ 28,\ (5 \times 9)\ mod\ 28) = (6, 17)$$

All previous steps are processed in a secure environment, which is considered a classified level. Now, we implement the formula $(x_1 + x_2)x_4$ on encrypted data at an unclassified level, such as in cloud-based environment. We add $(x_1 + x_2)$ in their encrypted forms, as follows:

$$\sum_{i=1}^{2} E(x_i) = (0 + 3\ mod\ 28, 25 + 7\ mod\ 28) = (3, 4)$$

Then, we multiply $E(x_3)$ with the addition result $(3, 4)$,

$$\begin{aligned}(3,4) \times E(x_4) &= (3,4) \times (6,17) \\ &= (0, 3 \times 6\ mod\ 28, (3 \times 17 + 4 \times 6)\ mod\ 28, 4 \times 17\ mod\ 28) \\ &= (0, 18, 19, 12)\end{aligned}$$

Finally, the result is returned to a classified level to perform the decryption process, based on Eq. (3), and obtain the plaintext result, as follows:

$$\begin{aligned}&(0 \times r^{-1}\ mod\, m, 18 \times r^{-2}\ mod\, m, 19 \times r^{-3}\ mod\, m, 12 \times r^{-4}\ mod\, m) \\ &(0 \times 19\ mod\, 28, 18 \times 19^2\ mod\, 28, 19 \times 19^3\ mod\, 28, \\ &12 \times 19^4\ mod\, 28) = (0, 2, 9, 24)\end{aligned}$$

The final step in the decryption process is adding all elements in the resulted tuple over Zm' to get $35\ mod\ 7 = 0$, which is the correct result if we do the same operations on plaintext values.

References

M. Atallah, M. Blanton, Techniques for Secure and Reliable Computational Outsourcing, Technical Report, DTIC Document, 2013.

Bayer-Fluckiger, E., 2002. Panorama Number Theory, 168–184.

Benaloh, J., 1994. In: Proceedings of the Workshop on Selected Areas of Cryptography, pp. 120–128.

Ben-Or, M., Goldwasser, S., Wigderson, A., 1988. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. ACM, pp. 1–10.

Chaum, D., Crépeau, C., Damgard, I., 1988. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. ACM, pp. 11–19.

Damgård, I., Pastro, V., Smart, N., Zakarias, S., 2012. In: Advances in Cryptology–CRYPTO 2012. Springer, pp. 643–662.

Domingo-Ferrer, J., 2002. In: Information Security. Springer, pp. 471–483.

ElGamal, T., 1985. In: Advances in Cryptology. Springer, pp. 10–18.

Ertaul, L., Kedlaya, V., 2007. In: ICWN, pp. 186–192.

Ferrer, J.D.I., 1996. Information Processing Letters 60, 277–282.

Gentry, C., 2009. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC '09. ACM, New York, NY, USA, pp. 169–178.

Gentry, C., 2009. A fully homomorphic encryption scheme, Ph.D. thesis, Stanford University.

Gentry, C., 2010. Commun. ACM 53, 97–105.

Gosling, J., Joy, B., Steele, G., Bracha, G., 2005. Java (TM) Language Specification. The (Java (Addison-Wesley)), Addison-Wesley Professional.

López-Alt, A., Tromer, E., Vaikuntanathan, V., 2012. In: Proceedings of the 44th Symposium on Theory of Computing. ACM, pp. 1219–1234.

Paillier, P., 1999. In: Advances in Cryptology–EUROCRYPT99. Springer, pp. 223–238.

Prabhakaran, M.M., Sahai, A., 2013. Secure Multi-party Computation. IOS Press.

Rivest, R.L., Shamir, A., Adleman, L., 1978. Commun. ACM 21, 120–126.

Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V., 2010. In: Advances in Cryptology–EUROCRYPT. Springer, pp. 24–43.

Wagner, D., 2003. In: Information Security. Springer, pp. 234–239.

Yao, A.C.-C. In: Foundations of Computer Science, 27th Annual Symposium on, IEEE, 1986, pp. 162–167.

Yokoo, M., Suzuki, K., 2002. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1. ACM, pp. 112–119.