Privacy homomorphisms

RÉKA LIMBEK, PÉTER SZIKLAI

Eötvös Loránd University, Faculty of Natural Sciences,
Department of Information Systems,Department of Computer Science
lreka@elte.hu, sziklai@cs.elte.hu

Reviewed

Keywords: data security, calulation- and data-delegation, coding

Cryptographic homomorphisms are used to charge a company with performing calculations where both input and output data are confident and therefore can be communicated in cryptographic ways only. This requires the use of special cryptography so that operations can be performed on coded data and decoding the results one gets the same result as if the calculation were made with uncoded data. This is the main point of the article which also introduces basic methods and possible attacks.

Introduction

Privacy homomorphisms were introduced by Rivest, Adleman and Dertouzos [6] in 1978 to solve the *computing delegation* problem in the first instance. This typically occurs when the data owner has only limited computing facilities, i.e. either the computation to be performed is too complex or an unmanageable bulk of data must be processed. In this case, the owner is constrained to deliver his data to a computer centre (data manager) that is able to perform the necessary computations. If the data are sensitive, i.e. of a confidential nature, there is the obvious problem that they could get into an environment which is not necessarily reliable.

This situation emerges in relation of a great number of Internet-based applications, typically when we are using a remote service. Even a modest software for currency conversion or for route planning can fall into this category, but the classical examples include programs for portfolio management and income tax calculation, running not on our computer but on a server operated by third parties. The problem of computing delegation may also occur in the field of academic research, where for example a medical research team uses a(n insecure) university mainframe for processing confidential healthcare records.

The question of *data delegation* represents a similar problem. The two kinds of delegation differ from each other in the fact that in the case of data delegation the results are relevant not for the data owner but for the data manager. As for the owner, the complete computing process is indifferent, he often does not possess the appropriate facilities for performing the computation, and "lends" his data only for the duration of the computing process. The data manager, for his part, wants to own not the raw data, but the processed data (for example the results of statistical analysis). As far as sensitive data is concerned, there is again the problem that this data will get out for the duration of the "lending" into a not necessarily reliable environment.

Data delegation typically occurs in the case of organisations having a federal-like structure. The state administration has generally a structure like this, but equally adequate examples are the European Union or the United States of America. Each member state has its own data (budget, registration of the population, etc.), which a central organisation would like to use for analysing purposes. In return for supplying the data the member states can require the opportunity to analyse the collected data. Because of data protection considerations, however, they have no possibility to store data of other member states, and have neither the computing capacity to perform the necessary operations. In turn, the central organisation, which is not facing any of these problems, has no excess capacity to perform for each member state the computing task they ordered. How and in which form could the collected data be transferred to the member states so that all these problems could be solved?

The secure solution of data delegation could also expand the application of smart cards. In this way, the need that a resource-demanding application must run on the data stored by the card would not present any problem. The card would simply export the data required for the computation to the unit running the application.

In both cases of delegation, there is a double challenge we must meet. On the one hand, we have to prevent the unauthorized access during the data transfer, and on the other, appropriate measures must be taken in order to impede that, in the case of computing delegation, the computing software, the computer and its personnel and, in the case of data delegation, the members can get (unauthorized) access to valuable information as regards the raw data, based on the data made available for them. The deep exploration of the former problem began as early as before the Internet had been going to spread out, especially due to the military and state security applications. Once the world wide net became the forum of more and more applications, the same question has been set in some other way, but the introduction and the convenient imple-

mentation of public key encryption have largely solved the data security problem.

The aim of this contribution is to investigate the second data security problem we mentioned above. Hardware and/or database management solutions are available in both cases, such as the use of physically protected processors or data fragments [8]. The cryptographic solution we are interested in, however, is the application of *privacy homomorphisms* that provide the possibility to transfer the data in an encrypted form to the unreliable level (computer centre, member states) and to perform there the relevant operations *without data decryption*, in their encrypted form. After the results have been sent back in an encrypted form to the reliable level (limited-capacity user, central organisation), we will get, after decryption, *the same result* as in the case where we would have performed the operations in the original form.

Higher security can be achieved by combining the different solutions, but the interpretation of these possibilities lies beyond the boundary of this contribution.

Terminology

Data appearing in a form that is interpretable for everyone is called *plaintext*, irrespective of the data type. The enciphering of the plaintext is called *coding* or *encryption*, the result of which is the *ciphertext* or *encrypted* text. If the legitimate user decrypts the ciphertext, we are speaking of *deciphering* of the encrypted text. The result of this is the original plaintext. If an attacker tries to decrypt the encrypted text, we have to do with *breaking* of the text. The decoding process is a sort of inversion of the coding process.

The encryption is performed by an encryption algorithm, which possesses a parameter, i.e. the *key*. Basically, the deciphering algorithm possesses this key parameter as well. In the case where both keys are identical, the process is called *symmetric* or *secret key encryption*. If, however, the two keys are different, it is the case of *asymmetric* or *public key encryption*, where one of the keys is generally public, i.e. accessible for everyone.

The attacker intending to break either the code or the encrypting algorithm seeks principally the way to break the encrypted text but occasionally also to define the secret key used for the coding. As the security of an encryption algorithm, i.e. in which measure it is resistant to the attacks, shall never depend on whether the algorithm itself is known for the public or not, we always suppose that the attacker knows the encryption method. For the attacker it is important to define the plaintext and the key used.

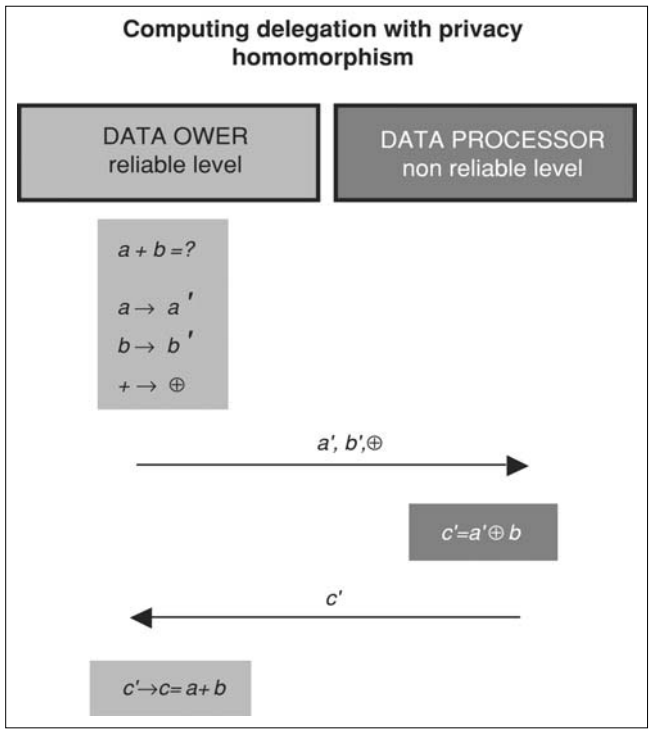
Privacy homomorphisms

Privacy homomorphisms can be used for processing encrypted data. These homomorphisms actually perform the encryption of the data so that the operations can be executed on the encrypted data in the same way as otherwise.

A privacy homomorphism represents an operation-preserving mapping from the set of plaintexts to the set of encrypted texts. Formally, if S is the set of plaintexts and S' is the set of encrypted texts, then an E_K: $S \rightarrow S'$ homomorphism can be defined where K is the key used as the parameter of the function.

Let the operations and predicates interpreted on S and S' be $f_1, f_2, \ldots, f_k$ and $p_1, p_2, \ldots, p_l$ and $f'_1, f'_2, \ldots, f'_k$ and $p'_1, p'_2, \ldots, p'_l$, respectively. Each operation or predicate within S corresponds with one interpreted on S', which generally differs from the original (as we select the operands from other sets), but it also can be very similar. The operation-preservation of E_K means formally that for

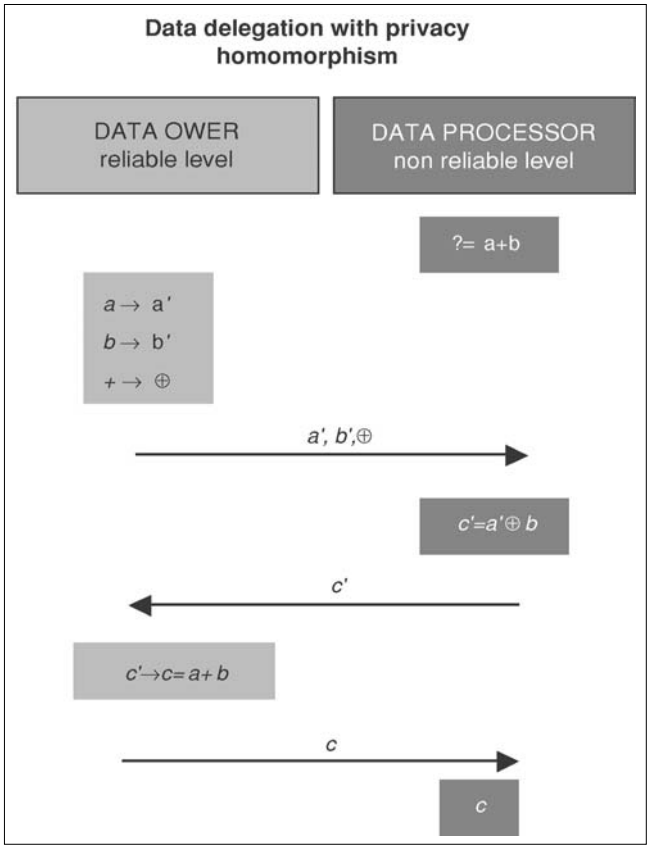
any $a,b,\ldots \in S$ and any $i = 1,\ldots,\mathbf{k}$
$E_K(f_i(a,b,\ldots)) = f'_i(E_K(a), E_K(b),\ldots)$,
as well as for any $j = 1,\ldots,l$
$p_j(a,b,\ldots) = p'_j(E_K(a), E_K(b),\ldots)$.



The deciphering (decryption) of the encrypted text is performed by a function D_K: $S' \rightarrow S$, with S' being the value set of the homomorphism E_K and K' the key. The function D corresponds to the inverse of E, so that it is also an operation-preserving mapping. This feature ensures that performing the operations on the encrypted text, then deciphering it yields the expected result, i.e. for any $a,b,\ldots \in S$ and i $f_i(a,b,\ldots) = D_K(f'_i(E_K(a), E_K(b),\ldots))$.

How does such a homomorphism work?

For illustration consider an RSA-related privacy homomorphism. As a key we select two large primes $K=(p, q)$, and denote the product of them by m. The set of plaintexts is $Z_m = \{0,1,2,\ldots, m\text{-}1\}$, the modulo m residual classes, with the conventional modulo m operators $+, -,$?. The encrypted correspondent of a plaintext $a \in Z_m$ is



formed by a number pair, the components of which are the remainders of the division of a with p and q, respectively. Formally it can be expressed by: $E_{(p,q)}(a) = (a \bmod p, a \bmod q)$. On encrypted texts as well, the modulo m +,-,? will be performed, in this case on every component. From a pair of numbers, (a mod p, a mod q), we obtain the original a by means of the Chinese remainder theorem, that is the decryption process is the application of the Chinese remainder theorem.

Chinese remainder theorem

If $m_1, m_2, \ldots, m_k$ are pairwise coprimes then the system of congruences

$$x \equiv a_1 \ (m_1)$$
$$x \equiv a_2 \ (m_2)$$
$$\ldots$$
$$x \equiv a_k \ (m_k)$$

has a unique solution mod $m_1 \_ m_2 \_ \ldots \_ m_k$.

Example:

$$x \equiv 1 \ (3)$$
$$x \equiv 3 \ (5)$$
$$x \equiv -1 \ (7)$$

We search a solution of form $x = 35y_1 + 21y_2 + 15y_3$. Then from the system of congruences

$$35y_1 \equiv 1 \ (3)$$
$$21y_2 \equiv 3 \ (5)$$
$$15y3 \equiv -1 \ (7)$$

A solution of it is:
$y_1 = -1$, $y_2 = 3$, $y_3 = -1$, from which $x = 13$ follows.

We will show the function of the homomorphism through a numeric example, and as we are interested in the demonstration, we had to decide on desperately small parameters. The secret key will be (3, 11), $m = 33$. Suppose that the formula to be computed is (12+4)×2 (with modulo 33 operations), thus the three plaintexts are 12, 4, 2, and the expected result is 32 mod 33. We send the encrypted plaintext in the following form: $E_{(3,11)}(12) = (0, 1)$, $E_{(3,11)}(4) = (1, 4)$, $E_{(3,11)}(2) = (2, 2)$. Of course, we also communicate to the entity performing the computation that we would like to multiply the sum of the first two members with the third factor. Accordingly, they compute the value of ((0, 1)+(1, 4))×(2, 2), where these operations are modulo 33 operations taken by every component. The computation goes like this: ((0, 1)+(1, 4))×(2, 2) = (0+1, 1+4)×(2, 2) = (1, 5)×(2, 2) =(2, 10). We obtain the result in the encrypted form: (2, 10), which we decrypt by means of the Chinese remainder theorem. We thus have to solve the congruence system $x \equiv 2 \pmod 3$, $x \equiv 10 \pmod{11}$. We are searching for the solution in the form $x = 3y + 11z$. As $3y$ is divisible by 3, the congruence $11z \equiv 2 \pmod 3$ must be fulfilled. For the same reason, $3y \equiv 10 \pmod{11}$. Hence $z = 1$ and $y = 7$, respectively, that is $x = 3 \times 7 + 11 = 21 + 11 = 32 \pmod{33}$.

Therefore, the privacy homomorphism becomes specifically useful for computing and data delegation problems because the “encrypted operations” corresponding to the “plain operations” can be performed on the encrypted text corresponding to the plaintext, and we obtain the appropriate “plain result” by deciphering the encrypted result thus achieved.

Why is a privacy homomorphism good?

First of all, a privacy homomorphism must be appropriate for the given application, that is it must preserve those operations, which are necessary from the point of view of the application. In addition, we can raise requirements basically on the *efficiency* and the *security* of the privacy homomorphism. These requirements include the ease of computation of the function itself, and the same must be valid also for the deciphering. A further requirement on the efficiency is that the computation of the f'_i-s and p'_i-s corresponding to the operations and predicates, respectively, of the plaintext must be performed quickly, and that the encumbrance of the encrypted text must not be much more than that of the plaintext.

From the cryptographic perspective, the question of security is more exciting. For this examination, we must overview the (passive) attacks that can be carried out against a cryptosystem.

Attacks

As far as privacy homomorphisms are concerned, we can mention essentially three kinds of attacks. All three are passive in the sense that the attacker tries not to

modify data or to make the computations physically impossible, but intends to get valuable information through the possession of certain data with respect to the plaintext, the encrypting algorithm or the key.

As we mentioned earlier, the security of the privacy homomorphism, but also that of any encryption system shall not depend on whether the attacker knows the encryption algorithm or not. The key used for the encryption is of course secret, but for the algorithm (or mapping) we always suppose that it is publicly accessible (though we do not explicitly intend to provide this access).

In the simplest case, the attacker has the least standing-ground to rely on, so he is facing the most complex task. Texts encrypted with the same key K are available for him; hence this attack is called *ciphertext only* attack. The attacker seeks then to abduct as much information as possible from this "knowledge base" with respect to the plaintext and the key used.

In the case of *known plaintext* attacks the attacker tries to get information related to the key K (and to the encrypting algorithm) based on plaintext-ciphertext pairs $(a, E_K(a))$.

The attacker possesses the most information in the *chosen plaintext* or *ciphertext* attacks. In this case, the attacker has the option to decide on the plaintext or ciphertext, and ask for the corresponding ciphertext or plaintext pair, and accordingly tries to find out the key and the encrypting algorithm.

The complexity of the attacks decreases in the above sequence, as the starting information available for the attacker is increasing. We can also say that the potency of the attacks is increasing in the sequence, because the attacker becomes stronger with a steadily growing arsenal. As regards, however, the preparation for the attacks, an inverse classification can be of use, as the attacker only has to tap the communication channel to execute the ciphertext only attack, while the chosen text attack needs obviously the addition of some other practices.

We define the safety of a privacy homomorphism in a "natural" manner: a homomorphism is safer if it resists stronger attacks. Generally speaking, the higher a privacy homomophisms's security level is, the more useful it is. But the necessary security level is greatly depending on the given application. The data delegation requires a higher security degree than the computing delegation. In the latter case, the data manager receives the encrypted text, performs the necessary computation, and sends back the encrypted result. The data owner carries out the coding and decoding of the data, and after decoding it, the communication with respect to the relevant data will not continue. Thus it is obvious that the data manager gets into contact only with encrypted texts, and therefore he can launch ciphertext only attacks, which is the weakest among all attacks. This attack can be made even weaker, if the data owner uses a different key for encoding for every single computation needed.

When data delegation is performed, returning the encrypted result does not terminate the communication between the data owner and the data manager, according to the fact that the data manager claims the decoded result. Accordingly, the data owner returns it to the data manager after decoding the encrypted result. In other words, the data manager will possess (plaintext, ciphertext) pairs even when the operation is normal, so that the premises are given for starting a known plaintext attack. Changing the key frequently is a useful tool to weaken the attacks in this case as well.

Attack against the RSA-related homomorphism

The RSA-related homomorphism can be broken, i.e. the values of p and q can be restored, by a known plaintext attack. The attack is performed as follows: The plaintexts $M_1, M_2, \ldots M_r$ as well as their encrypted correspondents $E_{(p, q)}(M_i)=(C_i, D_i)$ are available, with $M_i=C_i \bmod p$, $M_i=D_i \bmod q$ and $i=1, 2, \ldots r$.

According to the definition of the congruence, it is obvious that $p|C_i-M_i$, $i=1,\ldots,r$. Let us take the greatest common divisor of these differences, and let it be $p'=\gcd\{C_i-M_i : i=1,\ldots,r\}$. Similarly we define q'. As p and q, respectively, are divisors of the differences C_i-M_i and D_i-M_i, they are also the divisors of their greatest common divisor (p' and q', respectively), i.e. $p|p'$ and $q|q'$. Even for a small r there is a high probability of $p'=p$ and $q'=q$. If this is not the case, every new $(M, (C, D))$ pair lets the attacker come closer to the secret primes (i.e. the key) by introducing $p''=\gcd(C-M, p')$ and $q''=\gcd(C-M, q')$, respectively.

The generalization of the homomorphism presented in [4] permits to defend fortunately this kind of attacks. In addition, the RSA-related privacy homomorphism is so "successful", that this will be used as a starting point for the solution of a data delegation system. The next section shows more details on the results.

What is known about privacy homomorphisms?

The possibly most restricting feature in the use of privacy homomorphisms had been detected as early as at the beginning of the researches. If the person performing the computation tasks has an option to encrypt any constant, and is able to compare encrypted texts by means of the predicate $\leq$, then the privacy homomorphism is not secure, so that it can not resist even the weakest attack, i.e. the ciphertext only attack. In fact, the value of $a'=E_K(a)$ can be "caught" through a binary search, thus the value of a can be easily defined.

To start the search, the encrypted constants are needed, e.g. it may be known that $E_K(1)=1'$. Hence, due to the operation preservation, $2'=E_K(2)=E_K(1+1)$

$= E_K(1)‡E_K(1) = 1'‡1'$ can be calculated. + denotes the operation which the homomorphism will preserve, and ‡ is its equivalent in the image space. This process can be continued till the comparison lets us find a $2'^n$ where $2'^{n-1} \leq a' \leq 2'^n$. While continuing the search, we verify whether $a' \leq 2'^{n-1}‡2'^{n-2}$ is fulfilled. If so, we continue the search in this half of the interval by applying the bisectional method, i.e. by verifying the $a' \leq 2'^{n-1}‡2'^{n-3}$ condition; if not, we continue to bisect the other half of the interval and verify the fulfilment of the $a' \leq 2'^{n-1}‡2'^{n-2}‡2'^{n-3}$ condition.

At the end of the search, we will have the sum of the encrypted powers of two: $a' = 2'^i‡2'^j‡…‡2'^m$. That is $a' = E_K(2^i)‡E_K(2^j)‡…‡E_K(2^m)$, which means, due to the operation preservation, that $a' = E_K(2^i+2^j+…+2^m)$, i.e. $E_K(a)=E_K(2^i+2^j+…+2^m)$, or in other terms $a = 2^i+2^j+…+2^m$. The right-hand side of the equation can be calculated, so also the value of a can be defined.

It has been proven in [1] that the additive privacy homomorphisms cannot resist the chosen ciphertext attack. Note that for additive homomorphism, the set of ciphertexts can be regarded as a vector space above the body {0, 1}. Let us select a based in this vector space (for example the encrypted "powers of two"), and ask for the parent image of it, i.e. its plain version. Thus the following pairs are available: (1, 1'), (2, 2'), (2^2, $2'^2$),... Therefore, if a given $a' = E_K(a)$ encrypted text is supposed, based on which we want to define a, then logically a' can be written as a sum of the encrypted powers of two, i.e. in the selected base: $a' = 2'^i‡2'^j‡…‡2'^m$. From this point on, the attack can continue in the same way as in the case mentioned before.

The restriction of additivity outlined in [3] has the goal to defend this attack. The r-additive privacy homomorphisms permit the summation of only r members. So, if r is sufficiently small and the set of the ciphertexts is sufficiently large, the above-mentioned attack does not work, and the homomorphism is secure for ciphertext only attacks.

The [4] shows a generalization of the RSA-based homomorphism presented. Even in its original form, the homomorphism has preserved addition and multiplication, but as we have seen, it has not been secure against a known plaintext attack. The point of the generalization is that the former encrypted text (a mod p, a mod q) appears in a more complex form, so there is no possibility to set up the statements relating to the divisibility.

The result shown in [5] presents a breakthrough in the number of the operations preserved. Note that this is a privacy homomorphism, which preserves all four field-operations (+,-,?,/) and is at the same time resistant against the ciphertext only attacks. However, as the results highlighted in [1] say, the homomorphism of such type is able to achieve a security level where it can stand also against known plaintext attacks. Finally, [2] has included an advanced version of the additive and multiplicative homomorphism introduced in [4], which already facilitates the division as well and, at the same time, is provably a secure option in the case of the known plaintext attacks as well.

Based on this homomorphism, the prototype of a system delegating sensitive statistical data has been created and standardised by the authors as a "Method for secure delegation of statistical data" [7], so we have no further information on the features of the system.

The above-mentioned homomorphism is a useful tool for any computation, which needs field-operations (+,-,?,/) only. More complex financial or engineering computations may however require additional operations as well, for example logarithm and exponentiation.

An additional, interesting question is whether the personal income tax can be solved by means of the privacy homomorphism. Of course, the relevant tax calculation software tools can be downloaded from the Internet by means of which the most appropriate declaration version can be found out under the shelter of one's own computer, but an application might be more comfortable that could permit to specify in on-line mode the imposed sum of the tax, based on different input data. But because the user would certainly experiment with a series of values, he would not willingly send out such data in a form ready for processing.

Logarithmic calculation with privacy mapping

The logarithmic calculation can be solved by a privacy mapping operation, which, strictly speaking, cannot be regarded as a privacy homomorphism. Let R^+ be the set of plaintexts and ciphertexts, i.e. the set of positive real numbers. Let the key be an arbitrarily chosen positive integer r. The encrypted version of an $a \in R^+$ is $E_r(a) = a^r$. The decryption is not exactly the inverse of E, as we presume that we will have also a logarithmic calculation between the encryption and the decryption. The deciphering of a ciphertext a' is accordingly delivered by $D_r(a') = a'/r$.

It is easy to verify that if we decrypt the logarithm of the encrypted text, we obtain the logarithm of the plaintext, because $(\log(a^r))/r = (r \times \log(a))/r = \log(a)$. It can however be stated that this is not a classical privacy homomorphism, as in this case, the logarithmic operation interpreted on the set of plaintexts is the same as its counterpart interpreted on the set of the ciphertexts, thus the following equality will not be fulfilled: $E_r(\log(a)) = \log(E_r(a))$, as $(\log(a))^r \neq \log(a^r)$.

Impediments of the tax calculation

An on-line income tax calculation service is burdened with the problem of the computing delegation. In this section we outlined some concepts that could be of use for the exploration of an appropriate homomorphism.

This service could be operated for example by an accountant organisation addressed by the tax-paying entity ordering the calculation of the most advantageous taxation form. The taxpayer is clearly not in the possession of the extensive knowledge required for this work, and, on the other hand, he would not willingly expose his financial situation for the service provider.

Once the aspects able to influence the tax base and the tax extent taken into account, the calculations will land sooner or later at a point where we have to define from the tax schedule the sum of the tax relating to the income. Because the tax function is linear but changes interval by interval, ranging the income between the relevant limits must precede the definition of the function value. For this purpose, we need the encrypted version of the end points of the intervals and also an encrypted comparative predicate. The tax function can be supposed to be known for the service provider, and if so, the text pairs of the end points of the interval (plain, encrypted) are available for a possible attack.

Under such conditions the privacy homomorphism will not be secure, because a searching attack can be carried out by means of the comparison. Let us suppose that the encrypted image of the income representing the tax base is $x'= E_K(x)$. We intend to use the tax function for the calculation of the relevant tax value. First we shall define the tax range or interval in which x' lies. The breakpoints of the tax function are 0, a, b, c, and their encrypted correspondents are $0'= E_K(0)$, $a' = E_K(a)$, $b'= E_K(b)$, $c'= E_K(c)$. It can be supposed that $a'\leq x'\leq b'$. Then the value of x can be defined on the interval [a', b'] by means of the binary search described earlier.

But we can also choose the solution according to which we specify the tax schedule individually for each income sum comprised in every interval [1,N]. This means that we define the tax function for each value within a constrained definition domain, of course in an encrypted form. No doubt, the service provider may know the open tax schedule in its present form as well, that is he knows the pairs (x, $t(x)$), with t being the tax function. Although we also make available the pairs (x', $t(x)'$) for him, the data manager will not be able to compose from them the quarts without any additional information. Thus it seems that we did not have shown favour toward the data manager as a potential attacker by ensuring the conditions to perform a known plaintext attack. On the other hand, some apparent efficiency questions are also arising (encryption and transfer of a huge quantity of data) that could impair the application possibilities of the concept.

Summary

The privacy homomorphisms have been introduced in relation to data delegation and computing delegation, but because a great number of Internet-based services are marked by these problems, their application possibility has again come to the fore [8]. The privacy homomorphisms are confined into specific limits, but the performance of the statistically relevant calculations at an unreliable level is ensured by means of a mapping concept being provably secure, which preserves all the four field-operations (+,-,?,/).

At present there exists no convenient homomorphism which could be a viable solution for complex services, such as for example in the case of applications involving logarithm or limit value calculations. Finally we have outlined a few concepts, which could possibly be set as a guide for future researches.

References

[1] N. Ahituv, Y. Lapid, S. Neumann:
Processing Encrypted Data,
Communications of the ACM,
Vol.30, No.9, pp.777–780,
September 1987.

[2] J. Domingo-Ferrer:
A Provably Secure Additive and
Multiplicative Privacy Homomorphism,
Information Security 2002,
Lecture Notes in Computer Science,
Vol.2433, pp.471–483

[3] E. Brickell, Y. Yacobi:
On Privacy Homomorphisms, Advances in Cryptology,
EUROCRYPT '87,
Lecture Notes in Computer Science,
Vol.304, pp.117–125.

[4] J. Domingo-Ferrer:
A New Privacy Homomorphism and Applications,
Information Processing Letters,
Vol.60, No.5, pp.277–282.,
December 1996.

[5] J. Domingo-Ferrer, J. Herrera-Joancomartí:
A Privacy Homomorphism Allowing Field Operations
on Encrypted Data,
Jornades de Matemàtica Discreta i Algorísmica,
Barcelona, March 1998.

[6] R. L. Rivest, L. Adleman, M. L. Dertouzos:
On Data Banks and Privacy Homomorphisms,
Foundations of Secure Computation,
pp.169–179.,
New York, 1978.

[7] J. Domingo-Ferrer, Ricardo X. Sánchez del Castillo:
Method for secure delegation of statistical data,
P9800608 patent in Spain,
December 2000.

[8] C. Boyens, O. Günther:
Trust is not Enough: Privacy and Security in ASP
and Web Service Environments,
Advances in Databases and Information Systems,
6th East-European Conference, ADBIS 2002,
Lecture Notes in Computer Science,
Vol. 2435, pp.8–22.