UDC 004.056.5 **DOI:10.30837/rt.2019.3.198.09**

V.I. YESIN, Dr. Sc. (Engineering) V.V. VILIHURA

SOME APPROACH TO DATA MASKING AS MEANS TO COUNTER THE INFERENCE THREAT

Introduction

Information in the modern world has become one of the most important resources of society, and information systems (IS) whose main functional components are databases (DB) have become a necessary tool in almost all spheres of human activity providing him reliable information for making optimal decision. That in turn was reflected in the reverse side of this process. Namely, interest in the information circulating inside the IS has increased not only from legitimate users and owners, but also from attackers. The database, as the most important corporate information resource, is one of the most vulnerable and attractive elements of IS for attackers. Attackers are interested in many things: internal operational information of the organization, enterprise, company, personal data of employees, financial information, information about customers, intellectual property products, market research results, etc. The main threats to database security today are [1-5]:

- excessive and unused privileges;
- legitimate privilege abuse;
- input injection;
- malware;
- weak audit trail;
- storage media exposure;
- exploitation of vulnerable databases;
- unmanaged sensitive data;
- inference;
- denial of service;
- limited security expertise and education (the human factor);
- database communication protocol vulnerabilities and some others.

Attacks on data warehouses (DW) and databases are one of the most dangerous for various enterprises and organizations, practically any branch, be it banks and finance, medicine, trade, high technology, industry, transport, government agencies and law enforcement agencies, education, municipal institutions, etc.

According to statistics, in recent years the number of breaches and the amount of compromised data in the world has been steadily increasing. Thus, according to the estimates of the InfoWatch analytical center over the past 12 years (2007 – 2019), more than 30 billion personal data records have been compromised in the world, including over 20 billion in the last two years [6, 7]. And this is despite the fact that various security experts around the world are constantly doing a great job of researching, creating and developing principles and systems for protecting information.

Among the threats that are difficult to control within the framework of the database management systems (DBMS) and data warehous, a special place is occupied by the threat associated with the ability of attackers to make conclusions based on various algebraic calculations, comparisons, filtering the data to which they are admitted, without the need for direct access to the protected object itself [8]. This so-called threat of inference. Inference is a way to infer or derive sensitive data from nonsensitive data [1, 2, 8], which is closely related to two other ways of obtaining confidential information – aggregation and association of data [1]. The problem of aggregation and association occurs whenever a set of aggregating or associated data forms more important information than the importance of the individual data based on which this information is obtained. So information about the activities of one department or branch of the corporation have a certain weight. The data for the entire corporation are already much more significance. Or another example: a list consisting of the names of all employees and a list containing the salaries of employees are open on their own, and the combined list with the names of employees and their salaries is already confidential information.

The usage of complex as well as a sequence of simple logically related queries allows an attacker to obtain data that is not directly accessible to him. Such a possibility is available, first of all, in databases that allow to obtain statistical data [9, 10]. Big Data is also subject to the same threat [8]. In the era of Big Data, the problem of protecting sensitive data is further exacerbated, as technical methods of protecting privacy are losing ground [11]. In this regard, for example, Google when implementing the Google's Street View (for this, photos of roads and houses in many countries of the world were collected) in Germany faced widespread public and media protests. People feared that photos of their houses and gardens could aid gangs of robbers to choose profitable targets [11].

To date, there are no perfect solutions to the inference, aggregation and association problems [8]. As a rule, countering to similar threats is carried out by such methods as [1, 2, 8, 10]:

– blocking the response with the wrong number of queries;

– control of incoming queries;

– distortion of the response by deliberate correcting the data: limited response suppression; combined results; random data perturbation; swapping; concealment;

– random sample;

– context-oriented protection;

– polyinstantiation;

– auditing and some others.

At the same time, it is important to understand that, by limiting the attacker's ability to make inferences based on the information obtained using such methods, we automatically limits queries from users who do not intend to implement unauthorized access to data. Moreover, attempts to check requested accesses for possible unacceptable inferences may actually degrade the performance of the DBMS [8].

Much of the research on inference based on data obtained in various ways from databases, data warehouses was done in the early 1980s. However, and today, aspects related to the compromise of data confidentiality obtained through inference remain largely unresolved [8].

The following is an approach to hiding data, making it difficult for an attacker to implement the inference threat. It is based on the principles of random permutation of elements (bytes, characters) of a specific field of the corresponding column (attribute) of a row (tuple) of data and dynamic data masking (DDM), defined by Gartner as an emerging technology that aims at real-time data masking of production data [12]. At that, a feature of the proposed approach is that a preliminary physical change of sensitive data is made in the production database, and it is possible, if necessary (if masking is no longer required), to lead all changes made during the masking to the initial state (without data masking) by the user who has the corresponding rights to it. This profitably distinguishes the proposed mechanism from most of the typical commercial tools for masking sensitive data. The legitimate user in the proposed approach gets access to sensitive data due to the ability to transform (rewrite) the query "on the fly", and the attacker can only read the previously modified data that is stored in the database. This approach can also be used in non-production databases, expanding the possibilities of the so-called static data masking.

1. Preliminaries. Data masking method based on the calculation of modulo operations

Today, various masking methods are known (in various sources one can also find the following terms related to information hiding, such as data anonymization, data de-identification, data scrambling, data scrubbing , data obfuscation), which are widely used in certain classes of tasks, namely [2, 13 – 15]:

– substitution. This technique consists in randomly replacing the contents of a data column by information that looks similar but completely unrelated to the real data (for example, real customer last names in the database can be replaced with last names taken from a large random list). Substitution is very effective in terms of preserving the appearance of existing data. The disadvantage is that for each column to be replaced, a large amount of replaceable information must be available;

– shuffling is a technique of random shuffling the existing field values in a table column (for example, data of a table column containing medical records about the patients' health status are randomly shuffled);

– random data deviation (random decimal numbers, random dates, random digits, random strings) is number and date variance technique. The existing value is replaced with a random one in a certain range. This technique can prevent attempts to discover true records using known date data or the exposure of sensitive numeric or date data;

– encryption. The format preserving encryption (FPE) method is used, since ordinary encryption, as rule, changes the format of the original data and may increase the data dimension, which is not always desirable;

– nulling out or deletion is simple deletion of column data by replacing it with NULL;

– masking out. This technique is a special case of the substitution technique, when all masked characters are replaced with the same symbol, for example, “X” (in this case, the credit card number would be 4343 XXXX XXXX 7357);

– technique of masking numerical data using modulo operations (MOBAT – Modulus Based Technique);

– compound masking is the technique of masking related columns as a group, ensuring that masked data across the related columns retains the same relationship. For example, consider masking address fields, such as city, state (region), and postal codes. These values must be consistent after masking;

– tokenization. In this technique, data elements are replaced with random tokens – values that should not be associated with the replaced sensitive data either mathematically or in any other way. The token does not carry any confidential information, it is only logically associated with real data that is stored in a well-protected database

and some others.

However, most of the masking techniques described above, except for the encryption, tokenization and MOBAT techniques are used for static masking of non-production databases and, after their application, do not allow canceling operations in order to return to the original data, that is not always acceptable. This is especially important for production databases if it is supposed that this mechanism will be used to counter the inference threat.

At that, the encryption method is quite resource-intensive, and MOBAT is specifically designed to mask only numerical values [15]. But, quite often there is a need for masking not only numerical values. For example, in databases built on the basis of the schema with the universal basis of relations [16, 17], that can be used, including as data warehouse of various subject domains, the attributes (columns) of relations (tables) containing sensitive data are defined on the domain of character strings.

Therefore, the need arose to find some new solution that would be no worse than the existing ones and would allow, to a certain extent, to reduce the probability of the inference threat.

After analyzing the capabilities of the above techniques, as well as the best practices of hiding information from leading vendors in the masking market [18], first an attempt was made to use mathematical transformations based on calculating modulo operations not only for numerical, but also for data types such as a character string, converted to a numerical value.

So, for example, the characters of the string 'Abc123-Ю', first converted into a hexadecimal string `4162633132332DD0AE` (similar character conversion is done in the encryption method with preservation of the format [19]), is converted to a numerical value in the decimal number system – `1206127929121208914094`, over which the transformation is then performed similar to the MOBAT technique:

$$R'_{ij} = R_{ij} - ((K_3^i \bmod K_1^R) \bmod K_2^j) + K_2^j, \qquad (1)$$

This is a direct transformation, which is necessary to mask a specific value of the field R_{ij} of the i tuple of specified attribute j of the certain table (relation) R, where R'_{ij} is the masked value of the field; K_1^R is a 128-bit random generated value (private key) that is constant for the table R; K_2^j is a 128-bit random generated value (private key) that is constant for the attribute j of the table R; K_3^i is a public key, each value of which is determined by the value of the i-th row field, one of the selected column (attribute) of the table. For this purpose, it is recommended to use a long integer typed column. Namely, as a public key, it is best to use the value of the integer identifier (ID) column of the primary key of the table R.

The inverse transformation is performed to unmask a specific value of the tuple field of the specified attribute of the certain table R:

$$R_{ij} = R'_{ij} + ((K_3^i \bmod K_1^R) \bmod K_2^j) - K_2^j, \qquad (2)$$

To perform mathematical transformations (1), (2) over the obtained long integers, algorithms and programs for their implementation were developed. The algorithms associated with the implementation of computational operations on converting long integers into different number systems, without which in practical implementation it was impossible to do, are based on the Horner's rule, which allows reducing computation and memory.

However, after calculating modulo operations over converted characters, in general, the result contained not only normal (print), but also control characters, which made it difficult to represent them in a readable form. Therefore, the result was represented in the form of a hexadecimal string, which is convenient to store and process in the table field with the type of a character string, or convert to decimal form, which is convenient for numerical processing. However, such a representation could not always be suitable for various kinds of applications. In addition, when using long character strings, the implementation time was not only increased, but the high bytes were practically not transformed due to insufficient key length, except for converting string type values to numerical type. This is shown in example 2 below.

Example 1. Let in some sequence of rows of one of the table columns, whose data should be masked, the string of Cyrillic characters 'ГРС-56' is stored.

For a better representation and understanding of the corresponding data transformation, let it be sequential rows (table rows with sequentially increasing numbering (increment on ID)).

The result of applying the formula (1) to the values ('ГРС-56') of the column (we define it as DATA_X) the corresponding rows of some table is represented below:

```
        ID DATA_X
---------- --------------------------------
     19746 E144A5509211952AEBE3C470A25F8C0
     19747 CF518B3AC5B88CB41835AE555349764
     19748 8BDC0D5DE66F27BF9280344028AB827
     19749 8F48799ABEC3709CE2D45656532205E
     19750 506741F22A4834CDEA6DEE65EACD651
     19751 E5CBFC700E29E9374A81A902FF7BD9F
     19752 C70DA272B74C3303321CA6E4FCF0A20
     19753 1600D8CE183AEE9AE073EA796A06275D
     19754 1649CD15E479D33CADD0193E19894C1D
     19755 14D66A41674FFD516D9CC884BD8A6530
     19756 C39366DA0C9E841C5A7516A42A41EDF
     19757 FF1C5FA372FBA9337CE685174794AC7
     19758 90A9BDB60B7BFDD79602050C8E8B1E9
     19759 11DA56F36CB5A2611031BAD1BE2CC97F
```

Where ID column values are the public keys K_3^i for the i-th row of the table. In considered case these values are in the range [19746, 19759] ($K_3^i \in [19746,\ 19759]$).

Example 2. Let the Cyrillic character string `'Фосфоритный рудник'` is stored in some rows of the same column DATA_X.

Applying formula (1) to the corresponding data column leads to the following result (representation of the transformed character string in different table rows):

```
        ID DATA_MASKING
---------- ------------------------------------------------------------------
     20659 D0A4D0BED181D184D0BED180D0B8D182D0BDD19EC501E1077C3320E176F0F1B28A3505
     20660 D0A4D0BED181D184D0BED180D0B8D182D0BDD193555D8DA87CF439DBA3A015B0ACC2C7
     20661 D0A4D0BED181D184D0BED180D0B8D182D0BDD196A3BCDB859EA0ED94B53DB4421A399B
     20662 D0A4D0BED181D184D0BED180D0B8D182D0BDD1957238A723EF5F6E3989C1D314106159
     20663 D0A4D0BED181D184D0BED180D0B8D182D0BDD1913798672CA2D33A572138BA761E3594
     20664 D0A4D0BED181D184D0BED180D0B8D182D0BDD193224CFBA0E0590C3D70A2EACD6F05BE
     20665 D0A4D0BED181D184D0BED180D0B8D182D0BDD1A1F3333683E2691BA17399EC724353CD
     20666 D0A4D0BED181D184D0BED180D0B8D182D0BDD193A795A0B46246692D738F4A63F4B648
     20667 D0A4D0BED181D184D0BED180D0B8D182D0BDD19F44BF9C5416208F6C5E33D44F9B811C
     20668 D0A4D0BED181D184D0BED180D0B8D182D0BDD190AEE542469687FCE3FF8DD1C18DEA7C
     20669 D0A4D0BED181D184D0BED180D0B8D182D0BDD19D68DAD6C0BAA7FC24FAE62B2526F3D1
     20670 D0A4D0BED181D184D0BED180D0B8D182D0BDD19BF2CB316D1D2908F7FDC4F142E73CD0
     20671 D0A4D0BED181D184D0BED180D0B8D182D0BDD196FA6E18263F66DA86C3793598F6B2FD
     20672 D0A4D0BED181D184D0BED180D0B8D182D0BDD1A0892F4B6BE930B3E3D0D7CFFEE311CA
     20673 D0A4D0BED181D184D0BED180D0B8D182D0BDD19EBF7F8AAC53585510F40366E236CF5A
     20674 D0A4D0BED181D184D0BED180D0B8D182D0BDD19C1A189D7A0A289CB6F5AC214962308A
```

As you can see from the last example, the part of the transformed data (highlighted in color) for the same original row is the same.

To eliminate this shortcoming, it became necessary to mix them. As one of the expedient options, an obvious solution was seen, the essence of which lies in the random permutation of the corresponding bytes of the received row code with the possibility of their inverse recovery.

As is known, most of the cryptographic algorithms are still combining substitutions and permutations (transposition) [20, 21], what else C. Shannon noticed in his work [22], summarizing the experience gained before him in developing ciphers. As it turned out, even in complex ciphers, simple ciphers, such as substitution, permutation, or a combination of them, can be distinguished as its typical components. "Substitution and transposition are still the most important kernel techniques in the construction of modern symmetric encryption algorithms" [21]. Well-known computer security experts, cryptography N. Ferguson, B. Schneier in the monograph [23], talking about what would the ideal block cipher look like, note that this should be a random permutation. Specifying at the same time that for each key value the block cipher must be a random permutation of the plaintext variants and the different permutations for the different key values should be chosen independently.

The proposed solution led further to a new method.

Thus, the initial approach led to a new more efficient and less computationally expensive method – the method of random permutation of the data elements of the row field. Although in some cases, for example, when small length of rows or increasing the key length and parallelizing the computation processes, and the initial approach can be used.

2. Hiding sensitive data of row field by method of random permutation of its elements

As it is known, a permutation of n objects is an arrangement of *n* distinct objects in a row [24]. If we number the places of these objects from left to right (1,2,…,*n*), then we can formulate the following definition: the one-to-one mapping $p: A \to A$ of a finite ordered $A = \{a_1, a_2, ..., a_n\}$ set from n elements onto itself is called a permutation of elements of the A set. In the general case, for n-element set A with a fixed order of $a_1, a_2, ..., a_n$ elements the permutation is an arbitrary sequence of length n from different elements of the set A. Permutations of n elements of the set A differ from each other only in the order of their elements.

Permutation p can be written as a matrix of two rows. For example, the permutation $\pi : A \to A$ of the set $A = \{a, b, c, d, e\}$ such that $\pi(a) = e$, $\pi(b) = d$, $\pi(c) = a$, $\pi(d) = b$, $\pi(e) = c$ can be written as follows:

$$\pi = \begin{pmatrix} a & b & c & d & e \\ e & d & a & b & c \end{pmatrix}. \tag{3}$$

Usually the nature of the elements of the set A are inessential, so without loss of generality, we can be considered that $A = \{1, 2, ..., n\}$ (otherwise you must go to the element numbers (a_i, where $i \in \{1, ..., n\}$)). Then each permutation π of these elements can be written as a matrix of the following two rows:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ a_1 & a_2 & a_3 & ... & a_n \end{pmatrix}, \tag{4}$$

where $\{a_1, a_2, a_3, ..., a_n\} = \{1, 2, 3, ..., n\}$, $\pi(i) = a_i$, for all $i \in \{1, ..., n\}$.

The number of all permutations π from n different elements is equal to $\pi_n = n!$

For each permutation π, there is a inverse permutation π^{-1} that undoes the effect of π. The product $\pi \cdot \pi^{-1}$ equals the identity permutation $\pi_e = \pi \cdot \pi^{-1}$.

The inverse permutation $a_1', a_2', a_3', ..., a_n'$ is obtained, if in (4) swap the rows of the matrix, and then arrange the columns in ascending order by the upper elements:

$$\begin{pmatrix} a_1 & a_2 & a_3 & ... & a_n \\ 1 & 2 & 3 & ... & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ a_1' & a_2' & a_3' & ... & a_n' \end{pmatrix}. \tag{5}$$

Let us apply the above theoretical information from combinatorics to the solution of our task of masking the specific value of R_{ij} field of i tuple of the specified attribute j of the specific table R, through the random permutation of the sensitive data elements of the row field. Randomness in this case means the equiprobability of obtaining any of $n!$ possible permutations from the set A.

The proposed method of random permutation of elements (bytes, characters) of data of a specific field of a different type (numeric, character strings, Binary Large Objects (BLOBs), Character Large Objects (CLOBs)) of table row is based a modern version of the Fisher-Yates shuffle algorithm [25], presented by R. Durstenfeld in [26]. This algorithm (called "Algorithm FY") in pseudocode is presented below.

```
Algorithm FY
-----------------------------------------------------------------------
Input: A is an array with n≥2 elements (n is permutation length)
Output: random permutation on A
   for i = n downto 1
     j = random(1..i) /* a random number is generated in the range [1,i] */
     swap(A[i], A[j]) /* exchange */
   end for
```

The main reasons for choosing the Fisher-Yates shuffling algorithm were its following advantages:

– a small number of steps performed operations. The asymptotic computational complexity of the modern version of the algorithm is $O(n)$, where n is the number of elements of the set (in this case, this is the number of elements (characters, bytes) of the data of specific field);

– when using a high quality unbiased random number generator, the algorithm guarantees an unbiased result;

– its efficiency and simplicity have so far stood the test of time [27].

The Fisher-Yates algorithm uses a sample of uniformly distributed random numbers from different ranges. Therefore, it is important that one could take advantages of this algorithm, it is necessary to use pseudorandom number generators (PRNGs), which form random numbers that exactly are unbiased in some part of the interval. On the other hand, as noted in [28], the Fisher-Yates algorithm is not able to generate more than m different permutations, that is, it cannot create more permutations than the number of internal generator states. And even when the number of possible generator states exceeds the number of permutations, some of them may appear more often than others. In order to avoid the appearance of distribution unevenness, it is usually recommended that the number of internal states of a random number generator exceed the number of permutations by several orders of magnitude, if this is actually possible. Although in most cases, there is actually no need to receive all permutations [28].

Therefore, based on the foregoing, the proposed method provides for the possibility of using, depending on the situation (this is mainly due to the need to perform the procedure of permutation of the data elements of various type fields with minimal time costs), different PRNGs that satisfy the above requirements (cryptographically strong PRNG in this case is not required). We need to perform such transformations – permutations in advance so that an attacker cannot, using complex as well as sequences of simple logically related queries, obtain data that is not directly accessible to him, based on inference (that is realize the inference threat) for an acceptable time for him. At that, a legitimate user could get the required sensitive data quickly enough and simply.

In the proposed implementation of the method were used:

1) linear congruential random number generator, popularized in [29]:

$$X_{j+1} = (aX_j + c) \bmod m \tag{6}$$

with constants (multiplier $a = 1664525$ and increment $c = 1013904223$) chosen by D. E. Knuth and H. W. Lewis, where $m=2^{32}$;

2) random number generator of built-in DBMS_RANDOM package for Oracle DBMS, namely DBMS_RANDOM.VALUE, which generates floating-point numbers with 38 digits to the right of the decimal (38-digit precision), with the possibility of setting them various range;

3) G. Marsaglia pseudo-random number generator (Xorshift) [30] with a period of 2^{128}-1:

```
unsigned long xor128()
{static unsigned long x=123456789, y=362436069, z=521288629, w=88675123;
unsigned long t;
t=(x^(x<<11)); x=y; y=z; z=w;
return(w=(w^(w>>19))^(t^(t>>8)));
}
```

In the Xorshift generator, some initial sequence is specified, to which the operations of the exclusive OR (XOR) and logical shift are applied. This PRNG was selected based on the recommendations given in [31]. In principle, based on these recommendations, you can choose any other PRNG, including those given in the same work [31].

All the generators in the software implementation were checked the correspondence of the output random numbers in the given range to the uniform distribution law.

Masking algorithm

Preliminary remarks. The proposed solution uses a universal scheme for hiding data of various type fields of a tuple row of some database table, based on the use of public and private keys K_1^R, K_2^j, K_3^i. K_1^R is a unique 128-bit random value (private key) generated by a cryptographically strong PRNG for each table R (it is constant for all values that will be masked in this table). K_2^j is a unique 128-bit random value (private key) generated by a cryptographically strong PRNG for each attribute j of the table R (it is constant for all values that need to be masked in this column).

K_3^i is a public key based on the value of the integer identifier of the primary key of the i-th row of the table R (it is constant for all values in the columns that will be masked in this row).

An authorized user, with appropriate privileges, which will provide the correct key in an open session to decrypt the row of the special table (R^{secret}) encrypted with the AES-256 algorithm (keys and some other information, such as table and column names, are stored in the rows of this table), has the mediate access (through the corresponding middleware) to the private keys K_1^R, K_2^j. All other users, even privileged, without knowing this key, will not be able to extract the private keys from the table R^{secret}, and, therefore, will not be able to perform neither data masking operations nor reverse operations (unmasking or inverse masking).

Algorithm operations

1. The initial value ($X_{R_{ij}}^0$) of PRNG is generated.

For each row i of the corresponding column j of the selected table R, this value is different:

$$X_{R_{ij}}^0 = hash(K_1^R + K_2^j - K_3^i)\bmod(N_{\max}), \qquad (7)$$

where $hash()$ is one of the cryptographic hash functions (such as: MD4, MD5, SHA-1, SHA-256, SHA-384, SHA-512, SHA-3). The purpose of using a hash function is mixing (non-injectively transform) private and public keys to make it impossible to recover them from the final result and getting significantly different from each other formed initial values $X_{R_{ij}}^0$ for PRNG, even if at least one of these keys changes by one character (one). $N_{\max}$ is the maximum allowable integer in the corresponding implementation. Modulo operation $N_{\max}$ is also non-injective.

2. Actions are performed in accordance with the Fisher-Yates algorithm:

– a random integer is generated (using one of the selected PRNG, to the input of which the generated initial value $X_{R_{ij}}^0$ is supplied);

– the permutation procedure of elements (bytes, characters) of the source string A of length n is performed.

As a result, we have the transformed (masked) field value (A) of each row i of the column j of the corresponding type for the selected table R.

The general scheme of the masking algorithm (MA-1) is represented below in pseudocode.

Masking algorithm 1 (MA-1)

```
Input: name_table, name_column, $K_3^i$, $A$, $N_{\max}$
Output: masked value of the data string - $A$
    Decrypt($R^{secret}$[name_table, name_column]) → ($K_1^R$, $K_2^j$, PRNG, hash)
    $X_{R_{ij}}^0 = hash(K_1^R + K_2^j - K_3^i) \bmod (N_{\max})$
switch(PRNG)
{case 1: linear congruential generator (LCG)
 case 2: built-in random number generator (package DBMS_RANDOM)
 case 3: pseudo-random number generator Xorshift
 ...       }
 for i = n downto 1
  j=random_PRNG(1..i)  /*a random number is generated in the range [1,i]*/
  swap(A[i], A[j])      /*exchange*/
 end for
```

Without knowing the initial value $X^0_{R_{ij}}$, it is rather difficult, and with long data strings (large dimensions A (large n)) it is almost impossible to determine the sequence of random numbers generated for the permutation (the number of which is equal $n!$). This means that it will be difficult for an attacker to determine the source strings after their corresponding transformation. Thus, we restrict an attacker to use a various set of queries for inference, for example, by presenting the same data stored in the database in a different form. As a result, misleading the attacker, we counteract the inference threat.

Example 3. Let the following string of Cyrillik characters `'КРП-17'` ($A = \{К, Р, П, -, 1, 7\}$) be stored in some row of one of the table columns whose data should be masked.

Applying the MA-1 algorithm (*Masking algorithm 1*), and getting, for example, one of the random permutations:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 3 & 4 & 1 & 2 \end{pmatrix},$$

we have the transformed (masked) value, namely, the string of characters $'7Р-К1П'$, as a result of the mapping: $'КРП-17' \to '71П-КР'$, that is $\pi('К') = '7'$, $\pi('Р') = '1'$, $\pi('П') = 'П'$, $\pi('-') = '-'$, $\pi('1') = 'К'$, $\pi('7') = 'Р'$, where a_i are the numbers of the i-th element ($i \in \{1, ..., 6\}$) of the source character string to be masked, each which should be placed in the appropriate position after transformation. So a_1 indicates to the sixth character ('7') of the string $'КРП-17'$ that should be placed in the first position of the transformed string; a_2 indicates to the fifth character ('1') of the string $'КРП-17'$ that should be placed in the second position of the transformed string, etc. for a_3, a_4, a_5, a_6.

Algorithm inverse to masking algorithm

Preliminary remarks. The proposed solution for recovering the masked data of the row field uses an inverse permutation algorithm, similar to that described in [24], with the peculiarity that it does not limit the permuted elements to only numbers $\{1, 2, 3, ..., n\}$, but can use any characters of national alphabets, numbers represented in hexadecimal or other number system. Namely, having an initial permutation:

$$\pi = (\pi(1), \pi(2), ..., \pi(n)) \tag{8}$$

and the result of its application:

$$(y_1, y_2, ..., y_n) = (x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(n)}), \tag{9}$$

the inverse permutation can be obtained using the formula [21, 24]:

$$\pi^{-1}(\pi(i)) = i, \tag{10}$$

as:

$$(x_1, x_2, ..., x_n) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, ..., y_{\pi^{-1}(n)}), \tag{11}$$

or in matrix form, as:

$$X[\pi(i)] = Y(i). \tag{12}$$

If through π_{ch} we denote the permutation $\pi_{ch} = \begin{pmatrix} x_1 & x_2 & x_3 & ... & x_n \\ y_1 & y_2 & y_3 & ... & y_n \end{pmatrix}$, where $y_i = \pi(x_i)$, $i \in \{1, ..., n\}$, and through $\pi_{num} = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ a_1 & a_2 & a_3 & ... & a_n \end{pmatrix}$ the corresponding permutation for it, as a mapping of the numbering onto the corresponding elements of the set, where

$\{a_1, a_2, a_3, ..., a_n\} = \{1,2,3,...,n\}$, then the formula (12) for the inverse permutation can be written as follows:

$$\pi_{ch}^{-1}(\pi_{num}(i)) = \pi_{ch}(i)\,. \tag{13}$$

However, first you need to get the initial permutation before finding the inverse for it. Therefore, the sequence of actions will be as follows.

1. The initial value ($X_{R_{ij}}^0$) of PRNG is generated.

It is necessary to choose exactly the PRNG which was used during the initial permutation. For each row i of the corresponding column j of the selected table R, the initial value is determined in accordance with formula (7).

It should be noted that an attacker to obtain an initial value (to form an initial permutation equivalent to that formed in the MA-1 algorithm) needs to know at least the values of two private keys (K_1^R , K_2^j), as well as the type of cryptographic hash function used and PRNG used. In addition, the number of permutations (how many times has a set of operations been performed to permute row characters) for each specific row of the protected table can be different. Brute force of only two 128-bit random numbers generated by cryptographically strong PRNG is a very resource-intensive task in order to realize it in a reasonable time. The number of combinations (excluding the definition of the hash function used, PRNG and the number of times performed operations when permutation of row characters) that need to be checked for the columns j of the R table is at least $\frac{1}{2}\cdot j \cdot 2^{128} \cdot 2^{128} = j \cdot 2^{255} \approx 5.8 \cdot j \cdot 10^{76}$ (half of the total amount of possible brute force tests; 50% chance).

2. The initial permutation is determined.

Thanks to the possibility of repeating a sequence of numbers formed by the PRNG from the same initial value, performing actions in accordance with the Fisher-Yates algorithm, we obtain the initial permutation (similar to that obtained when the implementation of the MA-1 algorithm): $\pi = (\pi(1), \pi(2), ..., \pi(n))$ or in other notation $\pi_{num} = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ a_1 & a_2 & a_3 & ... & a_n \end{pmatrix}$, as mapping the numbering onto the corresponding elements of a set.

3. The inverse to masking transformation is performed.

Having gotten the initial permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$, and, having the input string of the masked data $Y(i)$, in accordance with expression (14) you can determine the original (not masked) value of the row field $X(i)$, where $i \in \{1,...,n\}$.

The general scheme of the inverse masking algorithm (IMA-1) is represented below in pseudocode.

```
Inverse masking algorithm 1 (IMA-1)

Input: name_table, name_column, $K_3^i$, $Y$, $N_{\max}$
Output: inverse of masked value - $X$
    Decrypt($R^{secret}$[name _ table, name _ column]) $\rightarrow$ ($K_1^R$, $K_2^j$, PRNG, hash)
    $X_{R_{ij}}^0$ = hash($K_1^R$ + $K_2^j$ - $K_3^i$) mod ($N_{\max}$)
  for i = 1 to n
    $\pi_{num}[i] = i$     /*array preparation*/
  end for
switch(PRNG)
{
case 1: linear congruential generator (LCG)
case 2: built-in random number generator (package DBMS_RANDOM)
case 3: pseudo-random number generator Xorshift
...
}
  for i = n downto 1   /*getting initial permutation*/
   j=random_PRNG(1..i)
     swap($\pi_{num}[i]$, $\pi_{num}[j]$)
  end for

  for i = 1 to n       /*inverse of a permutation*/
    $X[\pi_{num}[i]] = Y(i)$
  end for
```

Example 4. Let the character string $'7P-K1\Pi'$ is stored in some field of the tuple i of the attribute j of table R as a result of masking (see Example 3). It is required to transform it to its original (before masking) state.

1. On the basis of the read values of the keys (two private and one public), knowledge of the hash function used and the PRNG, in accordance with the formula (7), the initial value for PRNG is formed. This step, naturally, is available only to an authorized user with the appropriate privileges and knowledge of the access key to the table R^{secret}.

2. Determination of the initial permutation (π_{num}) of the numbers of the elements for the character data string. After the corresponding initialization of the PRNG, which was used during the masking, actions are performed in accordance with the Fisher-Yates algorithm.

In this example, the initial permutation has the form (see Example 3):

$$\pi_{num} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 3 & 4 & 1 & 2 \end{pmatrix}.$$

3. Inverse to masking transformation.

In accordance with the formula (12) we have: $X[6]=Y(1)='7'$; $X[5]=Y(2)='1'$; $X[3]=Y(3)='\Pi'$; $X[4]=Y(4)='-'$; $X[1]=Y(5)='K'$; $X[2]=Y(6)='P'$.

Having ordered the indexes in ascending order, we get the string: `'КРП-17'`. The resulting value is equivalent to the masked value (see Example 3). That confirms the correctness of the algorithm IMA-1 (*Inverse masking algorithm 1*).

3. Comparative analysis of the quantitative and qualitative characteristics of the proposed algorithm and an encryption algorithm

Comparative analysis of quality characteristics

For a better representation and understanding of the corresponding data transformation (quality characteristics), Table 1 below shows the masking results of the character string `'КРП-17'` using the proposed algorithm using various PRNGs and the ordinary encryption algorithm AES-128. The following PRNGs were used: 1 – linear congruential generator (expression (6)); 2 – random number generator of built-in DBMS_RANDOM package for Oracle DBMS; 3 – pseudorandom number generator Xorshift with the (2^{128}-1) period. For clarity, the adjacent rows were selected (rows of the table with sequentially increasing numbering).

When encrypting character strings using the AES algorithm, the same mechanism for extracting and applying keys from the table R^{secret} was used as in the above proposed masking algorithm. Herewith the result of the formula (7) was used to determine the initialization vector in the encryption algorithm.

As we can see from Table 1, all the resulting strings are different from the original string. And this is important! But at that, the converted strings using the proposed approach retain the format and do not increase the dimension of the string, as in the case with the encryption algorithm. That in certain applications is critical and unacceptable. In this respect, the ordinary encryption algorithm concedes the proposed method. To eliminate this shortcoming, you can use encryption algorithms with format preservation (format-preserving encryption – FPE), but their implementation, for example, in Oracle, as noted in [14], involves significant processing.

Table 1

Character string masking results

ID	DATA_MASKING 1	DATA_MASKING 2	DATA_MASKING 3	DATA_ENCRYPT
18136	П-Р7К1	Р1КП7-	-71ПРК	F03E00DBD4D478A1D331F6CFAEC6A3DF
18137	1П7-КР	П71РК-	К1Р7-П	689783C6F95F526126C48FE6F11BEF13
18138	П-7КР1	РП17-К	КП1Р7-	C5B6BAFB17C10C528A7490B66E19A2B0
18139	П1К-Р7	Р-К7П1	-1К7ПР	E908DB627314C84B572486457ED0670A
18140	Р17П-К	КП7Р-1	Р7КП-1	4766A178B39586108DDB57F8ADBE5A5A
18141	17ПРК-	71-КРП	КП17Р-	6ECF167783564759E8D31753D59A53E3
18142	-КР7П1	1К-РП7	КР-П71	0F0D486A4F9CB189FA7092CB2CA4D47F
18143	1ПР7К-	71КП-Р	-РКП71	315690B8B84DA3460A4F4933DCA7648F
18144	Р7-К1П	КПР1-7	РК-71П	59DAA907A9CC38FF1A52C8AB9CD9DCC8
18145	7-РК1П	К7-1РП	1КР7П-	58F46C87949558D2C02CCBE3F368D09F
18146	КП7Р1-	П-17КР	Р1П7-К	DD3F2CC50EBD51FE991B8334EB50C347
18147	Р1П7-К	К-7ПР1	КР17-П	CD50A2ECA5D987419BB8C6794BDCBE71
18148	-К7ПР1	ПК71Р-	К-7ПР1	6549D15CC334245CDF7182CC2983C3C4
18149	-1РП7К	РК1-7П	7-РК1П	B37403B75997F2A535FA002239EDDEF5
18150	П71РК-	-П1КР7	К7П-1Р	D0C9509DDBD4ED6952E4434EA5FC13C2
18151	К7П-1Р	-К7Р1П	-Р7П1К	975AE4AEF567EDEEC9E40EC7F48A75D9
18152	Р17П-К	7-К1ПР	71П-КР	82084A1E15F818543D0845A535810F7D
18153	П1К-7Р	Р-К7П1	ПК1-Р7	9E7F647331CAF130D02C9CBCBB53968A
18154	П-Р17К	-К71ПР	Р-КП71	2C85EDFE7307EAF7EF11B7766B7416DF
18155	Р7ПК-1	К7-П1Р	ПР-71К	8FECDE765DE3773788C79040118B9A6B
18156	7-1КРП	КП17-Р	К-П1Р7	25F2F78484CD4872FC8FE6E0A35AB22F
18157	П1КР7-	КР-7П1	Р7-1ПК	CD62C80E53122CCA4AEF2472734EC138
18158	Р-7КП1	К7-П1Р	ПК-Р17	2DF64FC7780264AEF4E7D3BF49214ACF

Analyzing the results of Table 1, it can conclude that an attacker has little chance to determine from the available information that all these values are associated with the same character string 'КРП-17' (with the same real object). Although the number of permutations is not so large 6!=720, but in this particular case even statistical cryptanalysis is difficult, since, in fact, after the corresponding transformations, it may turn out that completely other and different objects of the modeled subject domain will be associated with the obtained transformed names. This significantly makes difficult the implementation of inference threat for an attacker. And, therefore, the proposed approach to hiding will not allow an attacker to obtain data in a reasonable time for him, access to which is directly closed to him.

Sometimes it is necessary to mask part of the field value of a table row, for example, for masking phone numbers, discount cards, bank cards, serial numbers of equipment and devices, car numbers, etc. In this case, the ordinary encryption method is not acceptable. The proposed approach, on the contrary, is suitable for solving this problem; for this, only some modification of the MA-1 algorithms (*Masking algorithm 1*) and IMA-1 (*Inverse masking algorithm 1*) represented above is necessary.

Schemes of the modified algorithms MA-2 and IMA-2 (differences from MA-1 and IMA-1 are highlighted in color) are represented below.

```
Masking algorithm (MA-2)

Input: name_table, name_column, $K_3^i$, $A$, $N_{\max}$, i_end, i_beg
Output: masked value of the data string – $A$
    Decrypt($R^{\sec ret}$[name _ table, name _ column]) → ($K_1^R$, $K_2^j$, PRNG, hash)
    $X^0_{R_{ij}} = hash(K_1^R + K_2^j - K_3^i) \bmod (N_{\max})$
switch(PRNG)
{
case 1: linear congruential generator (LCG)
case 2: built-in random number generator (package DBMS_RANDOM)
case 3: pseudo-random number generator Xorshift
...
}
 for i = n-i_end downto i_beg
  j=random_PRNG(i_beg..i) /*a random number is generated in the range [i_beg,i]*/
  swap($A[i]$,$A[j]$)      /*exchange*/
 end for
```

Inverse masking algorithm (IMA-2)

```
Input: name_table, name_column, $K_3^i$, $Y$, $N_{\max}$, i_end, i_beg
Output: inverse of masked value – $X$
    Decrypt($R^{secret}$[name _ table, name _ column]) $\rightarrow$ ($K_1^R$, $K_2^j$, PRNG, hash)
    $X^0_{R_{ij}} = hash(K_1^R + K_2^j - K_3^i) \bmod (N_{\max})$
  for i = 1 to n
    $\pi_{num}[i] = i$      /*array preparation*/
  end for
switch(PRNG)
{
case 1: linear congruential generator (LCG)
case 2: built-in random number generator (package DBMS_RANDOM)
case 3: pseudo-random number generator Xorshift
...
}
  for i = n-i_end downto i_beg   /*getting initial permutation*/
   j=random_PRNG(i_beg..i)
     swap($\pi_{num}[i]$, $\pi_{num}[j]$)
  end for
  for i = 1 to n       /*inverse of a permutation*/
    $X[\pi_{num}[i]] = Y(i)$
  end for
```

Where the parameters i_beg, i_end define the boundaries of the transformation interval, namely: i_beg – from which position from the beginning of the string the permutation should be performed; i_end – up to which position from the end of the string permutation should be performed.

In addition, to mask, for example, bank cards after the corresponding permutation procedure (see algorithms MA-2, IMA-2), it is also necessary to calculate the last check digit using the Luhn algorithm [32], in accordance with the ISO/IEC 7812 standard.

Tables 2, 3, 4 show the masking results using the proposed algorithm (with different PRNG: 1 – LCG – the result in the DATA_MASKING 1 column; 2 – built-in PRNG – result in the DATA_MASKING 2 column; 3 – PRNG Xorshift – result in the DATA_MASKING 3 column) of data on card numbers stored in the corresponding sequences of adjacent rows of some table, namely, on Visa bank card – number '4454102135347018', Master Card bank card – number '5167135104128196', American Express payment system card – number '378282246310005', with preservation of the card type.

If you need to mask some table fields that store, for example, important text documents (data in one of the text formats: both in TXT format itself and in other, for example, such specialized formats as INI, HTML, XML, TeX, JSON, LOG, source texts of programming languages and others for which it serves as the basis), which are significantly larger in volume (length) the character strings discussed above, then this task can also be effectively solved using the proposed method.

Table 2

The masking results of Visa bank card number

ID	DATA_MASKING 1	DATA_MASKING 2	DATA_MASKING 3
35359	4411443305170521	4401235043741510	4400451172135348
35360	4413021740415533	4437312451510402	4413535021740418
35361	4434230511514078	4451124347013504	4433412071450155
35362	4441031715425306	4454317254110308	4445701534021132
35363	4410203415715438	4401473215035144	4401112347355406
35364	4471310435041525	4431424310075158	4403441103715257
35365	4442031413755102	4453113442705013	4450104341275318
35366	4434032105174511	4470310431215454	4432740405113516
35367	4442714501531309	4413274510410354	4401351344017250
35368	4411233574040152	4401354023541715	4470315450124131

Table 3

Masking results the number of Master Card bank card

ID	DATA_MASKING 1	DATA_MASKING 2	DATA_MASKING 3
35369	5174311028615912	5121381165147098	5173192416150188
35370	5193751482161100	5116471112850931	5111091716482355
35371	5193711261150849	5173051612198419	5131072681541199
35372	5174513918161028	5124311678019515	5171350492111688
35373	5110341258911670	5176018951231413	5162081179141356
35374	5111501217936485	5114811259061374	5193110548211674
35375	5105311179468123	5168143975012118	5143196250811174
35376	5113517462018918	5129713164051180	5171591241013689
35377	5101512147198361	5164113082517196	5192304161118750
35378	5141351291186076	5118107163219455	5171084112195363

Table 4

Masking results the card number
of the American Express payment system

ID	DATA_MASKING 1	DATA_MASKING 2	DATA_MASKING 3
35379	372800263182043	370020288642317	371288240036022
35380	378022631084025	370816438022205	372206321048087
35381	372006822180433	370010822348265	372142063008825
35382	373206140282809	372382004682011	376482020308215
35383	372860104823205	371242806280300	372200346821081
35384	370280463102285	376182320802400	372302880420161
35385	372843260180024	376048820221039	372042318028604
35386	378231264800025	376082412008322	371862302802040
35387	372012648080237	370102628084325	373802460082124
35388	372020023868415	378023801262045	372822460183004

The proposed method, as it was noted above, can be applied to the masking of the fields of tables that store data not only of the character string type (*character*, *character varying*) and numeric types (*integer*, *number*), but also data of the BLOB, CLOB type that allow you to store in the database, for example, documents of such formats as: DOC, DOCX, RTF, XLS, XLSX, XPS, PDF, PPT, BMP, GIF, TIF, MP3, AVI, etc. But this important question, which has interesting features of practical implementation, in this paper, in view of its limited scope, will not be considered. This is the subject of a separate article.

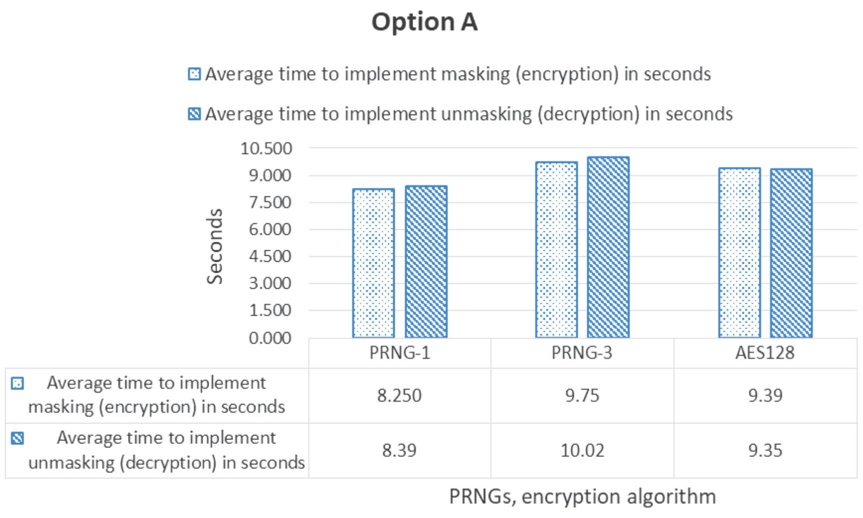
Quantitative characteristics of comparative analysis

The results below were obtained with the appropriate data transformations of the database tables. The database was implemented on the Oracle 12.2 c DBMS platform installed on the Windows 10 (x64) operating system on various computers.

Option A. Computer with Intel (R) Core (TM) 2 Duo CPU 2.16 GHz, RAM 4 GB, HDD 320 GB was used.

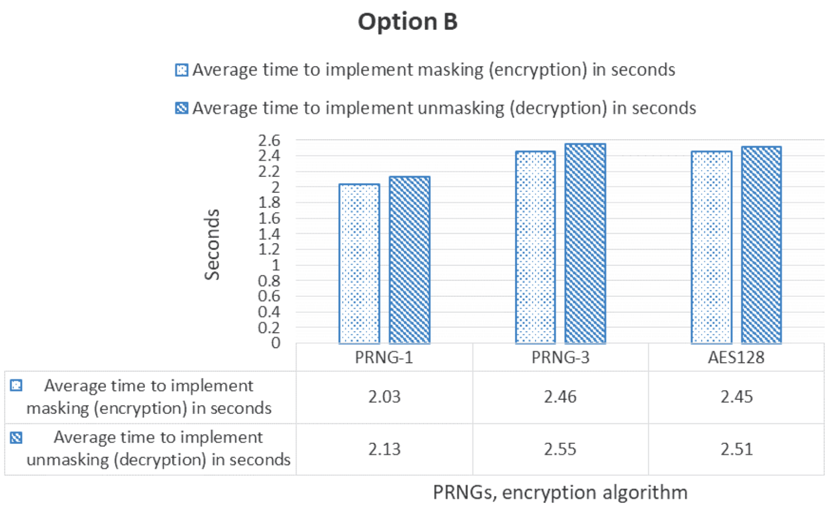
Option B. Computer with Intel(R) Core (TM) i3-7100 CPU 3.90 GHz, RAM 4 GB, HDD 500 GB was used.

About 18,000 data rows of some *varchar2*(255) column of a table of real database were subjected to transformation (masking, encryption) with recording to the database. Fig. 1, 2 shows the results of the average times (in seconds) of masking and unmasking, encrypting and decrypting character strings with writing to the database all 18,000 data rows using the proposed permutation algorithm (were used linear congruential PRNG (PRNG-1) and pseudo-random number generator Xorshift (PRNN-3)) and the ordinary AES-128 encryption algorithm for options A and B respectively.



	PRNG-1	PRNG-3	AES128
▨ Average time to implement masking (encryption) in seconds	8.250	9.75	9.39
▨ Average time to implement unmasking (decryption) in seconds	8.39	10.02	9.35

Fig. 1. Average time to transform character strings (option A)



	PRNG-1	PRNG-3	AES128
▨ Average time to implement masking (encryption) in seconds	2.03	2.46	2.45
▨ Average time to implement unmasking (decryption) in seconds	2.13	2.55	2.51

Fig. 2. Average time to transform character strings (option B)

As follows from the conducted research, the usage of the ordinary encryption algorithm loses to the proposed method not only in qualitative characteristics, namely, changes in the presentation format, but also quantitative due to the increase in the length of the stored data and the transformation time for hiding data. If, when using Xorshift PRNG, the compared times of corresponding transformations for both methods are almost the same, then already compared to the permutation algorithm, when using the linear congruential PRNG, the implementation of the encryption algorithm loses to it (11-12)% for option A and (15-17)% for option B, respectively. (Note. The current implementation of Xorshift PRNG in PL/SQL is currently not an optimized implementation due to this language does not support hardware implementation of logical shifts, XOR operations, and therefore these ones are solved algebraically).

4. Usage restrictions of the proposed method and features of its implementation

When the length of the string is less than three characters or, if all the characters are the same, masking using the proposed method is inexpedient. In this case, it is advisable to use:

– or a substitution method with elements of the proposed approach, namely, by creating a mapping table with a random selection from it of characters or their combinations, using the available PRNGs and the shuffling mechanism of the proposed permutation method for this purpose (with the ability to subsequently restore the sequence, required for the inverse transformation, of the generated numbers from the initial value $X^0_{R_{ij}}$);

– or a method of masking data based on the calculation of operations modulo, by transformation of a short string to a numerical form, if it includes letter characters, special characters and digits, or by simply converting a string of digits to a number using standard conversion functions (for example, for Oracle, this is the `TO_NUMBER` function).

The privacy of the masking keys depends on where the keys are stored and who has access to them. The proposed solution uses three keys (K^R_1, K^j_2, K^i_3): two are private (generated cryptographically strong PRNG) and one is public. All private keys are encrypted using the AES-256 algorithm and stored in a special database table R^{secret}. The values of these keys are never shown and not known either to the database administrator (if he does not combine the functions of the security administrator), or to any other user. An authorized user, with appropriate privileges, which will provide the correct key in an open session to decrypt the row of the special table, has the mediate access through the corresponding middleware to the private keys K^R_1, K^j_2. At that, the value of this key is not shown anywhere in the clear, it cannot be traced even through the available means of documenting executed queries (a historical command log). It is extracted in a certain way by the special software of the DBMS server from the key container file, which an authenticated user with the appropriate privileges must provide during the session opening.

Actions in case of encryption key compromise of the table R^{secret}

If the key with which the data (the private K^R_1, K^j_2 keys and some other information) of the R^{secret} table is encrypted, is compromised, it can be replaced with a new generated one using specially developed software (cryptographically strong PRNG is used for this). After that, the data of the entire table R^{secret} is encrypted with the new key. At that, the private K^R_1, K^j_2 keys are not shown anywhere explicitly. Then, using special software, for legitimate users key container files for decrypting the data of the table R^{secret} are formed.

Actions in case of compromise of separate private K^R_1, K^j_2 keys

When the compromise of separate private keys from sets of K^R_1, K^j_2 they can also be generated using cryptographically strong PRNG and special developed software similarly the decryption key of the R^{secret} table. After that, it remains only to replace the compromised private keys with new ones, pre-encrypting them with an AES-256 algorithm. At that, the newly generated private keys are also not shown anywhere explicitly.

Conclusions

1. Analyzing the best practices of hiding information from leading vendors, a new approach to data hiding was proposed, making it difficult for an attacker to implement the inference threat. This approach was based on the principles of random permutation of the elements of a specific field of the corresponding column (attribute) of the row (tuple) of the production database table data and dynamic masking. It differs from the existing ones in that a preliminary physical change of sensitive data is made in the production database, and a user who has the appropriate rights can cancel these changes if it is necessary. That is, the corresponding user can restore all data changes made during the masking procedure to their original state.

2. The authenticated user in the proposed solution gets access to sensitive data due to the ability to transform (rewrite) the query "on the fly", and an attacker, even using complex as well as sequences of simple logically related queries, is limited in implementing the threat of inference during an acceptable time for him (due to the fact that only the masked data is available to him).

3. It is possible to mask both an entire value of the field of the table row and its part using the proposed solution. This is relevant on the one hand for masking such data as phone numbers, discount cards, bank cards, serial numbers of equipments and devices, car numbers, etc. and on the other hand, it allows reducing the number of operations for transformation large binary objects, and, consequently, the implementation time, without losing the effectiveness of the masking procedure.

4. Studies have shown that the use of the ordinary encryption algorithm loses to the proposed method not only by qualitative characteristics (primarily due to a change in the data representation format and the inability to transform part of the row field value), but also by quantitative characteristics (due to an increase in the length of the stored data and the transformation time for hiding data (by about (10-17)%)).

5. A distinctive feature of the proposed solution is the approach to the process of data shuffling, namely, shuffling data value elements within the demanded row field. At that, the basic operations of the proposed method can also be used for vertical shuffling – permutation of values within the column of the selected table.

6. The proposed approach to data masking can be used in both production and non-production databases, expanding the possibilities of so-called static data masking.

References

1. Sandhu R. S., Jajodia S. (1993) Data and database security and controls // Handbook of information security management. Auerbach Publishers, pp. 481-499.

2. Kulkarni S., Urolagin S. (2012) Review of attacks on databases and database security techniques // International Journal of Emerging Technology and Advanced Engineering, **2**(11), pp. 2250-2459.

3. Top ten database security threats. The most significant risks of 2015 and how to mitigate them, Imperva Whitepaper, 2015 [Electronic resource]. Access mode : https://informationsecurity.report/Resources/Whitepapers/e763d022-6ee4-4215-9efd-1896b0d9c381_wp_topten_database_threats%20imperva.pdf, last accessed 2019/09/01.

4. Rohilla S., Mittal P. K. (2013) Database Security: Threads and Challenges // International Journal of Advanced Research in Computer Science and Software Engineering, **3**(5), pp. 810–813.

5. Top 5 Database Security Threats, Imperva Whitepaper, 2016 [Electronic resource]. Access mode: https://www.imperva.com/docs/gated/WP_Top_5_Database_Security_Threats.pdf, last accessed 2019/09/01.

6. Infowatch. Analytics. Digests and Reviews. Over 12 years, more than 30 billion personal data records have leaked [Electronic resource]. Access mode : https://www.infowatch.ru/analytics/digest/15281, last accessed 2019/09/01. (in Russian)

7. Global research on confidential information leaks in 2018, Analytical center InfoWatch, 2019 [Electronic resource]. Access mode: https://www.infowatch.ru/sites/default/files/report/analytics/russ/InfoWatch_Global_Report_2018_year.pdf?rel=1, last accessed 2019/09/01. (in Russian)

8. Pfleeger C. P., Pfleeger S. L., Margulies J. (2015) Security in Computing (Fifth Edition). Prentice Hall, 944 p.

9. Wang L., Jajodia S. (2008) Security in Data Warehouses and OLAP systems // Handbook of Database Security, Springer, Boston, MA, pp. 191-212.

10.Zavgorodniy V. I. (2001) Complex information protection in computer systems. M. : Logos, PBOYUL N.A. Egorov, 264 p. (in Russian).

11.Mayer-Schonberger V., Cukier K. (2013) Big Data: A Revolution That Will Transform How We Live, Work and Think. Canada, Eamon Dolan/Houghton Mifflin Harcourt, 242 p.

12.Gartner IT Glossary [Electronic resource]. Access mode : https://www.gartner.com/it-glossary/dynamic-data-masking-ddm, last accessed 2019/09/01.

13.A Net 2000 Ltd. White Paper. Data masking: What You Need to Know Before You Begin [Electronic resource]. Access mode : http://www.datamasker.com/DataMasking_WhatYouNeedToKnow.pdf, last accessed 2019/09/01.

14.Data Masking and Subsetting Guide [Electronic resource]. Access mode : https://docs.oracle.com/en/database/oracle/oracle-database/12.2/dmksb/introduction-to-oracle-data-maksing-and-subsetting.html#GUID-24B241AF-F77F-46ED-BEAE-3919BF1BBD80, last accessed 2019/09/01.

15.Santos R. J., Bernardino J., Vieira M. A. (2011) Data masking technique for data warehouses // Proceedings of the 15th Symposium on International Database Engineering & Applications, ACM, pp. 61-69.

16.Yesin V. I. (2018) Invariant to subject domains database schema and its distinctive features // Radiotekhnika : 193, pp. 133-142 (in Russian)

17.Yesin V. I., Yesina M. V., Rassomakhin S. G., Karpinski M. (2018) Ensuring Database Security with the Universal Basis of Relations // Saeed K., Homenda W. (eds) Computer Information Systems and Industrial Management. CISIM 2018. Lecture Notes in Computer Science, **11127**, Springer, Cham, Chapter 42, pp. 510-522.

18.Tirosh A., Meunier M. (2015) Magic Quadrant for Data Masking Technology, Worldwide Published: 22 December 2015 ID: G00273093 [Electronic resource]. Access mode : https://docplayer.net/12460751-Magic-quadrant-for-data-masking-technology-worldwide.html, last accessed 2019/09/01.

19.Dworkin M. (2019) Recommendation for block cipher modes of operation. Methods for format-preserving encryption // Draft NIST Special Publication, № 800-38G Revision 1 [Electronic resource]. Access mode : https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38Gr1-draft.pdf, last accessed 2019/09/01.

20.Schneier B. (1996) Applied cryptography: protocols, algorithms, and source code in C (2nd edition), John Wiley & Sons, Inc., 758 p.

21.Mao W. (2003) Modern Cryptography: Theory and Practice. Prentice Hall PTR, 707 p.

22.Shannon C. (1949) Communication Theory of Secrecy Systems // Bell System Technical Journal, **28**(4), pp. 656-715.

23.Ferguson N., Schneier B. (2003) Practical cryptography. New York, Wiley, 432 p.

24.Knuth, D. E., (1997) The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.), Addison-Wesley Professional, 650 p.

25.Fisher R. A, Yates F. (1948) Statistical Tables for Biological, Agricultural and Medical Research (3rd Edition). Edinburgh and London, **13**(3), pp.26-27.

26.Durstenfeld R. (1964) Algorithm 235: Random permutation // Communications of the ACM, **7**(7), pp. 420.

27.Bacher A., Bodini O., Hollender,A., Lumbroso J., (2018) Merge Shuffle: a very fast, parallel random permutation algorithm // Proceedings of the 11th International Conference on Random and Exhaustive Generation of Combinatorial Structures Athens, Greece, June 18-20, CEUR-WS.org/Vol-2113, pp. 43-52.

28.Knuth D. E. (1997) The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd ed.). Addison-Wesley, Reading, MA, 762 p.

29.Press W. H., Flannery B. P., Teukolsky S. A., Vetterling W. T. (1992) Numerical Recipes in C: The Art of Scientific Computing (Second Edition). Cambridge University Press, 994 p.

30.Marsaglia G. (2003) Xorshift rngs // Journal of Statistical Software, **8**(14), pp. 1-6.

31.Press W. H., Teukolsky S. A., Vetterling W. T. (2007) Flannery B. P. Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York, Cambridge University Press, 1235 p.

32.Patent No. 2,950,048, United States, Computer for Verifying Numbers / H. P. Luhn, Armonk, N.Y., assignor to International Business Machines Corporation, New York, N.Y., a corporation of New York. Ser. No. 402,491; Aug. 23, 1960.

33.Bacher A., Bodini O., Hwang H. K., Tsai T. H. (2017) Generating random permutations by coin tossing: Classical algorithms, new analysis, and modern implementation, ACM Transactions on Algorithms, **13**(2), 43 p.

34.Ravikumar G. K., Justus R., Ravindra S. H., Manjunath T. N., Archana R. A. (2011) Experimental study of various data masking techniques with random replacement using data volume // International Journal of Computer Science and Information Security, **9**(8), pp.154-158.

Kharkiv National V.N. Karazin University *Received 05.09.2019*