NJS: Database Protection Algorithm

Non-deterministic and post-quantum cryptography

Edimar Veríssimo da Silva[1]
[1]ETEP, São José dos Campos, São Paulo, Brazil
yugi386@yahoo.com.br

Abstract: NJS is a cryptographic protection algorithm for relational databases with non-deterministic symmetric encryption, making it possible to search data with almost the same speed as a clear text search (depending on the parameterization). The algorithm has the characteristic of performing a fast encryption on the data and a slightly slower decryption that is only performed on the client workstation. The entire process of searching, changing, adding and deleting data is performed on the server with the encrypted data. The NJS cipher is not a form of homomorphic encryption, but it can replace it with some search limitations. One advantage is the fact that noise added to the message does not interfere with its decryption, regardless of the number of operations performed on each record in a database table.

Keywords: symmetric cryptography, database protection, non-deterministic cryptography.

1 Introduction

In this paper we propose a new symmetric non-deterministic algorithm: NJS[1]. The algorithm attempts to realize an encryption with security level equivalent to RSA from a simple structure. Such a goal is achieved without using the factorization problem or the discrete logarithm problem. Therefore, without compromising the algorithm in the scenario where quantum computers are a reality. The RSA algorithm has been attacked in several ways [2][3][4] but most attacks start from the knowledge of the modulus N and the exponent used in encryption which is usually fixed.

In section 2 we present the structure of the NJS algorithm briefly. Section 3 presents the rationale of the algorithm, the proposed encryption of the data, the structure of the cipher key, and the interaction between numerical operations of different algebraic groups in arbitrary block sizes without the need for padding. Section 4 presents operational details such as the application of noise levels, examples of non-deterministic encryption, efficient

[1] NJS is the acronym for Noêmia Josefina da Silva.

use of hash functions to prevent decryption errors, search techniques and security in the use of the SQL language [5]. Section 5 makes an interesting comparison (respecting the difference between symmetric and asymmetric encryption) between the RSA system and the NJS algorithm. In this section we show that the use of the fixed exponent $2^{16}+1$ in RSA reduces the exponentiation of long integers to a single modular multiplication. Without delving into the question of a possible vulnerability of RSA, we have reasoned about the NJS algorithm, showing the use of at least 9 algebraic operations against a single operation in RSA. Can we at least conjecture that the level of security is equivalent? In section 6 we present the structure of the source code files that are available in appendix III. Appendix I presents an example of a system key and Appendix II shows a table of 30 records with dummy data in its native encrypted form. Such records are intended to give an objective idea of the increase in data volume in the encrypted records. In the conclusion we present some data thoughtfully.

The remainder of the introduction deals with the limitations of this work and our contributions.

1.1 Limitations of this paper

The NJS algorithm cannot fully replace a homomorphic encryption [6]. However, it can allow a group of users sharing the same key to access a cloud-hosted server [7] in encrypted form. The algorithm also allows searches on encrypted data, changes, deletions and additions of new records (at a rather high but feasible cost).

NJS is a highly parameterizable algorithm. In our example code we work with blocks of 480 bits and modules of at least 512 bits. These values can be adjusted almost without limit, and this limit only comes up against the speed at which the data can be processed. It has not yet been possible to establish a formal balance between security and execution speed for the NJS algorithm.

In order to make queries and data changes feasible, it is necessary to employ simpler primary keys, which can minimize the cost of accessing the encrypted data. In general, it is not possible to search for information that is smaller than the proposed block (480 bits), unless the block matches the entire field of a given record.

Another disadvantage of the proposed scheme is the use of "fake queries" to protect existing data relationships from attackers who might monitor SQL operations on the database. The use of "fake queries" has the advantage of confusing those who are monitoring the SQL operations, however the server must be able to handle a much larger number of requests to perform tasks that in a decrypted database would be much less costly.

The prototype executable that can be obtained from the source code should be further explored in the future. The ciphering and decryption results were encouraging, both when considering the performance of the algorithm and the non-deterministic characteristic of the cipher. Despite this, it was not possible to formalize these observations through mathematical proofs.

No major cryptoanalytic studies have been performed to date. However, in Section 6 we present a comparison of the NJS algorithm with the RSA algorithm and make a credible argument for the feasibility of this proposal.

Another negative point, but not a deterrent to using the NJS algorithm, is the increase in data volume. Using parameters of the same order of magnitude as those illustrated in the source code, it is possible to see an increase in the database of about 6.7 times compared to the same database unencrypted.

1.2 Our contribution

The NJS algorithm can encrypt a relational type database [8] with numerous tables using a non-deterministic cipher that is supposed to have a security level similar to RSA. But unlike RSA, the NJS algorithm will not suffer from quantum attacks because it does not use factorization or the discrete logarithm problem to implement its security. Its symmetric structure allows it to hide from the attacker the main pillars that he could use to carry out his attacks.

The algorithm clearly tries to benefit from employing numerical operations involving different algebraic groups, namely addition, multiplication, and the XOR operation. Such a strategy was successfully employed in the symmetric IDEA algorithm [1] (International Data Encryption Algorithm). NJS innovates this proposal by allowing the application of such operations with data blocks of 480 bits (proposed size) or even larger, without any limitations. The IDEA algorithm employs 64-bit blocks, with 16-bit words (using the prime $2^{16}+1$ as a modulus for multiplication). This block size is now obsolete.

With the proposed parameterization in our source code the NJS algorithm can generate up to 2056 combinations of cryptograms for the same block of text using the same key. The proposed text block size is 480 bits (60 characters). This value seems reasonable to us to avoid too many repetitions in the data while protecting, within a certain limit, the primary key of the table.

The NJS algorithm can also be used to encrypt simple data (not in a relational database structure). In this case it is possible to employ noise levels [9] of the magnitude of 256 bits (or larger) thus avoiding any kind of pattern repetition in the data, even if it is a concatenation of a single byte repeated billions of times. This is a huge advantage from the operational point of view of a non-deterministic cipher. The fact is further justified because noise removal is almost instantaneous in a legitimate decryption and very difficult in the absence of the correct cipher key. However such a practice prevents searching encrypted data due to the impossibility of mapping all possible search variants. If the encryption is of a simple file it can be advantageous from an operational point of view.

Finally, the data compression algorithm used in NJS generates a result that contains only printable characters, easy to save in files with a simple structure such as TXT or CSV.

2 Algorithm Structure

The NJS algorithm behaves like a non-deterministic cipher, i.e., using a fixed key **(K)** it is always possible to obtain a different cipher **(C)** using the same cleartext **(P)**.

The algorithm works with random data mixed with the clear message and the cipher key. The random data used in the encryption process can come from any source, it just has to be concentrated in a parameterized range of values to allow correct decryption.

The cost of encryption is minimal but the cost of decrypting a message is somewhat high, which translates into one more security parameter for the algorithm. If to legitimately decipher a cryptogram it is necessary to process more information then it is also very likely that an attacker, trying to break the code without access to the keys, will have bigger problems than the legitimate decryptor.

The structure of the algorithm is very simple and works by using different algebraic groups to encrypt a message. This strategy has precedents in cryptography and was successfully employed in the IDEA algorithm. A key Sbox is used before processing to make it difficult to identify the starting point of the data. This Sbox is used with a function based on its inverse Sbox, which makes it possible to differentiate the keys used for each field of a database record.

The use of the NJS algorithm is advantageous when we have an encrypted database that needs to be accessed by several legitimate clients. The database can be hosted on an insecure server but the decryption is only performed on the client computer. Of course, the data traffic between the client and the server can be strengthened from a security point of view by using a post-quantum asymmetric system, but the algorithm seems to be enabled to work without this extra security barrier if necessary.

3 Logical Rationale

Let **M** be a message (clear text) represented by the vector **Y** = [b_1, b_2, b_3, b_4, b_5, ..., b_n], where each $\mathbf{b_x}$ value is contained in the range [0, 255]. The vector **Y** represents a field of a database record or a fraction of such a field. Thus a record can have **X** number of fields. A table **T** may contain millions of **X** records, and a database **B** may contain tens, hundreds, or even thousands of **T** tables.

Table fields:

$X_1 = Y_{11}[b_1, b_2, b_3, ..., b_n] + Y_{12}[b_1, b_2, b_3, ..., b_n] + ... + Y_{1n}[b_1, b_2, b_3, ..., b_n]$
$X_2 = Y_{21}[b_1, b_2, b_3, ..., b_n] + ... + Y_{2n}[b_1, b_2, b_3, ..., b_n]$
$X_3 = Y_{31}[b_1, b_2, b_3, ..., b_n]$

Table records:

$$R_1 = X_1 + X_2 + ... + X_n$$

Full table:

$$T_1 = R_1 + R_2 + R_3 + ... + R_n$$

Database:

$$B = T_1 + T_2 + T_3 + ... + T_n$$

A field **X** of an **R**-record of a **T**-table is separated into blocks of size **Y** (512 bits or a value close to it). Each block **Y** is encoded as a long unsigned integer (BigUint). This long integer is modified by at least 3 algebraic operations: addition, multiplication and XOR. The order of execution of these operations does not depend on the cipher key and can be defined by a pseudo-random number generator (or even a truly random generator). The operations are modular for some prime **P** of size greater than 512 bits.

$$Y_1 = (257^3 * (b_3+1)) + (257^2 * (b_2+1)) + (257^1 * (b_1+1)) + (257^0 * (b_0+1))$$

This is an example of a 32-bit block. To avoid filling null bytes to complete a block (which for a database record would be hardly applicable in practice) we operate in base 257 instead of base 256. So we don't have any message represented by the value 0, in any of its bytes. This way we can work with incomplete blocks. Note also that it is necessary to add one unit to the value of each byte in the encryption and then subtract it in the decryption.

The word limits will be extended. In the case of 32-bit words we would have a limit of 4294967295 ($256^4 - 1$) and in the case of the NJS algorithm we would have 4362470400 ($257^4 - 1$). This implies that if we use blocks of 512 bits for processing the fields ($256^{64} - 1$) we will have to have a prime modulus **P** that is larger than ($257^{64} - 1$). This implies an increase in data volume of about 1.28 times.

The system key is represented by a vector with 48 values of at least 512 bits each, plus an Sbox (a vector that represents a permutation between all 256 8-bit values) and of course the Sbox^{-1} (inverse Sbox).

If a field in a record has the structure $R_1 = Y_1[b_1, b_2, b_3, ..., b_n]$, where b_n represents a value in the range [1, 256], then the corresponding cryptogram has the following structure:

$$R_{[0]} = [(r \; * \; 257^{59}) + ((S[b_1] \text{ xor } F_i)* \; 257^{58}) + (S[b_2] \text{ xor } F_i)* \; 257^{57}) + ... + \; (S[b_n] \text{ xor } F_i)* \; 257^{0})]$$

$$R_{[1]} = (R_0 \text{ ADD } K_{11}) \bmod M_1$$
$$R_{[2]} = \; (R_1 \text{ MUL } K_{12}) \bmod M_1$$
$$R_{[3]} = \; R_2 \text{ XOR } K_{13}$$

Here we see the 4 states of the register $\mathbf{R_{[x]}}$, where $\mathbf{R_{[0]}}$ is its initialization, which is done by passing through the Sbox **S** and adding a noise value **r**. Besides this a F_i value (field index) is added, which is directly linked to the field number of the register associated with the Sbox $\mathbf{S^{-1}}$ (inverse Sbox).

Then an addition (ADD) and a multiplication (MUL) are performed in module M (Keys K_{11} and K_{12}). Finally an XOR operation is performed with key K_{13}. This procedure is executed 3, 6, 9 or 12 times. The variable that defines these values comes from a pseudo-random number generator and has no relation to the cipher key.

The complete cipher looks like this:

$$R_{[0]} = [(r \; * \; 257^{59}) + ((S[b_1] \text{ xor } F_i)* \; 257^{58}) + (S[b_2] \text{ xor } F_i)* \; 257^{57}) + ... + \; (S[b_n] \text{ xor } F_i)* \; 257^{0})]$$

$R_{[1]} = (R_0 \text{ ADD } K_{XA}) \bmod M_N$ $R_{[2]} = \; (R_1 \text{ MUL } K_{XB}) \bmod M_N$ $R_{[3]} = \; R_2 \text{ XOR } K_{XC}$	or	$R_{[1]} = \; (R_0 \text{ MUL } K_{XA}) \bmod M_N$ $R_{[2]} = (R_1 \text{ ADD } K_{XB}) \bmod M_N$ $R_{[3]} = \; R_2 \text{ XOR } K_{XC}$	Number of rounds: 3, 6, 9 ou 12.

The presented structure allows you to choose from 8 possible variations to encrypt each database field. The number of repetitions can represent a key. However, for operational ease, fixed values were set for the number of rounds (3, 6, 9 or 12). Such values were chosen to thwart side-channel attacks [10]. These attacks analyze the algorithm's processing time. As an example we could say that it would not be possible to easily distinguish two encrypted records with 6 rounds from another encrypted record with 12 rounds, or even 4 encrypted records with 3 rounds each.

The set of keys in the algorithm can be defined thus:

Table 1: NJS system keys

Round	**Encryption / Decryption Operations**	**Prime module**
1	K_{11} (MUL or ADD)	M_1
	K_{12} (MUL or ADD)	M_1
	K_{13} (XOR)	
2	K_{21} (MUL or ADD)	M_2
	K_{22} (MUL or ADD)	M_2
	K_{23} (XOR)	

3	K_{31} (MUL or ADD)	M_3
	K_{32} (MUL or ADD)	M_3
	K_{33} (XOR)	
4	K_{41} (MUL or ADD)	M_4
	K_{42} (MUL or ADD)	M_4
	K_{43} (XOR)	
5	K_{51} (MUL or ADD)	M_5
	K_{52} (MUL or ADD)	M_5
	K_{53} (XOR)	
6	K_{61} (MUL or ADD)	M_6
	K_{62} (MUL or ADD)	M_6
	K_{63} (XOR)	
7	K_{71} (MUL or ADD)	M_7
	K_{72} (MUL or ADD)	M_7
	K_{73} (XOR)	
8	K_{81} (MUL or ADD)	M_8
	K_{82} (MUL or ADD)	M_8
	K_{83} (XOR)	
9	K_{91} (MUL or ADD)	M_9
	K_{92} (MUL or ADD)	M_9
	K_{93} (XOR)	
10	K_{A1} (MUL or ADD)	M_A
	K_{A2} (MUL or ADD)	M_A
	K_{A3} (XOR)	
11	K_{B1} (MUL or ADD)	M_B
	K_{B2} (MUL or ADD)	M_B
	K_{B3} (XOR)	
12	K_{C1} (MUL or ADD)	M_C
	K_{C2} (MUL or ADD)	M_C
	K_{C3} (XOR)	

An important note is to establish the relationship between the prime modulus M_x, and the keys used K_{x1}, K_{x1} and K_{x3}.

1. M_x must be a prime number of at least 512 bits: It is important that this value cannot be attacked by exhaustive search. There are about $1.88 * 10^{151}$ prime numbers of 512 bits.

$$\frac{2^{512}}{\log(2^{512})} - \frac{2^{511}}{\log(2^{511})} = 1.88 \times 10^{151}$$

2. K_{x1} and K_{x2} can be any number of 512 bits smaller than M_x. Since M_x is a prime number there will always be the multiplicative inverse K_{x1}^{-1} or K_{x2}^{-1} that will allow the data to be deciphered. The inverse of the sum is given by $M_x - K_{x1}$ or $M_x - K_{x2}$, all these values being unsigned.

3. The key K_{x3} is used in an XOR operation. Thus, in decryption the value of K_{x3}^{-1} matches the value of K_{x3}. An important remark should be made here: K_{x3} need not necessarily be smaller than M_x since in XOR operations we do not use a modular operation. However, the value M_{x+1} must be larger in number of bits than M_x and K_{x3} for decryption to be possible. The IDEA system takes advantage of the fact that $2^{16} + 1$ is a prime number. In the case of the NJS algorithm we do not have a limit of the cipher block by the number of bits. So for decryption to be possible, if M_x has 512 bits and K_{x3} has 512 bits then M_{x+1} must contain at least 513 bits. This is true for the complete string of M_x and K_{x3} values.

The full key can be understood as a set of 48 values of approximately 512 bits plus one Sbox[2].

Table 2: Total system key size

Keys	Size
K_x	3 * 12 * 512 (bits)
M_x	512 + 513 + 514 + 515 + 516 + 517 + 518 + 519 + 520 + 521 + 522 + 523 (bits)
Sbox	256! ≈ 2^{1684} bits
Total	26326 bits

A brute force attack is very complicated because none of these values are exposed (K_x, M_x and Sbox) and all the keys are used only in the client environment. All operations performed on the database are done with the encrypted values. To have a higher level of protection we recommend the use of a different Sbox for each database table. One last remark is to point out that it is possible to work with larger key numbers (1024 bits or more) but this will increase the size of the database considerably.

4 Operational details

In order to operate the NJS algorithm it is necessary to add an extra field to each database table. This field must have as content a hash value. For the NJS algorithm we adopted SHA3-224 but any strong hash function could have

[2] A complete example of a key can be seen in Annex I.

been used. In this case we opted for a smaller hash so that the impact on the database volume would be as small as possible. The hash value for this field is given by concatenating the contents of all fields in the record.

A record in a database could be represented as follows:

Table 3: Example of a record

Field	Value
Name	Tairis Geissler Behrends Delazare Groskopf
Document	180651975-23
Phone	(64)98073-4980
Date of Sale	11/12/2017
Sale Value	1377.23
Hash	dcc6efc903fbd40ae78c31ace6e39b39ea0d8e19e15673822eb27e8c

The SHA3-224 hash value is required to allow decryption of the data. Since the cipher is non-deterministic in nature then a parameter is needed to differentiate a "correct decryption" from an "incorrect decryption". Examples of possible encryptions with the same key:

Table 4: Ciphered record (version 1)

Field	Value
Name	)TI20v/40Bt+uCkDg/ t}<*800=+0Jryb60OX0SU`zmYk40cU60R:80cH&ijn0vY?! 70KM0MuWd!0Duv#mX20ZN0EJ09
Document	+w10@hkY;*(-F[0W]zL70f~-s0*tNbNaJH}00=)0CmPgV0psYUNFzA90f^m~]400~'<50G{+00050 Pywz20[vh402
Phone	1090MI0c5010TtQuTkXu<80b30')frm:W&Z<J{40$n0a0$f0a}B60w+tb $0l`j00k D&X#IS0zRJd80THd10r~I00>0>7
Date of Sale	Nq_U-3060-vkD)C0defm0*80z0/)_40F0MR0T!M`,00qK? A{30N20BgJ#Nl0N0XQ]jT10!Rjq-zFFT)R10,0Nr80i?^9
Sale Value	_#p30SHJ_(`*~?dPCRj0050?ovH#0/[N60? cWct00rofZ0do70#f;}p0r900/iaM=-os*_fQ'i0>LJ}DF@-pHzV
Hash	_Yr+XADIJv0x]*UwYg40Y90dN@(0=60600zBYrAyey0y_700`<80byl0i YP*C}<;60/wZ0h$ 0J,e*&)040@tD4040^b

Table 5: Ciphered record (version 2)

Field	Value
Name	M70?90kP0l200O>KCTOBJ0B500t+O:VD70nif0c70UUca?Cvf? 0)NjEv/TNShvQ0@(090P+kwsBMA50q L_Y)Kv!90`6
Document	vC0zOM70/S=bhk800?_qE&Qxt/>40Jb20i&K0Zk0o<0wER-0e0s])30LGZF`0y;40/06090nQY0hI,0[0=S0MMNSM~0M203

Phone	^uBT0yw>DB)0!D020$hz70i0!0'L0{>(`0! 0N=700XOx90nWj00k&0KSU&}/ hoO[40800z^jO[S,zwjUdyz0^iXKCQM0
Date of Sale	70rI00:XB^m{40/!<40T60+DY0N0AYY*W,[Y qL^+0J,! #J0n7060m+Dvhbz_o601040Dr`eCTNwnmt>G30_Jf0,A20V)
Sale Value	~m`>W  t^50'gdQD2050jgm^MpV{v70R20`020viz kbMEb*S[yB]q90,vC0dj)<40fVgZT^cX'40,=W#0i klf}c
Hash	BBLbUR,0@_Qi&MY20Q ^^K+q90k>E-;0q'fWTrXUs10kmA30^Wqx[T60QMD;b0=z{Yk0a,i? Rex0gFY00O70E0DB9

From the table plus the hash field it is possible to safely encrypt and decrypt the records. The noise r has 257 variations and the encryption functions are 8. This way we have a total of 2056 encryption variations for the same data with the same key. The noise level can easily be increased without compromising the performance of the algorithm, but by adding too high a noise level the speed of data search is compromised.

An apparently effective noise level is 16, thus allowing 128 combinations for each cipher block. This value seems reasonable to us in terms of security and effective in not hindering database searches. Thus, since a field can be encrypted in 128 different ways, it is necessary to search all of them to obtain the data for decryption.

But in the search it is necessary to include random values to avoid cryptoanalysis by search frequency, indicating that those data were produced by the same cleartext. So it is highly recommended that when referring an SQL query to the database more queries are done with random values that the legitimate client can easily isolate in decrypting the data but the attacker will be stuck in them, not knowing exactly which cryptograms are true and which are false.

Example of an SQL query:

> **Select** Code **From Table**
> **Where** (Code = x_1) **or** (Code = x_2) ... **or** (Code = x_{256})

In this case the legitimate client of the system, who has access to use the keys, creates a search vector $\mathbf{Y} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots, \mathbf{x}_{254}, \mathbf{x}_{255}, \mathbf{x}_{256}]$. The contents of $\mathbf{x}_1$ through $\mathbf{x}_{128}$ represent all possible encryptions for a given search key and the values $\mathbf{x}_{129}$ through $\mathbf{x}_{256}$ represent random values, however with characteristics similar to the true data with respect to their size.

The legitimate user applies a random permutation to the data in vector Y. This vector could look like this: $\mathbf{Y} = [\mathbf{x}_{159}, \mathbf{x}_{126}, \mathbf{x}_{13}, \mathbf{x}_{16}, \mathbf{x}_{149}, \ldots, \mathbf{x}_5, \mathbf{x}_{249}]$. Then he does an SQL query on the database with the data in this random order, and even if the database manager changes this order, the legitimate user of the system will be able to separate the real data from the fake data because he has the decryption key. The SHA3-224 hash allows the data to be verified by concatenating all the fields except the last one, which is the hash (this hash is encrypted and can only be accessed by the user who has access to the system keys).

It is important to note that while the fake data returns no search results, the real data may also return nothing, since a particular field in a record may not have been encrypted in all possible ways by the key in use.

So the legitimate user can get rid of the useless data, but the attacker will most likely have great difficulty knowing which data is “correct” and which is “random data”. Of course, for each database query there will be a cost of 256 queries. But it is a worthwhile effort if the server is powerful enough to handle the requests and the response time is satisfactory. In this case, we pay the price of security with a slower performance of searching, changing, adding and deleting data.

5 Cryptoanalysis

The NJS algorithm is symmetric and non-deterministic as we have already shown in this paper. RSA is perhaps the world's best known asymmetric system that is still used in many scenarios and has not yet been completely broken. It will only be effectively broken with the advent of quantum computers.

But why mention RSA in this context if we are talking about a symmetric cryptosystem? The answer is simple: the RSA system and the NJS algorithm share a very close level of security, at least theoretically. RSA is a deterministic cryptosystem, i.e. a message, encrypted with the same key, always produces the same ciphertext. In the case of the NJS algorithm we have a non-deterministic system that can produce different ciphertexts from the same key, using a pseudo-random (or totally random) number generator.

When we analyze the structure of RSA we see that the key is mostly restricted to knowing the modulus (N = p * q) which in turn is a composite number with 2 prime factors of high magnitude. The exponent, which raises the number that represents the clear message, is usually fixed at 65537 (2^{16} + 1). This number, when transformed into binary format is represented by the bit sequence **10000000000000000001**.

Employing the techniques of fast powers we have that an RSA cipher corresponds to only 2 simple modular multiplications (being even more precise only one). This is because there are only two “1” bits in the number 65537. And what does this have to do with the NJS algorithm? In the NJS algorithm the clear message is encrypted by at least three multiplications, three sums and three XOR operations, and to perform these operations three different modules are used. In the RSA system the module is public, in NJS there is no public information other than the cryptogram.

In an RSA encryption we clearly see what has been explained in this section[3]:

[3] Let's use small numbers to make our reasoning easier.

Table 6: RSA Key

Prime Number p	246497
Prime Number q	8909161
N = p * q	2196081459017
D = (p-1) * (q-1)	2196072303360
e	65537
e^{-1}	1845334723073

Consider a message M, whose content is 1234567890. To encrypt this message we can consider exponentiation:

$$C = M^{e} \bmod N = 1234567890^{65537} \bmod 2196081459017 = \mathbf{1033311264004}$$

The operation of modular exponentiation with giant numbers (as is mandatory in a case of real encryption with RSA) is very costly, and can only be accomplished by employing fast exponentiation techniques. To apply this technique in the cited example we would have the following situation:

Table 7: Speed Exponentiation Technique

Message	Exponent	e = 65537 (binary)	Result
1234567890	1	1	1234567890
	2	0	479610188505
	4	0	991099202795
	8	0	1515777153364
	16	0	1521376824852
	32	0	1515437109397
	64	0	693892638319
	128	0	920413295621
	256	0	1401497722674
	512	0	601437657220
	1024	0	1027144228206
	2048	0	1173322269509
	4096	0	1762446373179
	8192	0	499695743722
	16384	0	1441246836953
	32768	0	1665320632773
	65536	1	222544470823

Then we have that:

$$C = M^{e} \bmod N = 1234567890^{65537} \bmod 2196081459017 = \mathbf{1033311264004}$$

Which is equivalent to:

$$C = C * C^{65536} \bmod N = 1234567890 * 222544470823 \bmod N = \mathbf{1033311264004}$$

In fact, the multiplicative inverse of **222544470823** to modulo N (**2196081459017**) exists (we will name this value **K**) and can be represented by **L** (**1977535799413**). And we conclude our reasoning by saying that:

$$\mathbf{M = C * L \bmod N} = 1033311264004 * 1977535799413 \bmod N = \mathbf{1234567890}$$

With this exposition we want to show that an RSA encryption with the fixed exponent 65537 ($2^{16} + 1$) is equivalent to a single multiplication of any message M by a value smaller than the modulus N (which we call K). It is clear that for each M there will be a different K value, and consequently for each K value there will be a corresponding multiplicative inverse L.

Therefore, it is plausible to conjecture that, given the longevity of the RSA algorithm, the codebook **R** formed by all possible messages generates a codebook **S** formed by all possible cryptograms and the numerical ratio between the codebook **R** and the codebook **S** is sufficient to provide the necessary security against all known cryptoanalytical attacks applied against RSA. If such a fact were not true the RSA would already have been broken without needing to factor the N-module, which, to date has not been performed, or at least not disclosed.

The **R** codebook and **S** codebook mentioned in the previous paragraph can be understood (to remain using the RSA key we presented) as a permutation between the elements of the clear message and the cryptogram.

Table 8: Codebook

Position of the clear text	Plain Text R-code book	Position of the cipher text	Cryptogram S-Codebook
1	1234567890	4	1033311264004
2	1234567891	8	1933023821024
3	1234567892	10	2018464103304
4	1234567893	1	556086417688
5	1234567894	7	1590524243055
6	1234567895	9	1547302878353
7	1234567896	6	1480606826760
8	1234567897	5	701131536877
9	1234567898	2	1522501699385
10	1234567899	3	863040139835

In the case of fixed exponent RSA (2^{16}+1) the order of the codebook S is determined only by the value of N (modulus). When we refer to the order of the S-codebook we are referring to the values [4, 8, 10, 1, 7, 9, 6, 5, 2, 3]. Of course, by increasing our R-codebook and calculating our S-codebook we will obtain a list with N-1 elements arranged in a seemingly random order (although it is not). As we clearly show in this section the modular exponentiation of RSA, in this specific case, can be replaced by a simple modular multiplication on N, although for each message M this multiplier is distinct.

The NJS algorithm attempts to replace this single modular RSA multiplication with a set of at least 9 operations using distinct algebraic groups. The operations are contained in vector A = [MUL_1, ADD_1, XOR_1, MUL_2, ADD_2, XOR_2, MUL_3, ADD_3, XOR_3] or in vector B = [ADD_1, MUL_1, XOR_1, ADD_2, MUL_2, XOR_2, ADD_3, MUL_3, XOR_3] or in vector C (which can be defined as A*2, A*3 or A*4) or finally in vector D (which can be defined as B*2, B*3 or B*4). The total number of operations can vary between 9, 18, 27 or 36 changes in the clear message M, and the number of modules used can vary between 3, 6, 9 or 12. These parameters can be easily adjusted and adapted to promote a better level of security.

It seems reasonable to us to believe that this change made in the NJS algorithm relative to the RSA algorithm should not compromise the security of the algorithm. However, we point out here that we have no mathematical proof for such a statement. But considering the fact that the NJS algorithm is not deterministic like RSA the number of distinct S-codebooks relative to a fixed R-codebook (plaintext) increases exponentially (without the key being changed).

Disregarding the initial noise that is added to the plain text we can establish the following relationship between an R codebook and its corresponding codebooks:

Table 9: Different encryption possibilities

R code book **Number of elements**	**S Codebook** **Possible permutations**
1	8
10	1073741824
100	2.037×10^{90}
1000	1.230×10^{903}
10000	7.940×10^{9030}
100000	9.970×10^{90308}
1000000	9.704×10^{903089}

The number of elements in the codebook refers to the modular limit N. In the NJS algorithm the size of the N module we recommend is at least 512 bits. It is also important to note that in the case of the RSA system the N module is exposed (the system attacker has access to it) and in the case of the NJS algorithm the various N modules are not accessible to a potential attacker, given that the NJS algorithm is symmetric. The modules used in the NJS cryptosystem must be prime numbers.

6 Source-code

To facilitate the understanding of the algorithm we have made available the complete source code written in the Rust language. The code is divided into 8 files with the following names:

[1] **main.rs:** This file contains the call to all functions of the NJS algorithm.
[2] **compress.rs:** File with functions to compress and decompress numeric data using only printable characters and easy to save even in TXT files.
[3] **show_message.rs:** Functions to show messages on the screen.
[4] **math.rs:** Functions to calculate SHA3-224 hashes, create long integers, and potentiation of long integers.
[5] **intermediate_file.rs:** Processing functions, hash calculation, encryption, data pattern transformations, and more.
[6] **cripto.rs:** Data encryption and decryption.
[7] **read_write.rs:** Reading and writing files.
[8] **cargo.toml:** Rust language configuration file.
[9] **database_min.txt:** Small example database with dummy values.

The source code is available in Annex III of this paper.

Conclusion

The NJS algorithm presents a non-deterministic encryption model that can be used to encrypt relational databases. The security level of the algorithm seems to be close to the security level of RSA with fixed exponent (2^{16}+1) although this is not proven in this work. The encryption technique (although non-deterministic) allows to have an exact decryption (at least for secure hash functions).

The NJS system can protect information on a cloud-hosted database server by performing direct lookups on the encrypted data using a larger number of requests for this. Although there is a stress that the server must endure due to a significant increase in the number of requests, the data remains encrypted at all times and is decrypted only on the client computer. To encrypt a simulated database in TXT format with 100,000 records took 43 seconds (using a personal computer with i5, 3210M, 2.5 GHz, 6 GB RAM processor) but decryption was much more costly spending 302 seconds. Importantly, the database was increased 6.72-fold. Encrypting a large database can take a long time but searching and changing records can be done quickly on the client workstation with bearable processing stress. On the other side, the database server is forced to make fast and frequent responses, requiring the machine to have sufficient hardware to support this demand.

Although we have presented some analysis points in section 6 of this paper, we know that the algorithm is not properly validated in this respect, and this is a long road for future work. Given the rigor necessarily indispensable in the validation of a cryptographic algorithm, further research is needed.

We finally conclude this work by making available the source code written in the Rust language (Appendix III). It proves that the NJS algorithm works flawlessly and will certainly be a useful tool to better understand its structure, either in its highlights or weaknesses that can be pointed out in cryptoanalysis.

Acknowledgments

We thank all the supporters of this work. The NJS algorithm is entirely dedicated to Noêmia Josefina da Silva, a woman who knew how to face life's challenges and overcome them with an unshakable faith. A unique example of maximum overcoming portrayed indelibly in her honorable and extremely ethical life.

References

[1] Hoffman, Nick. **A Simplified Idea Algorithm.** Available at: https://www.nku.edu/~christensen/simplified%20IDEA%20algorithm.pdf. Last access: July 07, 2022.

[2] Aissi, Chandra M. Kotaand Cherif. **Implementation of the RSA algorithm and its cryptanalysis.** Available at: https://peer.asee.org/implementation-of-the-rsa-algorithm-and-its-cryptanalysis.pdf. Last access: July 07, 2022.

[3] Sarkar, Santanu; Maitra, Subhamoy; Sumanta, Sarkar. **RSA Cryptanalysis with Increased Bounds on the Secret Exponent using Less Lattice Dimension.** Available at: https://eprint.iacr.org/2008/315.pdf. Last access: July 07, 2022.

[4] Peng, L., Hu, L., Lu, Y. et al. **Cryptanalysis of Dual RSA.** Des. Codes Cryptogr. 83, 1–21 (2017). https://doi.org/10.1007/s10623-016-0196-5. Last access: July 07, 2022.

[5] Melton, Jim. **SQL Language Summary**. Sybase, Inc., Sandy, Utah. Available at: https://dl.acm.org/doi/pdf/10.1145/234313.234374. Last access: July 07, 2022.

[6] Patel, N., Oza, P., Agrawal, S. (2019). **Homomorphic Cryptography and Its Applications in Various Domains.** In: Bhattacharyya, S., Hassanien, A., Gupta, D., Khanna, A., Pan, I. (eds) International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, vol 55. Springer, Singapore. https://doi.org/10.1007/978-981-13-2324-9_27. Last access: July 07, 2022.

[7] Sakr, S. **Cloud-hosted databases: technologies, challenges and opportunities**. Cluster Comput 17, 487–502 (2014). https://doi.org/10.1007/s10586-013-0290-7. Last access: July 07, 2022.

[8] Kulkarni, Saurabh; Urolagin, Siddhaling. **Review of Attacks on Databases and Database Security Techniques.** Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.1729&rep=rep1&type=pdf. Last access: July 07, 2022.

[9] Hussain, Iqtadar; Ahmed, Fawad; M. Khokhar, Umar; Anees, Amir. **Applied Cryptography and Noise Resistant Data Security.** Available at: https://downloads.hindawi.com/journals/scn/2018/3962821.pdf. Last access: July 07, 2022.

[10] Wright, Gavin; S. Gillis, Alexander. **Side-Channel Attack.** Available at: https://www.techtarget.com/searchsecurity/definition/side-channel-attack. Last access: July 07, 2022.

Annex I: Example of system key

Encryption key:

k_{11}	798599428663203109956158163914448953869352662328770360215573629792670326080243059579885575172159941160996858791492426951050125434098259893785644324473094
k_{12}	389497126888089591881357186534982077698653556404469446893882300647692836143353197228390384194426695893807917767545151499602887246947718593727448997248475
k_{13}	1652692441615594879688938721164988030021629470487536871620129120090230908908754745799804672338033800526737708514965500441016077256195537282122354800594924
M_1	1072246721897826892625869622674792749167613767464104561426341265328962212586363221049615864429570199144438093224998782442392073359444316181120260357700 8773
k_{21}	3941381251296432391240385372518378389889668192544602147367559773318242289362065146375652442786705743394905107240300826602999576955133001725533228366416498
k_{22}	679968032814501248768476391390032756738339217273517137049704455723925545469640575828725844859598672641936331271979289600118870937100720954066294927776550
k_{23}	3835248109069771657151213531800158263562575766525562242988648811472403732399692957653608768504173863214872485435240209474647530755370512731260367140136452
M_2	1413197736680543678556160459898319343866136471176340024857287975340225030583680900639064786516517906611244245165352236994763206927610986677000365232318779 1
k_{31}	8701184376278655751331556964155239257951186269523030826634992031106613114893203081022028234991995476452470920166304139172438412582464232396044419717239034
k_{32}	1340127116577903967713207742920363232826314476987431297006814887631412839234950178595408302582366253635305483214109280304313135150182939779447630669779720 0
k_{33}	736505582021365225789618479870370210971864544396648125138203811247454017305834324812681620108568952119615480939105624796258362583151978281100977742026848 8
M_3	6275156795718212190858999603757570074706648867634946845738553786978291408590463022031701834327362407206692603627565562578804757449679930245520524674856550 9
k_{41}	1069883750971272118706027493842752945703320592318877912780385213122630895279543772853364099525420238147177122549775819502190946730038286030552907129690211 49
k_{42}	751299818827266525133625043921491184450961490847860967905426199534353298330671426654526320110691796614792707698579867548141640981200290448835234216002791 5
k_{43}	6978826144345539294367057488062306106202334623620533984050642970163427975490920876620982580987177272162610722862490779196356398057286127244517742585881012 9
M_4	563182193817448165444395118267526105913191028582645492631266510546744315713640483789976327078295689508583931189713378020461035111061571711358175709862669281
k_{51}	3173671788286906186321182064974056956122532573664193750022473407373752771566457770926459693434439847281202170845638211537446063960516322482144024281909438 20
k_{52}	338559096142347577138253528274203912748566218636779391682231383470610819277759221343251133863422154101312082161426639930879444864114808225291457998801536749
k_{53}	892001741802407690031249107856599359755681745259557720085868767529673462308350686742496129317105300712117800044040353224165261038217755522237486401422533964
M_5	33329743085315113737010539623297528881942999390631349022533616076173198088719102556131655564526399609486058699100037633838013750607937854655542009954619074 67
k_{61}	269385232771186345344106811925749342773619735064283182479294410913063925743275693890540779453874934283866779638389084384909089126500444691201429589055808795

k_{62}	620789417278440290185624306916416001489066155397609749041523203619835528988780863380724625315105746738160711515857826425880862823431747203180962260225557075
k_{63}	827619511185480261716651886890239297243718710546933353516063281658468658073920139316084532288748260709746842094866590537398443302680752591017630247877018164
M_6	9957360231301321489089519270327658387667679133723634179699764598863956916075326927518819600196215681241225228051832571702783982090495383748302721565963337207
k_{71}	3500384289759527548859348523658400723429669726390847413171074610727586918805708513630388310382189102763575543001892976610026095416646397685085920654636174580
k_{72}	4803899192299543323063204159992576142453222130529272022672444979632686149928226788565735945251474418239196023030971051538936883008977042636655287059018166508
k_{73}	5898759033125983532752367356407877324919289011212141704153050118338785183143884303808618179712495623483198034552686468407356268370134766461896934886729989200
M_7	39336567463872650496661187719967295324131671510484908235533132078268965879985258102876153189975637397803255967592003990034868715526061633664076597821547293153
k_{81}	5832291003276764274912584542249002332379483882645481377083249624756443947933706519124179904353686527271447612249232616222101378568238038791702869116715412384
k_{82}	2066735922108930579965107027039035560890318216539334587319610103575954315244288013327766155639207932505253195089353084835441222656328647980638370435195223679
k_{83}	8517096333202442940332690116185642973007465316314727214817751099953687312168758973588885647323336295639280105466124593940423282734707279076090573637085090889
M_8	176423026492805379452535196675197741188845962167097146321914937703777274971449079065040780065967098297333212276392743191552230738505519831810235757400980386471
k_{91}	1350743840345525570672405535289515050958995927045219129786165932760542988030634100700263457212457699324987213515820194852225349473294173470236326449624180564
k_{92}	46708533305859862246342212395656626817956263986528451684539220950284947582850968288996469102829137996028889939932253615710652507912419720255383696419754095008
k_{93}	89863313163967525687555563825704682515897308079485684094045147873611610871146150758815673077156715279412146409320241309980819503775179165765310459810392097430
M_9	265156451511952280844531442218623830166845676522190229730541892028456373361413164492877214978703953815327028686632303249577196727133760024471154069296166709617
k_{A1}	177616685012854735530825038140228670242814237563313605357533530920301373276932624106482823869389442271099009008253407617780462477030070460389023891643549384758
k_{A2}	123079347458416563142365066939477891697707265452515502324139747903078160679685726436401968264126233552740948506939798378342033491735883412403247875785573802278
k_{A3}	190854684300341645086901788245016404467730702362342213971124248146292125735295058788639181172579016001234921786707930677227420135596224864681373291165079328504
M_A	3347274343780223862823291341646964725903199566549021438196990099331996183105207242477693761428579390816977510197372565370535689628837057251024553191256985937601
k_{B1}	1629141337980619199831025715730707169578223492545353971276058708777282392368748853228139625142330463138009979138144450660871209198834994992155138819752752139914
k_{B2}	966117457876338988967473813045694119842036013924189698016967733504775203512847620221066938931175651215660345952187457885434446403919701747723740130023475656217
k_{B3}	1705516819451008281453498162591529629767235539374824184166398283502433703472630231229867453581482908661022261923489950309410526603388975337157910467857787639168
M_B	7190616924772204931865998742111727354826338817900580731475647537380892488220345785109323388049519310522590087131429105564260801558422560148681900427878042427877

k_{C1}	20921476184925849892820946399610401784199696341693606040145251807049758429016552876388618439671812153877383445942136220275855085146452302378096770075398324908 91
k_{C2}	23824064693359259554861463289975866469588906139761712641988273604471077261508001292222741081656634963531167450521551048781714938277993417930086737526556332855 96
k_{C3}	42505579115717305913948872663460264689492004477578030477039025278694294274175075394391187278328797338225539912276755057500806247807590233091037848389560653410 27
M_C	545016257147091535136795878518077919309397705961035010053872588859744171970421331064789949712076816684508871729940463335661198143399280551661696722646110271104 17

Decryption key:

k_{11}^{-1}	9923867790315065816302538062833478537806785012312275254047839023496951799783389150916273069123542050283384073458495397472870608160344901917416959252535679
k_{12}^{-1}	4690921015443614285419659973259535820446650757012081778085295702963808885227781396776512034061141177653033055237065260332117677781772365708940493776907720
k_{13}^{-1}	1652692441615594879688938721164988030021629470487536871620129120090230908908754745799804672338033800526737708514965500441016077256195537282122354800594924
M_1	10722467218978268926258696226747927491676137674641045614263412653289622125863632210496158644295701991444380932249987824423920733594443161811202603577008773
k_{21}^{-1}	10190596115509004394321219226464815048771696519218798101205319980084008016474743860014995422378473322717537344413221543344632492320976865044470423956771293
k_{22}^{-1}	746254082170794130418688890624880993729536167667490966090174920866478835383646881398515661065048489419473944580260803556822874234435848619335362737271967
k_{23}^{-1}	3835248109069771657151213531800158263562575766525562242988648811472403732399692957653608768504173863214872485435240209474647530755370512731260367140136452
M_2	14131977366805436785561604598983193438661364711763400248572879753402250305836809006390647865165179066112442451653522369947632069276109866770003652323187791
k_{31}^{-1}	54050383580903466157258439073420461489115302406826437630750545838676300971011427139294990108281628595614455116109351486615609161914335070059160827031326475
k_{32}^{-1}	55166610868712856244865043441162380233950704810569034232218842807868814522154318307410670775636551710115116263161270196905021500080324071491294262481378139
k_{33}^{-1}	7365055820213652257896184798703702109718645443966481251382038112474540173058343248126816201085689521196154809391056247962583625831519782811009777420268488
M_3	62751567957182121908589996037575700747066488676349468457385537869782914085904630220317018343273624072066926036275655625788047574496799302455205246748565509
k_{41}^{-1}	456193818720320953573792368883250811342858969350757701353227989234481226185686106504639917125753665693866218934735796070241940438057743108302884996893648132
k_{42}^{-1}	12662894560411496715599214033734503578697658366294305692346068347071341491024804555663575158889760211902844728485199053395544938089443755939917310618134632
k_{43}^{-1}	69788261443455392943670574880623061062023346236205339840506429701634279754909208766209825809871772721626107228624907791963563980572861272445177425858810129
M_4	563182193817448165444395118267526105913191028582645492631266510546744315713640483789976327078295689508583931189713378020461035111061571711358175709862669281
k_{51}^{-1}	3015607129702820755068935755832347192582046681696715527251114266879944531715264478520519587109195976220485652825439942230056768664742153217339798567270963647
k_{52}^{-1}	836077549888947732041958162391294791310977602553466253579926226530890929775473109987727922380128680656188384225302151012056729182016324868737300420198651949

k_{53}^{-1}	89200174180240769003124910785659935975568174525955772008586876752967346230835068674249612931710530071211780004404035322416526103821775522237486401422533964
M_5	3332974308531511373701053962329752888194299939063134902253361607617319808871910255613165556452639960948605869910003763383801375060793785465554200995461907467
k_{61}^{-1}	9687974998530135143745412458401909044894059398659350997220470187950892990332051233628278820742340746957358448413443487317874892963994939057101291976907528412
k_{62}^{-1}	1299279621548368249837273450007052032348154128998589882214917750624902832006875413486579499511852506225864082227915489122117831327292593168904954837841337349
k_{63}^{-1}	827619511185480261716651886890239297243718710546933353516063281658468658073920139316084532288748260709746842094866590537398443302680752591017630247877018164
M_6	9957360231301321489089519270327658387667679133723634179699764598863956916075326927518819600196215681241225228051832571702783982090495383748302721565963337207
k_{71}^{-1}	35836183174113122947801839196308894600702001784094060822362057467541378961179549589245764879593448295039680424590111013424842620109415235978990677166911118573
k_{72}^{-1}	27022493229487031671447974820306537179927272650829693984524487079073864903469408172704027770681993450029548163333926030892816967336219810621844602569321637611
k_{73}^{-1}	5898759033125983532752367356407877324919289011212141704153050118338785183143884303808618179712495623483198034552686468407356268370134766461896934886729989200
M_7	39336567463872650496661187719967295324131671510484908235533132078268965879985258102876153189975637397803255967592003990034868715526061633664076597821547293153
k_{81}^{-1}	170590735489528615177622612132948738856466478284451664944831688079020831023515372545916600161613411770061764664143510575330129359937281793018532888284264974087
k_{82}^{-1}	175502270593753722372146901781449653816224933074638743123889964437828608775327948333333104244349172997529785232911718598220394449346773574207938573270075999885
k_{83}^{-1}	8517096333202442940332690116185642973007465316314727214817751099953687312168758973588885647323336295639280105466124593940423282734707279076090573637085090889
M_8	176423026492805379452535196675197741188845962167097146321914937703777274971449079065040780065967098297333212276392743191552230738505519831810235757400980386471
k_{91}^{-1}	263805707671606755273859036683334315115886680595145010600755726095695830373382530392176951521491496116002041473116483054724971377660465851000917742846542529053
k_{92}^{-1}	191924978123257537159475387956464914377899165696609533852188042168914691750650757463270809251983718727573571024549840505407353605970963598335105504580222137717
k_{93}^{-1}	89863313163967525687555563825704682515897308079485684094045147873611610871146150758815673077156715279412146409320241309980819503775179165765310459810392097430
M_9	265156451511952280844531442218623830166845676522190229730541892028456373361413164492877214978703953815327028686632303249577196727133760024471154069296166709617
k_{A1}^{-1}	3169657658767369127292466303506736055660385328985707832839456568411694809828274618371210937559189948545878501189119157752755227151806986790635529299613436552843
k_{A2}^{-1}	1677772935690591878331841466997645726421256858089774386352396769619906915147825331055782444672971075779064328651566999867139042234917358055068772216323896006193
k_{A3}^{-1}	190854684300341645086901788245016404467730702362342213971124248146292125735295058788639181172579016001234921786707930677227420135596224864681373291165079328504
M_A	3347274343780223862823291341646964725903199566549021438196990099331996183105207242477693761428579390816977510197372565370535689628837057251024553191256985937601
k_{B1}^{-1}	5561475586791585732034973026381020185248115325355226760199588828603610095851596931881183762907188847384580107993284654903389592359587565156526761608125290287963

k_{B2}^{-1}	5954663124525235688839890939411347347550952415967689320314648174427882623780048428438741634375738396937361308961230100995076218153351169712180851384167372565182
k_{B3}^{-1}	1705516819451008281453498162591529629767235539374824184166398283502433703472630231229867453581482908661022261923489950309410526603388975337157910467857787639168
M_B	7190616924772204931865998742111727354826338817900580731475647537380892488220345785109323388049519310522590087131429105564260801558422560148681900427878042427877
k_{C1}^{-1}	52409478096216568524397493211846751752519800961934140401372733705269441354140477818840133127240500453063148828399832711538534305825282824928359995257071194619526
k_{C2}^{-1}	44600100520712772545029384541829422187201732511866482786918855121853130557404655702518681485631023417772288774363854198398244970249941064254119317535006939329779
k_{C3}^{-1}	4250557911571730591394887266346026468949200447757803047703902527869429427417507539439118727832879733822553991227675505750080624780759023309103784838956065341027
M_C	54501625714709153513679587851807791930939770596103501005387258885974417197042133106478994971207681668450887172994046333566119814339928055166169672264611027110417

Sbox:

```
 64, 124, 108,  28, 188, 254, 134, 154, 187, 235, 199, 240, 251,   2,  70, 156,
  3, 133,  66, 107, 128, 150,  79,  77, 130,  83,  58, 121,  49, 246,  94, 101,
176,  60,  57, 238, 202,  36, 126, 198, 158,  86, 194,  47, 174,  18, 144,  26,
151, 120, 175, 205, 201,  42,  82,  95, 179, 164, 112, 186, 217, 104,  45, 223,
200,  99, 195, 247,  46,  78, 102,  93,  34,  65,  97, 105,  92, 115, 139,  31,
 85, 145, 234, 110,  33, 123, 137, 173,  72,  40, 140, 252,  80, 185, 160, 215,
227, 177, 218,  21,  10, 161,   7,  71, 245, 109,  43, 116, 131, 142,  20, 113,
232, 129,  16, 210, 253,  30,  56, 167, 216,  63, 208, 125, 189,  25,   1,  52,
106, 163, 152, 182,  19,  69,   9,  29,  27, 193, 178,  51, 228,  90, 122,  73,
203,  12,  41,  35, 225,   4,   8, 118, 157, 165,  96, 206, 135, 127, 147,   0,
219, 244, 192,  59, 204, 184, 153,  14,  75, 111,  15, 168, 214, 138, 114, 249,
169, 207, 141,  32, 166, 143,  53, 149, 226,  22,  38, 146,  74,  76, 183, 224,
  6, 222, 233, 117,  48,  91, 236,   5, 170, 162, 243,  55, 213, 197, 248,  61,
 37,  11,  24, 100, 242,  87, 239, 103, 136,  89,  62,  54, 211, 237, 181,  88,
159,  23, 231, 196, 155, 220, 132,  13,  50,  67,  44, 221, 148,  17, 171,  39,
180,  68,  81, 209,  98, 119, 191, 230, 250, 255, 190,  84, 212, 229, 241, 172
```

Sbox^{-1}:

```
159, 126,  13,  16, 149, 199, 192, 102, 150, 134, 100, 209, 145, 231, 167, 170,
114, 237,  45, 132, 110,  99, 185, 225, 210, 125,  47, 136,   3, 135, 117,  79,
179,  84,  72, 147,  37, 208, 186, 239,  89, 146,  53, 106, 234,  62,  68,  43,
196,  28, 232, 139, 127, 182, 219, 203, 118,  34,  26, 163,  33, 207, 218, 121,
  0,  73,  18, 233, 241, 133,  14, 103,  88, 143, 188, 168, 189,  23,  69,  22,
 92, 242,  54,  25, 251,  80,  41, 213, 223, 217, 141, 197,  76,  71,  30,  55,
154,  74, 244,  65, 211,  31,  70, 215,  61,  75, 128,  19,   2, 105,  83, 169,
 58, 111, 174,  77, 107, 195, 151, 245,  49,  27, 142,  85,   1, 123,  38, 157,
 20, 113,  24, 108, 230,  17,   6, 156, 216,  86, 173,  78,  90, 178, 109, 181,
 46,  81, 187, 158, 236, 183,  21,  48, 130, 166,   7, 228,  15, 152,  40, 224,
 94, 101, 201, 129,  57, 153, 180, 119, 171, 176, 200, 238, 255,  87,  44,  50,
 32,  97, 138,  56, 240, 222, 131, 190, 165,  93,  59,   8,   4, 124, 250, 246,
162, 137,  42,  66, 227, 205,  39,  10,  64,  52,  36, 144, 164,  51, 155, 177,
122, 243, 115, 220, 252, 204, 172,  95, 120,  60,  98, 160, 229, 235, 193,  63,
191, 148, 184,  96, 140, 253, 247, 226, 112, 194,  82,   9, 198, 221,  35, 214,
 11, 254, 212, 202, 161, 104,  29,  67, 206, 175, 248,  12,  91, 116,   5, 249
```

Annex II: Example of a table

Table with fictitious data:

1	["Airtom Fregapani Carpeta Cunha Merraco", "382248368-07", "(41)98136-7571", "23/03/2019", "21837.88"]
2	["Layne Bellani Dallarmellina", "257253618-57", "(16)95336-9701", "08/11/2007", "21022.53"]
3	["Yuri Buchmann", "179790017-93", "(45)97040-6314", "23/11/2011", "21288.60"]
4	["Sibeli Ceolotto Grosseli", "991276898-05", "(74)94410-3803", "09/04/2022", "11690.95"]
5	["Soila Giacomel", "634814762-00", "(48)91308-7529", "21/04/2022", "5269.11"]
6	["Anelito Broccardo Bourguignon Balbinotti", "906757754-34", "(62)99754-9447", "13/07/2010", "10543.24"]
7	["Emelly Esmeria Sodre Etzberg Degel", "264697667-60", "(45)91010-2510", "28/02/2015", "11544.29"]
8	["Jonatham Caruso", "338338942-18", "(96)99154-0933", "06/12/2018", "9825.37"]
9	["Cipriano Gentina", "860563543-65", "(67)92805-7573", "22/02/2015", "8507.21"]
10	["Tailaine Bozzoni Anesia", "286895488-10", "(89)90742-9484", "28/03/2020", "11678.59"]
11	["Edimarcos Celleghin Rigone", "246561540-73", "(89)92624-4731", "15/08/2002", "134.58"]
12	["Ivano Dobger Green Claudete", "002409746-39", "(97)95326-8142", "03/09/2017", "4601.07"]
13	["Laynara Bacelar Bentele Cattani", "138044932-14", "(32)92012-0287", "27/09/2021", "9088.54"]
14	["Walner Mangieri Crass", "892722759-00", "(32)91775-6893", "20/03/2016", "9674.10"]
15	["Alyson de Zan", "958168983-24", "(41)94519-7954", "20/08/2010", "3426.03"]
16	["Noilson Buhnert", "565435673-27", "(17)95770-0423", "14/12/2018", "3532.94"]
17	["Rose Bizarra", "834993628-07", "(87)95527-1914", "15/03/2012", "14800.47"]
18	["Welder Guaresi", "764225432-77", "(45)92426-6847", "09/02/2013", "6279.69"]
19	["Erique Brunkmer Maiolino Divino", "598989914-28", "(65)95271-0645", "09/01/2006", "861.20"]
20	["Bernadino Friedrich Brantano Friolani Clelia Deolinda", "808203951-03", "(5)95105-9519", "09/03/2018", "21192.92"]
21	["Aldivan Aguzzi", "485410475-10", "(54)94240-0122", "09/02/2013", "1535.84"]
22	["Emerson Moura Filho Dias Bierbrauer Ferraz", "697960271-56", "(89)99572-4141", "23/08/2017", "21738.87"]
23	["Sheilla Faller Farinon Barneche", "756795415-03", "(12)95491-3910", "13/07/2003", "12639.07"]
24	["Blenda Favatto", "194238594-70", "(84)95599-8669", "19/11/2005", "16011.41"]
25	["Kelita Edilaine Barboza", "784825094-63", "(74)92803-6674", "04/06/2017", "5262.86"]
26	["Verenice Drugg", "793829161-39", "(19)97986-9204", "18/05/2007", "2967.25"]
27	["Jocenildo Aquilio Winters", "464520485-70", "(31)95886-0344", "21/08/2016", "5846.81"]
28	["Mairia Etcheverry Cuomo Zaccardi", "321857950-29", "(46)95030-2306", "02/05/2013", "1017.65"]
29	["Jefter Gasquez Legnaghi", "210287841-20", "(95)91085-3949", "07/06/2016", "6406.82"]
30	["Ibrain Eisenhut", "037333705-85", "(86)90453-8840", "21/09/2003", "8821.55"]

Encrypted table:

1	["20zpfG080I,70C`rz;;0d=200=o]<60l{G50o50Cw]80Py0ilZ;MT q&msa&0sw _0(W-4060tW-gem-wnU90L@0O^LZK0C6", " U60FgljH90ZFVDl,p;!w<100=:EqgzGb^ab<GdtAD 20)N0-fdppYnmS40CV10B!C<<H{W#Dz#f^0sI$8000/QOC{", "$00eZ200_0oH600*L=T4080xZ0mWcJXyRf0C) %#Uj00><Ytn{WztT[Zzy30I0ll&oH:00*x[i0A}KSIljW800G,mEf;YD3", "GGg{w[0$YJY p0W! 0fgG+U0Q+Ha40WjS090C500l}`xv+r40jUUOYVV^080cJ30>q)ZXMvu<B*PsH/vq}0Ay00}o$?0-", "h<po0P0B050!XSr_0NWg*70>S&Ne0(Yr0Y50QN90Ffyv%'xb50<ysWtX%,v$j@;l0Y40!m} 0{50KV~N0?R010nbW0M`L6", "J0%^P*Y700Eo200CZEA>!mu40B0HJ%ydsl-20Q]=`r,%/U10Eq0&SU010+RqJW~60/ UtULKEf010&`T40o90T80~ Vee0EH8"]
2	["e~{FN10Ol;!vF,gzh[~xQ^Nf`10St0@Umlwzh50J@-e6020( Ots{b0ihud30z[I@Umh>=q&bO0FEL:0g=20wx{7", "p=60VqfnjxYcyX(/6090UO30,0k{ {)'o(CnZ60zt<fo0-dR#>e] 30k$EQ#Df@d,0! [qt&Qw=u70D[tNbsbCa#", "60x00}}%=n]=;/0pYKn_:60F80#d::nodrp,@ %30E^0V10}BL00800w0$I50Ymj30a50M0300~fUx0@0p+Ik]O;ug0a10I0_6", "10-aR*KQl;}UUs`$70QzAPO<~M0yv}(40R$Gs{duy#90^VvpFhw0!FW30[0't/0T}S imCB-k[f70]w40)#[<IO(", "dcD':>kdqe0 %[:@p70b-iqHYWll'K0m`N]Cwn0BCx$100X0o050T;2000C40HL! Y0Yn 70sfsO~k 70?yglxtAbKJ", "b`10900M50:)O0NnVC{Vvy }30O,)'0aq'}K0i(OO30nDz>Vm10`(&60s% %0u$Sy10~MgEzy#0L010!H;[(70HJG?Ce"]
3	["Om5010aDmytq'?=P*M20F90'y!/J:jAQS>w,$N~090'0:,? [U#mnGa0d0y20c'0m_VC+uAsM30I$P0H10q;?szlx9", "[R30[x@S{:w10]t40AL#M>0ot00{Y30q0@m;y]c'(FWL{m]c70r01040-%*MZ&00VO %lL0:50)V0f+t50t0eobg?]=/3", ")neX`uQObG_^nBP[!00[*0+0Qgkc0u400300p<s$w)tJ0 Y-60HZe*[-Pik@0_80?0+RFB%v->d200% $30MQ0t60804", "ZU~T^Wu)R`A&<{u90Dj sP{%0#jVe@*F00<v60H0u70$Z0e:Q0B0iCp?v=YIU0fU^`yzF0U+^0o0Gb20Z70Owa[80", "Ez{KIC()`o=F40HA_60020*q90IFMPKd=a80wCoMH_0m-<$05070'BL<o50RB_0oyXtGBOwiKb]Vmh>;0f{% ieD", "20{V[&} =G{Cuc90l/B0@J20W +{/*U00,/Rq*d70~+b!+*snCN80pPm(y0Q=_dU`)fCPuGJrY0Ky90e$0#i`!9"]
4	["Uo]00[]!{t(+mdt<o`n600jst{['&0wNd(`b020tR[0v_s(;T_@y0,Yb+Wd:m[qS$C20y90A<0K#qi80H80 0Wt20:", "P60J-Hx`%CY$70YB10o:`00&V_LPO1020ssB~0%50=>~<_70C@bjG_aCN_900? 30QKB0dR+@y60HDwu)=GeR0novSrm5", "&0=90DT$r0,w0BI+;e90bv600{#XD0JRw0dH 0g':} [30$<vK40<A'[mJ90;fb70 '{S20LNBs0W&$*0/C=t{omFU]0", "30_VA}N0RTwZ70RCQ&v*, %c0*UY$0&n,DtE30i_+jcQU/G ent*20/TUkG70u<30Y:wo](<&O80?KikqHE'L-Dq", "h0#vvzs,M;>W)TI00mouW80cOW0:? 90m0DofP+D,K]*10SiN50abOpq0ELnEu600woeUf@Wh^i+s' dH?%Z< x<", "10$90z-?x0mEWe,10mSzQ!PUK@Xr0=E<30j40*>uXO10?NHI@T0mhl50DT30b50<icu@(*OO]jS0K0hBe? Oll$lLaT8"]
5	["(xR-04030`x0_/0gwC0-0<20&GIY))]+^lwT0orL~G80&0<OhYSKg60y[,D0d0sEtdoF!'50+? OL+xHCdt0[iRx<hZ9", "@04040Y%K,mQ`0060Bf?0T{N0L#AxAE$E'0B;+!/B XX0`0++xLh]wR %0ge<fKVbAwL06060Jc~P[O?Ya0SS0J#?b705", "*'is'<O40bGy^g :l50LaRUje-Lbv+-0VVIGroOOx80z0k!T0cT&+0 )s0=N]0RRQ010v{KRA0FWb`M0~%*_m?;G", "(m90_Y*svI>!gu:$ {A[{un-QzQl70$!40cSp`&W'm0zYn30NCaauXyhokP~bKa?[{0o>x_'WL]ezS07040oSj0L", "&? =th*zl10Yb0a}>50<0@lS]10G^J{20kwuv50dPp0:%c30Z*rH:M0Pcq0@pM60&q0j0RNDG0w0l<=g0 )Y0<zE(r80Dro", "100*ZLvorX#0HfbU,LOR040CrY`ft~oi@x &pLA0;^es0/60?Ws! m300dGTEHO90-W$/40pOv40ID800k&0~!+!40n#d0"]
6	[")A@800600_~yq50>O+#020h/0_? v2040^l%eq)'0EY]CDz60E70;m !KsbR040 D<c0xm0000dC/jQHACW(/60p<fmQ&", "Ws30j$H0^-'#AoY00010EW40DX0xE0q>0'z0*)fp0Q>80030Yq1002080D{050?m]<nM! B`C}vegr0^Jn0rNRM0z0VDB?x0o5", "t0>qefB90~]H?Mc 90oc'G0$y90C-10MrG0/V? XfyO50xB900$80_yi$yC!kOGK[IeB0]a0Ns_fm0d3000`0;yc0]50$`2", "M/I0*0&bN0,hsC];;pjSMU60(=P?tJKnO-iIEtV^#$A;90+0@<ap+LZO50zzKjoiVYrDL0BW100BGcwu!i>-{)", "#70uXpPUa0l70(R#070R[/`dA$@%&N~{ }nj$*s}]IooZC0Gkmj>U)80J#}-js}+jC0ob#?] [trAQO30:Ai]u0q", " _WnU50@0]50sEY*m>30b$I10iW{^4030Jj;20LxBGET$0,E>aRdJC80n00UaH b500]EkN20'0K&rO;?50g[_*20@70]`P7"]

7	["urFq?{wUk/k^;SI&u0?0x_[`t+_f z^Qo10b50hyJzjQ(B50AV0HIt/X^Gwy}n@F>'u/v[s`jtsWhPJE<` 5", "A0qg[_#! >kk30$Q@mL^D_Ky0EBUo70wT0j9040jx}0TRIq&u0lR<vV%M~P YpMJ60! YOo70H?;dcQ50KB$0v0AZwj3", "#0V00Qpxh0`buTls_}0_Swld;_$k~G'T0@~Hz'X})%=AIw^N^=o %00G20?!Y`ShP90_];S070rN/`0700<0%/{/0cD", "20V:z20b;O~`)hI^t^e/Xh+_V10060Mu80q0nG$uJ^030Q10a$iZBQ80~50^20Zs00+X]}F^m>wu*th( 0 E40d0YSvhD0N6", "=-600vcm[)80g) W@cS/0f0v50m70-d2010Mn070ak!onQ0C0scUssb fmh0i0a`030CF80u80,/v60 @kw>g{0m0Rg^q!C~", "10~0;A0+t0w]0L0$DMx60)/*V;ujz~T{}AMl,60xl)(G50elOAQ70Umkg'f/_Jn<20rWg70i^70 Ui0[lMGk~ta80-)"]
8	["+[&m0$IDd[RE($(f0uSrDq_0O}[90^=x`_w10T0j10r*'0`h90ecL +0x/#X20/Y00Tm60BYv0<F60{$QIr^},LLwK(", "10mUkk6000?;E0Qt$`E)xr70oe90}80^0 Iab0Mz}/KiC*a20wuxKo60;$NcCX%00'U^NL/odq00M>70+0jsKSZ50v(40v0", "+ovXK#HL@_ %080Iqwd30eM~=0N-hu60:x900T{Mp!*lqv:60&m,bt0AV*UZf0r$M@0K[0f0p{00_TxpMX)S %u0=b[>", " s0gUz70mR-((%kOW0;900HC$C_ya:r20_0nX]w -V^0b0bm!EOXI^- <'[0Kg+0if0%Up}uQ0Gv}C90e{jw)*q=0*", "@040&;Hz!0teW~}`zhEE y 0+p?]:h&E0[)F]:Osy=(`z;00TQ?=Pt@/E'80OVa=0000{QC`#Bb?50X10Ftw~O90X6", "=Rv X)&cRw=V40[+LYc800^0V?=RAl@q90D:-*E`h 0:Wz)}!]nAr60%^YhD/vxq+p%`}V0B30f %;dd0Z_r(m80S"]
9	["F^vzTPVZ20b60wGol0_vI0en}i}30M>`(WdH060^e?K0cR;]GD#/F_k0V+ezl0*(@~+0t20VA/ kD<lv0,A300<oB,", ":H(206050N+00E0j*HWB*Im10ZmN80]X0uvE0Ld+0dyLu'0nI0TjQ40xO60}R$Pou<E40/w0/ KU+nyHv080B0wy;0~50yo1", "r[30ojP^&BB90A0'{0eYf0Ku30MFj>R>0? Y0tf`fk0;DU+10Um:q{50@e0)[HsEH~FVJ#J$c#JV^]+m%0wV+pm9", "Ese>C-30v`r%>l`N %T70xuA40w0(L;(TumRpOZ>Os%QM,U=0I^100ty=W+z60w0D<`a40ANe;xgY,50:90gd>00>]7", " GYI@lA0J%20D+UmViS;*10t)@toZw*606010AS60H00oF>+Bwz!{y)[0AY/00<@At060t#F*Fub0z 30R;(!sE^L0L{", "mC0,(L+iurz;'c080AW0~0X80ri`*y^DD`o0*kV,X?ah'0(0070FGK!0`v j0tD0{C0$Ceh>qE<R0=eeKtvSpJjm"]
10	[",q)mfdY-400-U0IPi`Li ]03070QiZT$0uNtF+LX-s-}0#70KFd=0YAqsctCB10QE)&d?jc90A]) (0z,IMVL00Mqip70", "KN/[<Q040<^(rF*?K80O+050<bAV<8060*[-- k{o20V('f010P)NqNix0pbV'Ga0V>E~@sI~a0QE0k^@PTu+<<x10SI06", "~Eu,30q0',_K0L50? FDG20sqYF&;T0&v[$90Zkmo~$k`:aY=10(<g0PLSA@H{u %~>50bTWjTQ=700CEv010FXoABI9", "wD?m8080R0m(fi20~jCU80Y80%`+R/{o? Z80s=,P~10(#y:%[3050Ny>q_70ic501070HR*AH'CW50#*v0}nIA,R *Xw^w90", ";90`pNDV0KUE)70-?jTo$0d80mlkwk0`+0K30~$!0zJDV %skA#60X70t0t#aKOU>60c0W^sz80>##-0060P:`zp#]KP,#{", ";S!]QoA0XJC)60J sySf- 60]ZL+20Jb]Z@tw0V+w}`Ms'}Sa0vGC+{%40 P'Mi0BHo cLz$0A$r bbf'l~IDHd$"]
11	[")QO ;*oCRR:D(@{wkCM0X=050[x60,):h(BEUb}ymGvU>_*#aG}`IGMh$N<70%090&U,h#Ec@f $00%L~sR0n300R", "?`a[O{Ij_gIX['~HV=a-(c0gszh%_k0qaeBv~L?{O20IJC;<k20[q0Jo(J20- 0U0Z80P[!_^0uB20_n$0R080gG$P7", "+50 suU50G0m+_e*x^eF,cN&0e400U?50DNRvz %]]BEoFO00O40mEw,uL0KP<%zAB~?:0K(#l;#40M80=r40uJ+O`v_", "(~nS;OxZN{q&d /'I,:ov+ytW(00ube`z'C0WVM&lhSc40q}EB0pA,0E60g0B{<^X1030tR0k0n 'xqGv$_800i10)", "t20=:)00#80$SF40QDGmaR50UTw^S'JLN90TF%uLH@]- fo0rj0UoG{500`0vs0{fVKzzVe50%0Y-Y0x0iNQ,0g0>mtyb", "%(lpI/j~G'x20qE0uL'0j30myU %VBfC'i0U=]70C:z}L]DlTcYO0m00d900JUiy}wo0LNCzF?YuUXybMU')M0**30"]
12	["&PPo%`s0Z'90_i,EYP60Y]GM=0>cmry!60_W-Cr90yp>Th400:0b }EF)m$sYu)/aX! S`#=c`P{0VW0#kq10:_{0~", "n-S4090U{0(_90*wUX0$~l0AH{0!+< i40f#>30NT9040>*0~zIOchs902030 G?>_W00O_N-00C#?^W>eJ70OzbA30Ie{w0", "?/0*600Z_0g4040@@o/A=:dK70n*0=AYap0l={0X)fO<AvFtl)20T10=h&_I0]i]Xw0*R'XRzS>/;giq T ;O)0GT/&", "OLN90Bc@baT0]:r=Y j^ >Z40-_$^TO0WlD0@b~Ibx*r#qD@#:r^{vN,:80/{u`YOeM0w70Iu>b103000T00HJgc40", "#TQLFNBrouQWs0O@=n90/EBpP/90>8040~IN00#)00RJJay 40RON+S]Iw00[)!nU`KWRwrqI? fF%LH40nl sO5010U", "30N]aL`h0Jt{KM`$_70^}Z0g0W0gT^,x+TgS*20,10x=V#GfdHuQ60XG=QuC10Ljmk]S^Qh~ZW0 N200Gd80d0C=@60d07"]

13	["I/hzx40QG$(QINyS40800CaNCjp&00P0s#El;Q0iqVpPxV<(X)EL0a20Q %q0<zG]oa;40dt@0x0pc`<aJlt+qg@o6", "$=~zHG#DZ?Q0WFeph9040c<800yh<#VJmX00>z=I %G/#I-0RRu40ddkJ040k40=0O`I`b0h>vn40Tp%oxYtbxoL/YR", "? beG040r(BbA*o500jo{040D3060o$J100i0goj;paIz{0Trc080{=ba?W#06080J0)Tz$80}! l0`;>}~DhG#rNpa'$@90A7", "S5060Z0,70}iS*60i,;50)0ix*20xq]0C&zXB>QD80t)Pnqk-w90Z30t,]>- n=r[}Un50'0z300~<mZch0AJgTl10qR0m]9", "?MO)`$h'^0B$)C>T30]{Q608010r$h$S% (D;v'>0/0F}}0n40jINWw-z90e'{)@+dA{]Vh0`EBv&0iy0)0L 90o!?t8", "XR00yY[p>!JI*ZSN? _R04050zWAh=yG0DIKeXNt70smpkz{p0j?0g}=v9040vj:Q<W#N80T0hGA-xgto@i20F,T&*2"]
14	["As80=0~90`&h30=60>J'DPMKD[O0Aj10p70rF,20^mXF!60?wA0_%k0x0-? Zw[[40<hyK20XaZo50!!C*0bBK`/,>MI'G", "?=YY*iPKI<60wRL*(kndx! nQE90O20^80MEn<60fL6000?;lJ<H80BvsFw00o{!(Yfuk00/?yQt0<]00PE)<20lU=I!9", "10D}Sn{>0<0^'V20%Kh40lq$j=80 @w] {{(30Mpq80q0TQu>`a0ry90_R0]RQ0Ue^<4070+H>yQYp:+T$]30lwuB-rMr0", "30M00;x- 0#'[F&0YO'qAoBZeypK@<=30sr{,60<>)tqF?*Jx=20VmY0ufrR+m<in>^%90H#>JK!a20U@- hm-<oSN", "(Jfdb@CRB}*$0u_20V0S-Hu-0p4080LcOsv50@20t8050f*cZ0T/W70C]{x$qUcS0M$f _40l#kvG070s@Z~0=ofW[%0P4", "@BOF@Th``'ekk0500!10E`H'80oY!BJ*Gx00/f#*40,QcuBFj{80%v030Gb!!?;[b;rV$0,0*BLyI?pA{! V,}J/(Q03"]
15	["+q0}g0800wX+iR0gb10X0lE*XP#kN'%u>fvE*eFjp70[)LpZMSida()BD]=+pAGH90EZk^HY? z40@O:E[Y0{p3", "P^PNM0`]m^j020ePmzgwDv*BS50~/^#v$40Fhz-^>#pYxX/00LL#GHRWW/l}?Cpe-_0uUK)`s- Y;e&=dSFw", "UJqSF0MJPKycXUdga0'K050xOg60x0B`e0i?0l30swW&/WV0E30>B&! 80wl;L`e`:t0U>fZsvw+=iP!L0E0wX}70", "i<=6060QnqkP~h90lq'Gvm40l}P^00fpQXT 0+YTJ80tU&20K=s0ljOnU*`N0`b[rTBbHJh60{[-r[at~chRa[3", "BIbc[10/Fv_0R$EaiZQ#VIdi-}Wk0}0>0B'JA`nzm0)-}lNn(10U0S'_sfU0BK070P0]RLhHxk&li}i90u BcX4", "Zk-60!G40Kq30)-F,S@#0a0LyaL(C0`^iKVqIr0u0#@BF,: %S^`(0W=IG0~=W+05090;<Sr900T0s+mYZ60/W0pSX'"]
16	["r80JT[@}20C*eF!50=X30Tc*!0%~:a%o[x70K0!l,M80 60s^90:0#0Y,w,_<&(R)(7090,H* %mviQL>0lio(#BJH0;p", "%af60I70~90m+X90w800yM100(Xs'30)40?;0$O#+WGm30AOLVK,lgR#DkT*+&M0FneWhh*? 60PWOR'qUnk60O(uu0Y&n", "L`EB0D80)A~uas?/0{IRwMI<FJ}[/$ !t^0y0$}- qr#_ilTlG0apm0n#~%?0tl(0^b700jv20 m0;i,aIaj~ _o", "d[s?m*UZ60_Xe0d70`>d80T0!/w`%0? 40em=0^lk?h0ZMrz^c0L'Na)u^SQ50$Wj,p*x}90=50iiU:7050~^!-Mtr)D", "$bka'cmyVv&ZDJ(0:k9080ZL10(!vvlHJ$10[ty40>S<OT/@0>jhYZ* P0NQT@CzvG^JC;0CSUbsowxB0f)800J909", " oIZ%;x@K--20hF00OFX*@CjNdd_? lzWcmk000rFv0=~U#30En80#-x;30kUS8040o0{<-C_~rQD0CI#0Ter70R@0]'5"]
17	[")@020fC'SGg]<v}]$f0i010Y70+}W?T20})NjUexd70}'lK?z60~Qb60gE<-'Q20bc? Pp&i'0rJdmaP$0@tnA~802060X0", "F]0 :_0/T'u~80O10vsx>IH Mx0,R>SzPF`M^ 1060si>XRUKn0@#+60{0v{A)]liVOzf-&TJV_$0PRS0d0>(40000yi", "Da: [KKmq*g$BkJj`ly)g)~~Z70}Vk'80e[a/40[[WA_%yNxJ Jp0f;oDzMNrVfDPq$20KK0&'d*i&:t}e0! *CU", "BApIq=ie'_mK[xT~00!*0`(30c_V0]{SN{&:t2030Uw0~40v-*ARh0QV0X $0w?N %0i80SwT;sa'wbBFOi='mA-B~x", "TxL(,VqZXJlB~;ygq!w,aTDn'Jw30+ML, [P&f700RkGujAg200%20bFPpEE50aL%'<8060?+T;30$=Jw010EfO0/hPq", "Jnu=;eKU#WSDMoC- _s,S[^SP0VUkD0j}v'MwxqT`u40]QZ>a^)`_>^'0ij_0bY70TeIY)n)RJSi'P_+FG70YG7"]
18	["80{10wSHB %FE{10mX90jd0QZ=n0#@OzDO)}B&KSsy0{Cir*}P<A':20_iHAUUq0s5030`0s0hPhJk0wttaRBG +lE0=", "& ZJ0[e10E}H00 W0M0FwndH#t30G;yM0l]yKA0CIx?01080Y}mJG!AvO>ks'w >u`W~^0g)w[m'x)r^>-%d0`HB1", "$sh&70xL0<E*_00AGmpHsFnIlVrTvE X%Xl$K/IhKRnjViyZN)#90?Fj80J0060=0060{P%N0-rf(([70?^E^9080!~", "&YrqVnc200T0k}J0[^[f#v050xkV wTJH>60@]s#dM00LeU0]rARmB_Jiqh10r&'wkE! $0Cz0uu[wm00Tn*-{MO~30", "):abZXUm0u'40gWDy$gV@[Gk90e-+50v0D#>80'T O90'/YaSGE70[a80?s`PqeH70V0qMliq,0lh=Ibx70U[!]dA]3", "@1010-0`T0U{vtil@YgY,- #y$10wVu0>s))/ 0b$;050)M0%)0hAKwj<:80qf90`ShXO00rKOKG:r10<}0:ln`o(ML 3"]

19	["H0'Q)m0fU0/[rSArmFATtuEd/;vD7000YYZKi80cJd;iSW+pZ:P]r@50~030vURH-60-L60 C{_j<0M}08050ec0=;", ">DZ!:I*00Xf0,UO0{k&XS20(I0fDj>)N^40<qz;kuW&q10IS$80xm/)m? O{~n900R(KO0><=R700+0Nm!dG_00'H03", "^g=TQ0k,qh(KT$/}AYf&+070#j0V_y0[b10l_uH %b0jcJ0A]OPL!J090EPE80/10<S0=)b907050mHb90P020)Qi80aja", "30>'0Rve/-k$TLO^{`F0eT06020!{0VaS0,&#O>0'sng^}?%%iJL//l0-D,Ye0DI%~0{Aw/RTb0Np0p}l%:;/f", "y0+FH00=R50Rf40<]r[60pS$zioL#v*zOti)-QeSVCii_Uv]60>YJjB=ao*0~*G %B{Z0+*KOoB~V}30lM_DE4", "UF_^}y~001000nfc70vcF10kEbK}T0YRr4040f*QWz)0ynj=B30kZlUf_x>~d50X[200q70+10'%L600f N,10T'rt#E#Z2"]
20	["<?{00{Gza?BQ`p80T70Q&%YEx(NQ050&cjY;t@N60g]u0Tp`bVO700CCnl60 }CLc]n80M? #D20r/>s#040CqyGIM8", "5020lf]W-;C*:=-x0yuTU,@n/CGu0zMyBG80E70t}0e0oivn4010Fu? 3080I*Ae50MV30s{,[!Wmm%10+lI0X80tu^;}", "!0#R60/Z_0xOFR^8000u0X40Fg0<e0 -Dcg&_s[0hQU=$(t^*20SW}E}?E^@90*GM%V$X]~aCSKiV60AA;uok%3", "Z90Hd00k0r]xay#_0+j20+Npe90!0T30}B0QoDOu)^20%pE)ANN]LK^b80?jrav0@0?b:Dn0e n0oqe*=40_Z90#Af2", "$0<>=^XO]gs20{P0r0x pURrAk0xGz0r<w0[w~vN %]'30}<q)0Ji0>050c=20O0U0X40I(80~G[<n>>}0y(Ft}qE=3", "/zzC}UAtR00shH#QA{?I! Pu70wi:0<;+O%R80 KJe:SW900!g;A30pO0z0S0vHAe>m90~30ezQt0100TWVyIuaOrf"]
21	["500=R90Di*Tq30>(%F80T;Ov>olPvN[e{Pz~K20}60U*+000uPXj0g0k[jtw30+jhh_&I40*ew:0)400 900[#0pr0(d>0/60C5", "FX_/_0RSe;WL80?/0SMfym500``af_f>#~j0:RyeGNX0B3020[`s020~40h*`ugbsAX60L`NFG10ueG PNM;pyStuCm0", "(zg`>IV=xg90uB}qd:<G)Esd*0c[M?EReGESC'GE[`+yL/`Q-@Z%A0'M50}) {090J@900n_Qcj0xA30VfM/tlJG~", "N1000N=YDxNOl0200~50+TQd,9050]70900m;C<-1050? P0X<ak,/}O=50@0`G30=4040`M,(M;20u< l!@~%aU;0y?FklxVze0", "KBL/nfm$0`p^qNw#be2090Mn20mXm>;600iF0}HevVGKO`0a00XJQ=20-auX70c*0;T@Rk0~;0,}0B?LOMDc+Si#'U/2", "40OQ:G-S80?pC(P,0&$CGbH+;g? U{=Wb#WV00^xVXiO80CHr@0[x]I0)0EUTYK{00S7040b0L000vZM*>`muU400t800oZ8"]
22	["K~Mj0TP]80H=@e0#[@wx+-rpY0!0>=Kj:T10T00o;S0+-/VuUt/ CPp]O0yLLtfh00t$0cIX]oE,g30nx',_+0mHW07", "40z0s+u60 aAH]k30d! Dy@J0vnJFfdn*dOAju]0^z800IvxF?s:r050ZKlV0/h$#90INTdh80_u0/%yu])Tk'Y(H)Gy", "H0iitU50600[Y#%wiV?uGj00EZ300'*BlV70{! QfNK:RF400vD90Y0~b<noDjZT_Gu0)tPF]i;0Q4010BhJ20u]&/;?}yj", ",f0P80f#}#0CDg+(S0u %=T+{/W0V/&~0g&0*BJ[t10wIX$=0(<rq/20A0vC40mWH30Sh0MIdbE$'?J<Q70znV{x0tk*0", "B}b{?/*WMj<A,XAMi100F{Pc&}!Ys0sznr40j900rv%,EDPgnh;20TMeW] [0l+AtKdo30oC0u+k~lS0vZXa70+j:B0", "/wEI`oj0E!y2040)j070kb2040eTjs#ZfI @F30A%}0! a<h[070MPnE0000B60'qPjm#00U@Ck0{EUNY{O90h'Uya'y0Vp("]
23	["x0Sh0h tYE?0EG}au00h-0'L)u0=(MC$=+j+10y600@%y20ZT70/?010!80&picp'0_eqG WyQ]Z40yT:0z`@lIA{vk", "Ul!g@nC[0!fR0'}=A?F,)$}_0A;W ~kO2050+m80>=0igyN{#0(0Pkym=^0i,GS?rK703040*0c+70KF+mU90Z+a0+4", ")0'0:g0*eW090,jbYFJ?fPM`070#20_?+Q0x=!L0G<!t#k0R40NqF: {]qD0rr+M80D0I~o<090'[7050Q!S0WH+~%uB101", "<XTcbMqTiV0l'LOR00>U/0{v+);;iQo60)OP$ cWg0:0A-QkW='+cV?-,,H)10%pOY*300@nkP!/w;0ECk@ia8", "h#tLzxl'Jlqj40j-40050ul_,Gq1060fS0~20;0f70902050nPPX$Y@Uc)r>yAV]Gn0NDE(V@PPtQ}U5060=!^y!070^Dm", "?--!>f0:p90X''?CnZyfm>0dU`H(/PRc*M0EsW]>);50Vz70**h0;Gx30S GO0PxP30e>Y0Q-/JxV)]&<R10I&z3"]
24	["gj<jl40Y/FP*g,G-c0E;<jgV0X=)#g~WKRsESE%Y?YnGZjd/X30!ShP~>0PNza$ 0fiGDupL^Tv;? pse)60NU", "q;XQF>l10eO0n0Nje0U%50r(&!0< Pfwzk? G20cR<{MxJA40d)b*a)G`ITm^tWfBuzie20a/;[oWv_00CVF}jJZ70", " 0Z-/ZK70{&050^ue^TG+>lp/{X;0Y #oo[L<0T$10^&iy0<r0ScFhcJ0n80DYnv0A! OHyW>50~0jI'lO&qoVgtp<,", "?lH`d; 80>v/)H[=O10 QvML*DCEq+`10g70r40) _yS;ApOi;I40f^0&vu30-0+]F60U30(C:dSD!NjOV[r<0Lax q", "C#{l00X<YY] [0XGkI5060L;0F80AK@Jg^;@30<L$~I00:/90?Vso00JGeCoR30q`HJy`Ra70-0$0~TkY'c:lA,A60Tg0$6", ")aG100vA]sbk0 :uG0O0noR@=Q60#, /Sa0? 090d80kV}60qH)*le0Vva+KlG40M'20Y-<&%?30v00lO5000F=20JVC0rdP901"]

25	["%:,mc)0UEnXq?(70[0E~]WhHyT,l:900WsH+*5040+0'_b-20@-dr0D =`E70XDbeLy:3030TV<WXyDjH'3070FW--]:", "*aUPe[:Tc#%J@0>0>0080^Edi0i/0`^0Nm^bh~*q80m>0guA'QfoW0CX60ygE0E/^O! nv<nhv#{@Ezo50m=FN10do2", "w0e?Sk]FHXM40^E;40jiP$Wq-70$xpqpZ$50&@050H)0500S'V0djqYG(A>Y)/e!NdL800Zn[/0YX!]knD*acbAj]0", "ix&V}lh10V^czgJ/kd50u,,P-K%*AVUDJ&50V+&i70ZDZrk!xj}0h[yE=60?30]]w0[l0#b~K^!80pmp? #pgS0PM", ")8000'flpHQj? 80iy+20AYruSPZzl~<(*yknU@OVJCKcwrGEMi}80>U)*jC)=F_jm@vjdO<;SS 0+iGP:dNA04", "/lMo0FHPJZkj0A20k'R0*nls[G0K20W,+]J0g?{0`U}60mH]O0^FY0 ?t]C010u-i40v0;e>~}s>@60#m10~R100[*WK5"]
26	["Cj0Zy30J-q90;G0>60DCop(b#EvM,`30#0x?;10D^aE0!^;W$p=oxj_A, [00CRhTp$ol0#V:TvFI80CgW`{hr'`", "P+W!LS60Q!Qgdf^=l@F0Ku]60(K90%Fs0@`'0M/40&0J,UTfXy0g0HmC0kC40v[[d80J!]d`l'40c0+n 0;KzZ-30=2", "KC/eHpK`%'@aLZ,M]T,k0*<,O)-N+0l?D*0d#,+et010NlRs@%s_{B$ik)?a lgx! N0L<cdf40>)0aH~JL2", "A0m0xb1090qgQz]jWkl&50$UzA300'%0u40Tx'~dXR60M30FmlJEH20:$T?{[n0fa tPuRA60A U]pyc];dz;c90tV", ")z{DmK$MWK)!090ld(a40rc<j];D/`PglYJ5020KJhB80I,z: H>30]050LbyOpX %Kp%$a$0oT40mMzOkAtM80{b", "H}ZK@uw~[c:!JA0M+v30IIS BhQgO90Deaybt0Ycv(>Vk %u(y0y*cUe20w=DZ&{0PJHZb60~!!040TyV[[FRl"]
27	["B'<ak@0Nk_d$]Z0meCQU40I010+':]K)m^OLO!+:ym0'kR-20hiDP@C{P> oMQIeHf}=? fk:0rA0U=Aco004", "H_o>r,:N!;0MsXBMt<J(80W0$uJ&'010^%400f[30tp/jt0*Uk<10~>~'cSwM50es40! 90}aXIw20$0sENJmH10`r;2", "CW>mhynRS20f0}0[0G0qX}h w090l9040N %900:YsydZj{[q30nd;Enlsy!y0U+eH$zVY)-C#=IyXP0iJ^Umi}u2", "P?00*0Adb*0al0GUW:q! O>0kTMTxk>u0;boH70Qq(OOZ_0,z9060^aha;MJR90mpJ80VD40G0!?yTMeEY;0GY!pO4", "^<FlMMqTp;,[kA60a`wR<C;a050s000Ngw0q0900[T30/&TG&<V^E030lSE0]Pv`j80_!?k70]g!-<V0Ys>NvY'XJ6", "Gt{fS0(0+{@=0]f4060=([>ljt:WfH70}mH30@w0{&g&k*@800J!uc&wW0! {50;40,0}W`80yTs=yG0Q0EpQMH_0hS0"]
28	["mNL,o<x030[U40zBS*Jj&(h0Z0w&+@90d?T@80[*mW}MLO900Z}f%Gu*QgTA:]Tj{fH?[@'v -,! 0Yk50*}hCO508", "s]M/ynRykp&X<SN>30T0iiY@40V%0p?TZMEs0LJ0Zz,{20v80? vq$WiN,WF10{`MY20lc50y0sL$-T{RxY#uer&D4", "@0e0z20R]F!00i! o~U(j$I&u20)m0W{durXHRDE0b+Sp=/<aP<0Y%fYS>>HU?JW80f0W80Z+00Nah=? w+klEC0~0_8", "#s0*_][fvwYbk&+,=@WEP]zkfdK{PKUW@rCGp+p0u;qLCqni0l=W[&=0B(0?-nx^;10vm+L0`,&10l}~QIj{", "%L)xa0J<c&100gJk20Q,K0SA50D0 ~10N900{[tQB60Ojmd0M@R+_Gw0%sUoQN,FG0l0TgBYj]=:Kai0/j,0[K50W,t}", "10nQ<k^:~KNF'U20<anohk80s4010{h=20!k}050Pyq!+oa%0V}=o@IcqM{&zi<d$dj %0800Z0r??]Vrrr=!^^-Q6"]
29	["5090CG}0*e/R80Ge^0HSVNYj0j_y`d ~$PJ0TG0t* f?-Pc!F70@20smK0~-RJm0* %ZO^+}0y300*f{<g}tDd30w#40", "`(}/0LN,~W<'P0^GqZu`>;OU: w50%XqGQ)Eoq10r^E30dZ0u*00M&-=eFV`gL^K<y/H$HCkY-ci%'[0:%Tf7", "e[vZ0FiU)OTVRC:?-y_%(y+00030;0)SI0+iN60{dgHsnhdP00400:oBDp100d050S-sm_0RNd20sow]? 0vw=euK0B0U", "N80'tA&0D;sD%lY{IFLJ~<>/c`80Er0aMJ=Zt80bywHfB!k?[L)Y0-QVE*/u<<]KLyjX}WSmtNo6 00GlwJ0q%", ":o0$kGX60X/o]A0e30@]~Sqx_U0sovG0bhca#0j/5080f p !0qyo0+t*r,h0N$ {O30(10#'jo=MJAV0;<90F#x(V8", "10&c20^wBUVz50bLudF60@300uS0fCV#:10S10JLo xl>E30JfYkm/T/0p0:LEfcF10P{m*0b0U,DGT~^p0(e+=+80oU1"]
30	["OO#0800~,0040nf`^70gLsw0 90hBus^NA90V_0xHH;20Qb^ab5060Hn0OGB)!! Tm_50500RP<`0C3070Fg@ZQA<-50f#^,", "N0,30$JmK0Id0qE(+t0xl0@-Muh0_TtGO#tLp? *A;,GB?DLv;WhnN90%<Aq^500U40a-wF080n_Obu%m;azr20T", "`j>`'gYp[fcLF,M0:lC{90s]p90@,}YLt300B/10/vuZdNaTTL}70ND_vt20@z-40~TsaHL0xQke]jVsu-g-K)", "&YMJ:0o0]S90Fa;b0>KX+0_30w0 0FB %_Y20B10070X0i50603000pg20CHqcR70Ako;C90<c&>90y'<0tr&90,?R<i!Z0C/9", " A#10+s0g-60e'w^@0^A%/s>?b-50{sw]R_e)ffI@CJP100=hk]:kFd70zqeMi>%0=b+007010fmiJH[0/tF00Iq[", "500(nSWyQa20t >K0XK/QtV0KTv0'*,'_*90g+~-Kw0DmU]MMpxWPKjjIsx;/! AVokFVr^60}w^r050_;30vtO7"]

Annex III: Source code

[1] main.rs

```
/*
=====================================================================
NJS: Database Protection Algorithm
=====================================================================

The NJS algorithm is of symmetric and non-deterministic type. It can be used to protect information in a relational database.
The acronym NJS is the acronym of Noêmia Josefina da Silva, the great honoree of this work.

Designer: Edimar Veríssimo (Yugi).
=====================================================================
*/

use std::time::Instant;
extern crate base64;
extern crate num_bigint as bigint;
use num_bigint::BigUint;

// Use another source code file [show_mwssage.rs]
mod show_message;
use crate::show_message::*;

// Use another source code file [read_write.rs]
mod read_write;
use crate::read_write::*;

// Use another source code file [math.rs]
mod math;

// Use another source code file [intermediate_file.rs]
mod intermediate_file;
use crate::intermediate_file::*;

// Use another source code file [cripto.rs]
mod cripto;
use crate::cripto::*;

// Use another source code file [cripto.rs]
mod compress;
use crate::math::*;
extern crate rand;

extern crate num;
```

```
// Configurable variables and constants
const NUM_COL : usize = 5;  // number of columns in table:
const LEN_BLOCK : i32 = 60;  // size block in bytes - max 512
const NOISE : u16 = 257; // range noise [1,257]:
static FILE_PATH : &str = "/home/yugi/criptografia/NJS";   // path of database
static DATABASE : &str = "database_min.txt";    // filename database
const TOT_COL : usize = NUM_COL+1;  // don't change this value
const EMPTY_STRING: String = String::new();
// do no change this value!!!!
static COMPRESSAO : &str = " !#$%&()*+,-/:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{}~'";

fn main() {

    message_header("NJS: Database Protection Algorithm");
    let processing_time_total = Instant::now();

    let info_data = read_data();
    msg("\n\nShow Database Record - plaintext:\n ");
    show_record(&info_data,5);

    let encrypted = cypher_data(&info_data);
    msg("\n\nShow Database Record - Compress File:\n ");
    show_record2(&encrypted,5);

    let plaintext = decypher_data(&encrypted);
    msg("\n\nShow Database Record - plaintext:\n ");
    show_record(&plaintext,5);

    message_header("END OF THE PROCESS!");
    ptime(processing_time_total);

}

// ======================================================================
fn read_data() -> Vec<[String;NUM_COL]> {
    msg("[01] Reading Database: ");
    let processing_time = Instant::now();
    let info_data = read_database(DATABASE);
    ptime(processing_time);
    return info_data;
}

// ======================================================================
fn cypher_data(database : &Vec<[String;NUM_COL]>) -> Vec<[String;TOT_COL]> {

    msg("[01] Encrypted Database: ");
    let processing_time = Instant::now();
    let cipher_data = transform_database(&database);

    msg("\nWrite encrypted text: ");
    let _result = write_database_encrypted("encrypted_database.txt",&cipher_data);

    ptime(processing_time);

    return cipher_data;

}
// =====================================================================
fn decypher_data(database : &Vec<[String;TOT_COL]>) -> Vec<[String;NUM_COL]> {

    msg("[02] Decrypted Database: ");
    let processing_time = Instant::now();

    msg("[03] uncompress: ");

    // key tuples
    let ichave = controle_chave().1;

    msg("[04] decrypted: ");
    let database_encrypted = inv_cipher_database3(&database, &ichave);

    let dados = database_encrypted;
    msg("[06] Write: ");
    let plaintext = verify_database(&dados);

    msg("\nWrite plain text: ");
    let _result = write_database("texto_claro.txt",&plaintext);
    ptime(processing_time);

    return plaintext;

}

// =====================================================================
pub fn controle_chave() -> (Vec<BigUint>, Vec<BigUint> ,[u8;256],[u8;256]) {

    // Chave do sistema cifragem
    let chave = vec![
big_integer("798599428663203109956158163914448953869352662328770360215573629792670326080243059579885575172159941160996858791492426951050125434098259893785644324473094",false),
big_integer("389497126888089591881357186534982077698653556404469446893882300647692836143353197228390384194426695893807917767545151499602887246947718593727448997248475",false),
big_integer("1652692441615594879688938721164988030021629470487536871620129120090230908908754745799804672338033800526737708514965500441016077256195537282122354800594924",false),
big_integer("10722467218978268926258696226747927491676137674641045614263412653289622125863632210496158644295701991444380932249987824423920733594443161811202603577008773",false),
big_integer("39413812512964323912403853725183783898896681925446021473675597733182422893620651463756524427867057433949051072403008266029995769551330017255332283664164 98",false),
big_integer("67996803281450124876847639139003275673833921727351713704970445572392554546964057582872584485959867264193633127197928960011887093710072095406629492777 6550",false),
big_integer("38352481090697716571512135318001582635625757665255622429886488114724037323996929576536087685041738632148724854352402094746475307553705127312603671401 36452",false),
big_integer("14131977366805436785561604598983193438661364711763400248572879753402250305836809006390647865165179066112442451653522369947632069276109866770003652323187791",false),
big_integer("87011843762786557513315569641552392579511862695230308266349920311066131148932030810220282349919954764524709201663041391724384125824642323960444197172 39034",false),
big_integer("13401271165779039677132077429203632328263144769874312970068148876314128392349501785954083025823662536353054832141092803043131351501829397794476306697797200",false),
big_integer("73650558202136522578961847987037021097186454439664812513820381124745401730583432481268162010856895211961548093910562479625836258315197828110097774202 68488",false),
big_integer("62751567957182121908589996037575700747066488676349468457385537869782914085904630220317018343273624072066926036275655625788047574496799302455205246748565509",false),
big_integer("10698837509712721187060274938427529457033205923188779127803852131226308952795437728533640995254202381471771225497758195021909467300382860305529071296902 1149",false),
big_integer("75129981882726652513362504392149118445096149084786096790542619953435329833067142665452632011069179661479270769857986754814164098120029044883523421600 27915",false),
big_integer("69788261443455392943670574880623061062023346236205339840506429701634279754909208766209825809871772721626107228624907791963563980572861272445177425858810129",false),
big_integer("56318219381744816544439511826752610591319102858264549263126651054674431571364048378997632707829568950858393118971337802046103511106157171135817570986266 9281",false),
big_integer("31736717882869061863211820649740569561225325736641937500224734073737527715664577709264596934344398472812021708456382115374460639605163224821440242819094 3820",false),
big_integer("33855909614234757713825352827420391274856621863677939168223138347061081927775922134325113386342215410131208216142663993087944486411480822529145799880153 6749",false),
big_integer("89200174180240769003124910785659935975568174525955772008586876752967346230835068674249612931710530071211780004404035322416526103821775522237486401422533964",false),
big_integer("33329743085315113737010539623297528881942999390631349022533616076173198088719102556131655564526399609486058699100037633838013750607937854655542009954619 07467",false),
big_integer("26938523277118634534410681192574934277361973506428318247929441091306392574327569389054077945387493428386677963838908438490908912650044469120142958905580 8795",false),
big_integer("62078941727844029018562430691641600148906615539760974904152320361983552898878086338072462531510574673816071151585782642588086282343174720318096226022555 7075",false),
big_integer("82761951118548026171665188689023929724371871054693335351606328165846865807392013931608453228874826070974684209486659053739844330268075259101763024787701 8164",false),
big_integer("99573602313013214890895192703276583876676791337236341796997645988639569160753269275188196001962156812412252280518325717027839820904953837483027215659633 37207",false),
big_integer("35003842897595275488593485236584007234296697263908474131710746107275869188057085136303883103821891027635755430018929766100260954166463976850859206546361 74580",false),
big_integer("48038991922995433230632041599925761424532221305292720226724449796326861499282267885657359452514744182391960230309710515389368830089770426366552870590181 66508",false),
big_integer("5898759033125983532752367356407877324919289011212141704153050118338785183143884303808618179712495623483198034552686468407
3562683701347664618969348867299892 00",false),
big_integer("3933656746387265049666118771996729532413167151048490823553313207826896587998525810287615318997563739780325596759200399003
486871552606163366407659782154729315 3",false),
big_integer("5832291003276764274912584542249002332379483882645481377083249624756443947933706519124179904353686527271447612249232616222
1013785682380387917028691167154123 84",false),
big_integer("2066735922108930579965107027039035560890318216539334587319610103575954315244288013327766155639207932505253195089353084835
4412226563286479806383704351952236 79",false),
big_integer("8517096333202442940332690116185642973007465316314727214817751099953687312168758973588885647323336295639280105466124593940
4232827347072790760905736370850908 89",false),
big_integer("1764230264928053794525351966751977411888459621670971463219149377037772749714490790650407800659670982973332122763927431915
5223073850551983181023575740098038 6471",false),
big_integer("1350743840345525570672405535289515050958995927045219129786165932760542988030634100700263457212457699324987213515820194852
2253494732941734702363264496241805 64",false),
big_integer("4670853330585986224634221239565662681795626398652845168453922095028494758285096828899646910282913799602888993993225361571
0652507912419720255383696419754095 008",false),
big_integer("8986331316396752568755556382570468251589730807948568409404514787361161087114615075881567307715671527941214640932024130998
0819503775179165765310459810392097 430",false),
big_integer("2651564515119522808445314422186238301668456765221902297305418920284563733614131644928772149787039538153270286866323032495
7719672713376002447115406929616670 9617",false),
big_integer("1776166850128547355308250381402286702428142375633136053575335309203013732769326241064828238693894422710990090082534076177
8046247703007046038902389164354938 4758",false),
big_integer("1230793474584165631423650669394778916977072654525155023241397479030781606796857264364019682641262335527409485069397983783
4203349173588341240324787578557380 2278",false),
big_integer("1908546843003416450869017882450164044677307023623422139711242481462921257352950587886391811725790160012349217867079306772
2742013559622486468137329116507932 8504",false),
big_integer("3347274343780223862823291341646964725903199566549021438196990099331996183105207242477693761428579390816977510197372565370
5356896288370572510245531912569859 37601",false),
big_integer("1629141337980619199831025715730707169578223492545353971276058708777282392368748853228139625142330463138009979138144450660
8712091988349949921551388197527521 39914",false),
big_integer("9661174578763389889674738130456941198420360139241896980169677335047752035128476202210669389311756512156603459521874578854
3444640391970174772374013002347565 6217",false),
big_integer("1705516819451008281453498162591529629767235539374824184166398283502433703472630231229867453581482908661022261923489950309
4105266033889753371579104678577876 39168",false),
big_integer("7190616924772204931865998742111727354826338817900580731475647537380892488220345785109323388049519310522590087131429105564
2608015584225601486819004278780424 27877",false),
big_integer("2092147618492584989282094639961040178419969634169360604014525180704975842901655287638861843967181215387738344594213622027
5855085146452302378096770075398324 90891",false),
big_integer("2382406469335925955486146328997586646958890613976171264198827360447107726150800129222274108165663496353116745052155104878
1714938277993417930086737526556332 85596",false),
big_integer("4250557911571730591394887266346026468949200447757803047703902527869429427417507539439118727832879733822553991227675505750
0806247807590233091037848389560653 41027",false),
big_integer("5450162571470915351367958785180779193093977059610350100538725888597441719704213310647899497120768166845088717299404633356
6119814339928055166169672264611027 110417",false),
];

let ichave = vec![
big_integer("9923867790315065816302538062833478537806785012312275254047839023496951799783389150916273069123542050283384073458495397472
870608160344901917416959252535679",false),
big_integer("4690921015443614285419659973259535820446650757012081778085295702963808885227781396776512034061141177653033055237065260332
117677781772365708940493776907720",false),
big_integer("1652692441615594879688938721164988030021629470487536871620129120090230908908754745799804672338033800526737708514965500441
016077256195537282122354800594924",false),
big_integer("1072246721897826892625869622674792749167613767464104561426341265328962212586363221049615864429570199144438093224998782442
3920733594443161811202603577008773",false),
big_integer("1019059611550900439432121922646481504877169651921879810120531998008400801647474386001499542237847332271753734441322154334
4632492320976865044470423956771293",false),
big_integer("7462540821707941304186888906248809937295361676674909660901749208664788353836468813985156610650484894194739445802608035568
22874234435848619335362737271967",false),
big_integer("3835248109069771657151213531800158263562575766525562242988648811472403732399692957653608768504173863214872485435240209474
647530755370512731260367140136452",false),
big_integer("1413197736680543678556160459898319343866136471176340024857287975340225030583680900639064786516517906611244245165352236994
7632069276109866770003652323187791",false),
big_integer("5405038358090346615725843907342046148911530240682643763075054583867630097101142713929499010828162859561445511610935148661
5609161914335070059160827031326475",false),
big_integer("5516661086871285624486504344116238023395070481056903423221884280786881452215431830741067077563655171011511626316127019690
5021500080324071491294262481378139",false),
big_integer("7365055820213652257896184798703702109718645443966481251382038112474540173058343248126816201085689521196154809391056247962
583625831519782811009777420268488",false),
big_integer("6275156795718212190858999603757570074706648867634946845738553786978291408590463022031701834327362407206692603627565562578
8047574496799302455205246748565509",false),
big_integer("4561938187203209535737923688832508113428589693507577013532279892344812261856861065046399171257536656938662189347357960702
41940438057743108302884996893648132",false),
big_integer("1266289456041149671559921403373450357869765836629430569234606834707134149102480455566357515888976021190284472848519905339
5544938089443755939917310618134632",false),
big_integer("6978826144345539294367057488062306106202334623620533984050642970163427975490920876620982580987177272162610722862490779196
3563980572861272445177425858810129",false),
big_integer("5631821938174481654443951182675261059131910285826454926312665105467443157136404837899763270782956895085839311897133780204
61035111061571711358175709862669281",false),
big_integer("3015607129702820755068935755832347192582046681696715527251114266879944531715264478520519587109195976220485652825439942230
056768664742153217339798567270963647",false),
big_integer("8360775498889477320419581623912947913109776025534662535799262265308909297754731099877279223801286806561883842253021510120
56729182016324868737300420198651949",false),
big_integer("8920017418024076900312491078565993597556817452595577200858687675296734623083506867424961293171053007121178000440403532241
6526103821775522237486401422533964",false),
big_integer("3332974308531511373701053962329752888194299939063134902253361607617319808871910255613165556452639960948605869910003763383
801375060793785465554200995461907467",false),
big_integer("9687974998530135143745412458401909044894059398659350997220470187950892990332051233628278820742340746957358448413443487317
874892963994939057101291976907528412",false),
big_integer("1299279621548368249837273450007052032348154128998589882214917750624902832006875413486579499511852506225864082227915489122
117831327292593168904954837841337349",false),
big_integer("8276195111854802617166518868902392972437187105469333535160632816584686580739201393160845322887482607097468420948665905373
98443302680752591017630247877018164",false),
big_integer("9957360231301321489089519270327658387667679133723634179699764598863956916075326927518819600196215681241225228051832571702
78398209049538374830272156596333 7207",false),
big_integer("3583618317411312294780183919630889460070200178409406082236205746754137896117954958924576487959344829503968042459011101342
48426201094152359789906771669111185 73",false),
big_integer("2702249322948703167144797482030653717992727265082969398452448707907386490346940817270402777068199345002954816333392603089
2816967336219810621844602569321637611",false),
big_integer("5898759033125983532752367356407877324919289011212141704153050118338785183143884303808618179712495623483198034552686468407
3562683701347664618969348867299892 00",false),
big_integer("3933656746387265049666118771996729532413167151048490823553313207826896587998525810287615318997563739780325596759200399003
4868715526061633664076597821547293153",false),
big_integer("1705907354895286151776226121329487388564664782844516649448316880790208310235153725459166001616134117700617646641435105753
3012935993728179301853288828426497408 7",false),
big_integer("1755022705937537223721469017814496538162249330746387431238899644378286087753279483333331042443491729975297852329117185982
2039444934677357420793857327007599988 5",false),
big_integer("8517096333202442940332690116185642973007465316314727214817751099953687312168758973588885647323336295639280105466124593940
4232827347072790760905736370850908 89",false),
big_integer("1764230264928053794525351966751977411888459621670971463219149377037772749714490790650407800659670982973332122763927431915
5223073850551983181023575740098038 6471",false),
big_integer("2638057076716067552738590366833343151158866805951450106007557260956958303733825303921769515214914961160020414731164830547
2497137766046585100091774284654252 9053",false),
big_integer("1919249781232575371594753879564649143778991656966095338521880421689146917506507574632708092519837187275735710245498405054
0735360597096359833510550458022213 7717",false),
big_integer("8986331316396752568755556382570468251589730807948568409404514787361161087114615075881567307715671527941214640932024130998
0819503775179165765310459810392097430",false),
big_integer("2651564515119522808445314422186238301668456765221902297305418920284563733614131644928772149787039538153270286866323032495
7719672713376002447115406929616670961 7",false),
big_integer("3169657658767369127292466303506736055660385328985707832839456568411694809828274618371210937559189948545878501189119157752
7552271518069867906355292996134365528 43",false),
big_integer("1677772935690591878331841466997645726421256858089774386352396769619906915147825331055782444672971075779064328651566999867
1390422349173580550687722163238960061 93",false),
big_integer("1908546843003416450869017882450164044677307023623422139711242481462921257352950587886391811725790160012349217867079306772
2742013559622486468137329116507932850 4",false),
big_integer("3347274343780223862823291341646964725903199566549021438196990099331996183105207242477693761428579390816977510197372565370
5356896288370572510245531912569859376 01",false),
big_integer("5561475586791585732034973026381020185248115325355226760199588828603610095851596931881183762907188847384580107993284654903
3895923595875651565267616081252902879 63",false),
big_integer("5954663124525235688839890939411347347550952415967689320314648174427882623780048428438741634375738396937361308961230100995
0762181533511697121808513841673725651 82",false),
big_integer("1705516819451008281453498162591529629767235539374824184166398283502433703472630231229867453581482908661022261923489950309
4105266033889753371579104678577876391 68",false),
big_integer("7190616924772204931865998742111727354826338817900580731475647537380892488220345785109323388049519310522590087131429105564
2608015584225601486819004278780424278 77",false),
big_integer("5240947809621656852439749321184675175251980096193414040137273370526944135414047781884013312724050045306314882839983271153
8534305825282824928359995257071194619526",false),
big_integer("4460010052071277254502938454182942218720173251186648278691885512185313055740465570251868148563102341777228877436385419839
8244970249941064254119317535006939329779",false),
big_integer("4250557911571730591394887266346026468949200447757803047703902527869429427417507539439118727832879733822553991227675505750
0806247807590233091037848389560653410 27",false),
big_integer("5450162571470915351367958785180779193093977059610350100538725888597441719704213310647899497120768166845088717299404633356
611981433992805516616967226461102711 0417",false),
];

    // Sbox
    let sbox = [
        64, 124, 108,  28, 188, 254, 134, 154, 187, 235, 199, 240, 251,   2,  70, 156,
        3, 133,  66, 107, 128, 150,  79,  77, 130,  83,  58, 121,  49, 246,  94, 101,
      176,  60,  57, 238, 202,  36, 126, 198, 158,  86, 194,  47, 174,  18, 144,  26,
      151, 120, 175, 205, 201,  42,  82,  95, 179, 164, 112, 186, 217, 104,  45, 223,
      200,  99, 195, 247,  46,  78, 102,  93,  34,  65,  97, 105,  92, 115, 139,  31,
       85, 145, 234, 110,  33, 123, 137, 173,  72,  40, 140, 252,  80, 185, 160, 215,
      227, 177, 218,  21,  10, 161,   7,  71, 245, 109,  43, 116, 131, 142,  20, 113,
      232, 129,  16, 210, 253,  30,  56, 167, 216,  63, 208, 125, 189,  25,   1,  52,
      106, 163, 152, 182,  19,  69,   9,  29,  27, 193, 178,  51, 228,  90, 122,  73,
      203,  12,  41,  35, 225,   4,   8, 118, 157, 165,  96, 206, 135, 127, 147,   0,
      219, 244, 192,  59, 204, 184, 153,  14,  75, 111,  15, 168, 214, 138, 114, 249,
      169, 207, 141,  32, 166, 143,  53, 149, 226,  22,  38, 146,  74,  76, 183, 224,
        6, 222, 233, 117,  48,  91, 236,   5, 170, 162, 243,  55, 213, 197, 248,  61,
       37,  11,  24, 100, 242,  87, 239, 103, 136,  89,  62,  54, 211, 237, 181,  88,
      159,  23, 231, 196, 155, 220, 132,  13,  50,  67,  44, 221, 148,  17, 171,  39,
      180,  68,  81, 209,  98, 119, 191, 230, 250, 255, 190,  84, 212, 229, 241, 172
    ];

    let isbox = [
        159, 126,  13,  16, 149, 199, 192, 102, 150, 134, 100, 209, 145, 231, 167, 170,
114, 237,  45, 132, 110,  99, 185, 225, 210, 125,  47, 136,   3, 135, 117,  79,
179,  84,  72, 147,  37, 208, 186, 239,  89, 146,  53, 106, 234,  62,  68,  43,
196,  28, 232, 139, 127, 182, 219, 203, 118,  34,  26, 163,  33, 207, 218, 121,
  0,  73,  18, 233, 241, 133,  14, 103,  88, 143, 188, 168, 189,  23,  69,  22,
 92, 242,  54,  25, 251,  80,  41, 213, 223, 217, 141, 197,  76,  71,  30,  55,
154,  74, 244,  65, 211,  31,  70, 215,  61,  75, 128,  19,   2, 105,  83, 169,
 58, 111, 174,  77, 107, 195, 151, 245,  49,  27, 142,  85,   1, 123,  38, 157,
 20, 113,  24, 108, 230,  17,   6, 156, 216,  86, 173,  78,  90, 178, 109, 181,
 46,  81, 187, 158, 236, 183,  21,  48, 130, 166,   7, 228,  15, 152,  40, 224,
 94, 101, 201, 129,  57, 153, 180, 119, 171, 176, 200, 238, 255,  87,  44,  50,
 32,  97, 138,  56, 240, 222, 131, 190, 165,  93,  59,   8,   4, 124, 250, 246,
162, 137,  42,  66, 227, 205,  39,  10,  64,  52,  36, 144, 164,  51, 155, 177,
122, 243, 115, 220, 252, 204, 172,  95, 120,  60,  98, 160, 229, 235, 193,  63,
191, 148, 184,  96, 140, 253, 247, 226, 112, 194,  82,   9, 198, 221,  35, 214,
 11, 254, 212, 202, 161, 104,  29,  67, 206, 175, 248,  12,  91, 116,   5, 249
    ];

    return (chave, ichave, sbox, isbox);
}
```

[2] compress.rs

```
/*
---------------------------------------------------------------------
DATABASE COMPRESSION ROUTINES
=====================================================================
Function: transform_vetor()
Compress the database - auxiliary function
---------------------------------------------------------------------
Parameters:
    vetor: binary vector in byte format : u8
    mapa : binary vector in byte format : u8
Return:
    ret: String compressed : String
=====================================================================
*/
pub fn transform_vetor(vetor : Vec<u8>,  mapa : Vec<u8>) -> String {

    let mut ret = String::new();
    let mut controle : [i8;2]= [-1 , -1];
    let mut posicao_controle : usize = 0;

    // Transforming the field contents into compressed format
    for ct in vetor{
        if ct == 46  {
            if posicao_controle == 0 {
                ret.push_str(".");
            } else {
                let valor2 : Vec<u8> = [(controle[0]+49) as u8].to_vec();
                let s = String::from_utf8_lossy(&valor2);
                ret.push_str(&s);
                ret.push_str(".");
                posicao_controle = 0;
            }
            continue;
        }

        if ct == 48 {
            if posicao_controle == 0{
                ret.push_str("0");
            } else {
                let valor2 : Vec<u8> = [(controle[0]+49) as u8].to_vec();
                let s = String::from_utf8_lossy(&valor2);
                ret.push_str(&s);
                ret.push_str("0");
                posicao_controle = 0;
            }
            continue;
        }

        // Makes the control 0 to 8
        controle[posicao_controle] = (ct as i8)-49;
        posicao_controle = posicao_controle + 1;

        if posicao_controle > 1 {
            let valor : usize = ((controle[0]*9) + controle[1]) as usize;
            let valor2 : Vec<u8> = [mapa[valor]].to_vec();
            let s = String::from_utf8_lossy(&valor2);
            ret.push_str(&s);
            posicao_controle = 0;
        }
    }

    // Reading possible final residue
    if posicao_controle == 1{
        let valor2 : Vec<u8> = [(controle[0]+49) as u8].to_vec();
        let s = String::from_utf8_lossy(&valor2);
        ret.push_str(&s);
    }

    return ret;
}

/*
=====================================================================
Function: transform_vetor_dec()
Uncompress the database - auxiliary function
---------------------------------------------------------------------
Parameters:
    vetor: binary vector in byte format : u8
    mapa : binary vector in byte format : u8
Return:
    ret: String compressed : String
=====================================================================
*/
pub fn transform_vetor_dec(vetor : Vec<u8>,  mapa : Vec<u8>) -> String {

    let mut ret = String::new();

    for ct in vetor{
        if ct == 46  {
            ret.push_str(".");
            continue;
        }

        if ct == 48 {
            ret.push_str("0");
            continue;
        }

        // read the position element in the map
        let index_element = mapa.iter().position(|&valor| valor == ct as u8).unwrap_or(100);

        // Transcribing uncompressed characters
        if index_element == 100{
            let num = ct - 48;
            let r = num.to_string();
            ret.push_str(&r);
            continue;
        }

        // separate 2 elements found
        let c1 : usize = (index_element / 9) + 1;
        let c2 : usize = (index_element % 9) + 1;

        // descompress elements found
        let mut s = c1.to_string();
        s.push_str(&c2.to_string());
        ret.push_str(&s);

    }

    return ret;
}
```

[3] show_message.rs

```
/*
---------------------------------------------------------------------
DISPLAYS MESSAGES ON THE SCREEN
---------------------------------------------------------------------
*/

use crate::NUM_COL;
use crate::TOT_COL;

/*
=====================================================================
Function: message_header()
Show a message on the screen (header)
---------------------------------------------------------------------
Parameters:
    message_01: Message to show on the screen : &str
Return:
    No
=====================================================================
*/
pub fn message_header(message_01 : &str){

    let mut size_separator = 80;
    let separator = &vec!["=";size_separator];
    let mut s = String::from("");
    let mut t = String::from("");

    size_separator = size_separator - message_01.len();
    size_separator = size_separator / 2;

    for _ct in 0..size_separator {
        t.push_str(" ");
    }

    for ct in separator {
        s.push_str(ct);
    }

    println!("");
    println!("{}", s);
    println!("{}{}",t,message_01);
    println!("{}", s);
    println!("");
}

/*
======================================================================
Function: msg()
Show a message on the screen
----------------------------------------------------------------------
Parameters:
    message_01: Message to show on the screen : &str
Return:
    No
======================================================================
*/
pub fn msg(message_01 : &str){
    print!("{}\n", message_01);
}

/*
======================================================================
Function: ptime()
Show a message on the screen - Processing time
----------------------------------------------------------------------
Parameters:
    message_01: Start of processing : std::time::Instant
Return:
    No
======================================================================
*/
pub fn ptime(pro_time : std::time::Instant){

    let elapsed_time = pro_time.elapsed();
    let final_time = elapsed_time.as_secs();
    println!("The Processing Time is {} seconds.",final_time);
}

/*
======================================================================
Function: show_record()
presents a sample of the records of a matrix
----------------------------------------------------------------------
Parameters:
    regdatabase: vector with data : &Vec<[String;5]>
    n: number of registers to show : usize
Return:
    No
======================================================================
*/
```

```
pub fn show_record(regdatabase : &Vec<[String;NUM_COL]>, n : usize) {
    // Show the firsts register
    for ct in 0..n {
        print!("[ {} ] ", ct+1);
        for ct2 in 0..NUM_COL {
            if ct2 != (NUM_COL-1) {
                print!("{} | ",regdatabase[ct][ct2]);
            } else {
                print!("{}",regdatabase[ct][ct2]);
            }
        }
        println!("");
    }
}

/*
======================================================================
Function: show_record()
presents a sample of the records of a matrix
----------------------------------------------------------------------
Parameters:
    regdatabase: vector with data : &Vec<[String;5]>
    n: number of registers to show : usize
Return:
    No
======================================================================
*/
pub fn show_record2(regdatabase : &Vec<[String;TOT_COL]>, n : usize) {
    // Show the firsts register
    for ct in 0..n {
        print!("[ {} ] ", ct+1);
        for ct2 in 0..TOT_COL {
            if ct2 != (TOT_COL-1) {
                print!("{} | ",regdatabase[ct][ct2]);
            } else {
                print!("{}",regdatabase[ct][ct2]);
            }
        }
        println!("");
    }
}
```

[4] math.rs

```
/*
---------------------------------------------------------------------
MATHEMATICAL AND RELATED FUNCTIONS
---------------------------------------------------------------------
*/

use sha3::{Digest, Sha3_224};
extern crate num_bigint as bigint;
use num_bigint::BigUint;

/*
=====================================================================
Function: h224()
Extract hash with 224 bits of String
Return 56 characters hexadecimals
---------------------------------------------------------------------
Parameters:
    msg: Message for extract hash: String
Return:
    ret: hash : String
=====================================================================
*/
pub fn h224(msg : String) -> String {

    let mut hasher = Sha3_224::new();
    hasher.update(msg);
    let result = hasher.finalize();

    let mut ret = String::from("");

    for ct in result {
        let lbyte = format!("{:x}", ct);
        if ct < 16 {
            ret.push_str("0");
        }
        ret.push_str(&lbyte);
    }

    return ret;
}

/*
=====================================================================
Function: big_integer()
```

```
Lets you create a BigUint number
---------------------------------------------------------------------
Parameters:
    value: str for convert to BigUint : &str
    check: enables or disables error checking : bool
Return:
    number_1: BigUint
=====================================================================
*/
pub fn big_integer(value:  &str, _check : bool) -> BigUint {

    return value.parse::<BigUint>().unwrap();
    /*
    let number_1 = &value.to_string();

    // Security check - optional
    if check {
        let check_value = number_1.as_bytes();

        for ct in check_value {
            if ct < &48u8 || ct > &57u8 {
                println!("* * * * * There was a error in the conversion for BigUint number * * * * *");
                return BigUint::parse_bytes(b"0", 10).unwrap();
            }
        }
    }

    return number_1.parse::<BigUint>().unwrap();
    */
}

/*
=====================================================================
Function: pow_big_min()
Raises a bigUint to a small exponent
---------------------------------------------------------------------
Parameters:
    num: large integer without sign to raise to an exponent: &BigUint
    exponent: exponent to raise the base : &BigUint
Return:
    result: BigUint
=====================================================================
*/
pub fn pow_big_min(num : &BigUint , exponent : &BigUint) -> BigUint {

    let mut result = BigUint::parse_bytes(b"1", 10).unwrap();
    let control = BigUint::parse_bytes(b"1", 10).unwrap();
    let limit = BigUint::parse_bytes(b"0", 10).unwrap();
    let mut exp = exponent.clone();

    // if exp = 0, return 1
    if exp == limit {
        return control;
    }

    loop {
        result = result * num;
        exp = exp - &control;

        if exp == limit {
            break;
        }
    }

    return result;

}
```

[5] intermediate_file.rs

```
 /*
----------------------------------------------------------------------
INTERMEDIATE FILE - FUNCTIONS
----------------------------------------------------------------------
*/

extern crate rand;
use rand::thread_rng;
use rand::Rng;
extern crate num_bigint as bigint;
use num_bigint::BigUint;
extern crate bigdecimal;

// Use another source code file [math.rs]
use crate::math::*;
use crate::NUM_COL; // Number of collumns of table (cleartext)
use crate::TOT_COL; // Number of collumns of encrypted table - do not change this value!!!
use crate::NOISE;
use crate::LEN_BLOCK;
use crate::EMPTY_STRING;
use crate::cripto::*;
use crate::controle_chave;
use crate::compress::*;
use crate::COMPRESSAO;

/*
======================================================================
Function: transform_database()
creates the intermediate file (ready to be encrypted)
----------------------------------------------------------------------
Parameters:
    v: matrix with the data in plain text: &Vec<[String;NUM_COL]>
Return:
    x: numerically encoded file : Vec<[String;TOT_COL]
======================================================================
*/
pub fn transform_database(v : &Vec<[String;NUM_COL]>) -> Vec<[String;TOT_COL]> {

    // Prepare return this function:
    let mut x = Vec::new();

    // size of database
    let sizevet = v.len();
    // System key:
    let chave = controle_chave().0;
    let sbox = controle_chave().2;
    let isbox = controle_chave().3;

    // Compression vector
    let mapa = COMPRESSAO.as_bytes();

    // how to populate a String vector or declare the vector with default element
    let mut vet: [String; TOT_COL] = [EMPTY_STRING; TOT_COL];

    for ct in 0..sizevet {

        // convert each field of database
        for ct2 in 0..NUM_COL{
            // Differentiating the SBOX for each record field
            let campo : usize = isbox[(ct2.clone() % 256) as usize] as usize;
            vet[ct2] = convert_text(&v[ct][ct2].as_bytes().to_vec(),&LEN_BLOCK,&NOISE,&sbox,&campo) ;
        }

        // Add Hash Value (SHA3-224) - new field (see that vet contains TOT_COL elements)
        let campo : usize = isbox[NUM_COL % 256] as usize;
        vet[NUM_COL] = convert_text(&concatenate_texthash(&v[ct]).as_bytes().to_vec(),&LEN_BLOCK,&1,&sbox,&campo);

        // encrypting record
        let ret = cipher_database3(&vet,&chave);

        // Applying data compression
        for ct2 in 0..TOT_COL{
            let tmp = (ret[ct2]).as_bytes().to_vec();
            vet[ct2] = transform_vetor(tmp,mapa.to_vec());
        }

        // include the record in the return vector
        x.push(vet.clone());
    }

    return x;
}

/*
=====================================================================
Function: convert_text()
Function convert text into number-text
---------------------------------------------------------------------
Parameters:
    texto: text for convert number: &[u8]>
    sizeblock: size of the block that will be generated (in bytes) : i32
    noise: noise for encryption - value from 1 to 256 : u16;
    sbox: sbox system key: u8
    campo: variable to distinguish each field of the table
Return:
    ret: converted number string : String
=====================================================================
*/
fn convert_text(texto :  &[u8], sizeblock : &i32, noise : &u16, sbox: &[u8],campo:&usize) -> String {

    let mut ret = String::from("");
    let mut contador : i32 = sizeblock-2;  // 6
    let block_len = sizeblock -2;

    // size block:
    let big_exp = big_integer(&(sizeblock-1).to_string(), false);

    // base 256 informat bigUint
    let base = big_integer("257", false);

    // generate a random number (noise):
    let mut rng = thread_rng();
    let number_rand: u16 = rng.gen_range(0, noise).try_into().unwrap();

    // variable for beginning process
    let mut resultado : BigUint = number_rand * pow_big_min(&base , &big_exp);

    // Controlling the size of the information
    let tamanho_real : i32 = texto.len() as i32;
    let limite : i32 = tamanho_real % (sizeblock-1);
    let mut tamanho : i32 = tamanho_real - limite;

    // Very important this code for splitting short fields (sizeblock or less)
    if tamanho_real <= block_len {
        tamanho = tamanho_real;
        contador = tamanho -1;
    }

    let campo_cifra = campo.clone() as u8;

    // Main processing
    for ct in 0..tamanho{
        // Adding the information in the variable with the noise
        // passing the SBOX - differential for each field using SBOX
        let tmp = sbox[(texto[ct as usize]) as usize] ^ campo_cifra;
        resultado += ((tmp+1) as u16) * pow_big_min(&base , &big_integer(&contador.to_string(),false));
        contador = contador - 1;
        // Checking the end of each block
        if contador < 0 {
            // Reset blocksize
            contador = block_len;

            // Adding the value to the cumulative string
            let linha = &resultado.to_string();
            ret.push_str(linha);
            if ct != (tamanho-1){
                ret.push_str(".");
            }

            // Resetting the noise vector
            let number_rand: u16 = rng.gen_range(0, noise).try_into().unwrap();
            resultado = number_rand * pow_big_min(&base , &big_exp);

        }
    }

    // Code to check if there is an incomplete piece of block left to be processed:
    if tamanho != tamanho_real {
        contador = tamanho_real - tamanho -1;

        let number_rand: u16 = rng.gen_range(0, noise).try_into().unwrap();
        resultado = number_rand * pow_big_min(&base , &big_exp);

        for ct in tamanho..tamanho_real {
            // passing the SBOX - differential for each field using SBOX
            let tmp = sbox[(texto[ct as usize]) as usize] ^ campo_cifra;
            resultado += ((tmp+1) as u16) * pow_big_min(&base , &big_integer(&contador.to_string(),false));

            contador = contador - 1;
        }

            // create the final string
            let linha = &resultado.to_string();
            ret.push_str(".");
            ret.push_str(linha);

    }

    return ret;
}

/*
=====================================================================
Function: concatenate_texthash()
Return hash SHA3-224 for concatenated fields
----------------------------------------------------------------------
Parameters:
    vet: vect with fields of table: [String;NUM_COL]>
Return:
    ret: hash of concatened fields : String (Hexadecimal)
======================================================================
*/
fn concatenate_texthash(vet : &[String;NUM_COL]) -> String {
    let mut hash_text = String::from("");

    for ct2 in 0..NUM_COL{
        hash_text.push_str(&vet[ct2]);
    }

    return h224(hash_text);

}

/*
======================================================================
Function: convert_text_dec()
Function convert text into number-text
----------------------------------------------------------------------
Parameters:
    texto: text for convert number: &String
    sizeblock: size of the block that will be generated (in bytes) : i32
    others: auxiliary variables to speed up processing
Return:
    ret: converted number string : String
======================================================================
*/
fn convert_text_dec(texto :  &String, sizeblock : i32, zero : &BigUint, bigu : &[BigUint], bigu_pot : &[BigUint], guarda: &BigUint,
isbox: &[u8], campo:&usize) -> String {

    let mut ret = String::from("");
    let mat = texto.split(".");
    let mut valor : BigUint = big_integer("0",false);

    // main loop of this function
    for texto_parte in mat {
        let mut num : BigUint = texto_parte.parse::<BigUint>().unwrap();

        // Alternative code for removing noise
        let num2 = num.clone();
        num = num - (guarda * (num2 / guarda));
        let campo_cifra = campo.clone();

        // converting to string
        let mut limite_bloco2 = sizeblock.clone();

        for _ct3 in (0..sizeblock).rev(){
            let mut contador : usize = 0;

            for controle in (0..limite_bloco2).rev(){
                valor = &bigu_pot[controle as usize] * &bigu[256];
                if valor <= num{
                    contador = controle as usize + 1;
                    break;
                }
            }

            // reducing the threshold value to increase the speed
            limite_bloco2 = (contador as i32).clone();

            // if the values are equal we end the loop
            if num == valor {
                let vet : Vec<u8> = [isbox[(255^campo_cifra)as usize]].to_vec();
                let s = String::from_utf8_lossy(&vet);
                ret.push_str(&s);
                break;
            }

            // alternative decryption code
            let prop = &num / &bigu_pot[contador];
            let m = prop.to_string().parse::<u16>().unwrap();
            num = num - (&bigu_pot[contador] * &bigu[m as usize]);
            let tmp = ((m-1) as u8) ^ campo_cifra as u8;
            let vet : Vec<u8> = [isbox[tmp as usize]].to_vec();

            let s = String::from_utf8_lossy(&vet);
            ret.push_str(&s);

            if &num == zero{
                break;
            }

        }

    }

    return ret;
```

```
}

/*
=====================================================================
Function: verify_hash()
data checking function
---------------------------------------------------------------------
Parameters:
    vet: vect with fields of table: [String;TOT_COL]>
Return:
    ret: verify hash : bool
=====================================================================
*/
pub fn verify_hash(vet : &[String;TOT_COL]) -> bool {

    let mut hash_text = String::from("");
    for ct2 in 0..NUM_COL{
        hash_text.push_str(&vet[ct2]);
    }

    let hash1 = h224(hash_text);
    let hash2 = vet[NUM_COL].clone();

    if hash1 == hash2 {
        return true;
    } else {
        return false;
    }

}

/*
=====================================================================
Function: verify_database()
Does the final check of the data decryption
---------------------------------------------------------------------
Parameters:
    v: matrix with the data in plain text in decryption: &Vec<[String;NUM_TOT]>
Return:
    x: matrix plain text : Vec<[String;NUM_COL]
=====================================================================
*/
pub fn verify_database(v : &Vec<[String;TOT_COL]>) -> Vec<[String;NUM_COL]> {

    // Prepare return this function:
    let mut x = Vec::new();
    // size of database
    let sizevet = v.len();

    // how to populate a String vector or declare the vector with default element
    let mut vet: [String; NUM_COL] = [EMPTY_STRING; NUM_COL];

    let mut contador = 0;

    for ct in 0..sizevet {
        // data verification
        let ret = verify_hash(&v[ct]);

        if ret == false {
            println!("There was an error in decrypting the data!!!!");
            contador = contador + 1;
        }

        // Copy data for return
        for ct2 in 0..NUM_COL{
            vet[ct2] = v[ct][ct2].clone();
        }

        // include the record in the return vector
        x.push(vet.clone());
    }

    // checking if there were any errors in the decryption
    println!("\nNumber of errors: {}",contador);
    return x;
}

/*
=====================================================================
Function: transform_database_dec()
inverse function transform_database()
---------------------------------------------------------------------
Parameters:
    v: matrix with the data in plain text: &Vec<[String;TOT_COL]>
    others: auxiliary variables to speed up processing
Return:
    x: numerically encoded file : Vec<[String;NUM_COL]
=====================================================================
*/
pub                                 fn                                  transform_database_dec2(v                          :
[String;TOT_COL],zero:&BigUint,bigu:&Vec<BigUint>,bigu_pot:&Vec<BigUint>,guarda:&BigUint,isbox:&[u8])  -> [String;TOT_COL] {

    // how to populate a String vector or declare the vector with default element
    let mut vet: [String; TOT_COL] = [EMPTY_STRING; TOT_COL];

    // convert each field of database
    for ct in 0..TOT_COL{
        let campo : usize = isbox[ct.clone() % 256] as usize;
        vet[ct] = convert_text_dec(&v[ct],LEN_BLOCK,&zero,&bigu,&bigu_pot,&guarda,&isbox,&campo);
    }

    return vet;
}
```

[6] cripto.rs

```
/*
---------------------------------------------------------------------
FILE ENCRYPTION FUNCTION
---------------------------------------------------------------------
*/

extern crate num_bigint as bigint;
use num_bigint::BigUint;
use crate::TOT_COL;
// Use another source code file [math.rs]
use crate::math::big_integer;
use crate::EMPTY_STRING;
use crate::compress::*;
use crate::intermediate_file::*;
use crate::LEN_BLOCK;
use crate::math::*;
use crate::COMPRESSAO;
use crate::controle_chave;

extern crate rand;
use rand::thread_rng;
use rand::Rng;

/*
=====================================================================
function cipher_database()
Function for cipher the intermediate database
The key is passed as parameter
---------------------------------------------------------------------
Parameters:
    database: database intermediate format : Vec<[String;TOT_COL]>
    chave: vector with the keys and modules
    modulo: module for operations
Return:
    x : return database encrypted: Vec<[String;TOT_COL]>
=====================================================================
*/
pub fn cipher_database3(database : &[String;TOT_COL], chave : &Vec<BigUint>) -> [String;TOT_COL] {

    // how to populate a String vector or declare the vector with default element
    let mut cifra: [String; TOT_COL] = [EMPTY_STRING; TOT_COL];

        // random data insertion point
        let mut rng = thread_rng();
        let number_rand: u8 = rng.gen_range(0, 8).try_into().unwrap();

        for campo in 0..TOT_COL {
            // create the string for reading data
            let mut resultado = "".to_string();
            // Table field to be encrypted
            let registro = database[campo].split(".");
            // temporary vector
            let mut y: Vec<BigUint> = Vec::new();

            // transform String Vector in type BigUint
            for ct3 in registro{
                y.push(ct3.parse::<BigUint>().unwrap());
            }

            // Encryption
            let tam2 = y.len();
            for ct4 in 0..tam2{
                // Non-deterministic encryption
                y[ct4] = cifrar(&y[ct4], chave, &number_rand);

                resultado.push_str(&y[ct4].to_string());
                if ct4 != (tam2-1) {
                    resultado.push_str(".");
                }
            }

            // storing field information in the accumulator
            cifra[campo] = resultado;
        }

    return cifra;   // Function return
}


/*
=====================================================================
Function to encrypt the data
Receives the BigUint value and returns the encrypted BigUint value with the NJS pattern
=====================================================================
*/
fn cifrar(dados : &BigUint, chave : &Vec<BigUint>, controle: &u8) -> BigUint{

    // Choose a random operation type to encrypt
    let mut ret : BigUint = dados.clone();

    match controle {
```

```
0 =>
// ADD MUL XOR
{
    let mut fator = 0;
    loop {
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 44 {
            break;
        }
    }
},

1 =>
// MUL ADD XOR
{
    let mut fator = 0;
    loop {
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 44 {
            break;
        }
    }
},

2 =>
// ADD MUL XOR
{
    let mut fator = 0;
    loop {
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 32 {
            break;
        }
    }
},
3 =>
// MUL ADD XOR
{
    let mut fator = 0;
    loop {
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 32 {
            break;
        }
    }
},

4 =>
// ADD MUL XOR
{
    let mut fator = 0;
    loop {
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 20 {
            break;
        }
    }
},

5 =>
// MUL ADD XOR
{
    let mut fator = 0;
    loop {
        ret = (ret * (&chave[1+fator])) % &chave[3+fator];
        ret = (ret + &chave[0+fator]) % &chave[3+fator];
        ret = ret ^ (&chave[2+fator]);

        fator = fator + 4;
        if fator > 20 {
            break;
        }
    }
```

```
        },

        6 =>
        // ADD MUL XOR
        {
            let mut fator = 0;
            loop {
                ret = (ret + &chave[0+fator]) % &chave[3+fator];
                ret = (ret * (&chave[1+fator])) % &chave[3+fator];
                ret = ret ^ (&chave[2+fator]);

                fator = fator + 4;
                if fator > 8 {
                    break;
                }
            }
        },

        7..=255 =>
        // MUL ADD XOR
        {
            let mut fator = 0;
            loop {
                ret = (ret * (&chave[1+fator])) % &chave[3+fator];
                ret = (ret + &chave[0+fator]) % &chave[3+fator];
                ret = ret ^ (&chave[2+fator]);

                fator = fator + 4;
                if fator > 8 {
                    break;
                }
            }
        }

    };

    return ret;

}

/*
=====================================================================
Function to decrypt the data
Takes the BigUint value and returns the decrypted BigUint value with the NJS pattern
=====================================================================
*/
fn decifrar(dados : &BigUint, ichave  : &Vec<BigUint>, tipo: &u8) -> BigUint{
```

```
let controle : u8 = tipo.clone();
let mut ret : BigUint = dados.clone();

match controle {
    0 =>
    // XOR MUL ADD
    {
        let mut fator : i32 = 44;
        loop {
        ret = ret ^ &ichave[(2+fator) as usize];
        ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
        ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
        fator = fator - 4;
            if fator < 0 {
                break;
            }
        }
    },

    1 =>
    // XOR ADD MUL
    {
        let mut fator : i32 = 44;
        loop {
        ret = ret ^ &ichave[(2+fator) as usize];
        ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
        ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
        fator = fator - 4;
            if fator < 0 {
                break;
            }
        }
    },

2 =>
// XOR MUL ADD
{
    let mut fator : i32 = 32;
    loop {
    ret = ret ^ &ichave[(2+fator) as usize];
    ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
    ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
    fator = fator - 4;
        if fator < 0 {
            break;
        }
    }
},

3 =>
// XOR ADD MUL
{
    let mut fator : i32 = 32;
    loop {
    ret = ret ^ &ichave[(2+fator) as usize];
    ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
    ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
    fator = fator - 4;
        if fator < 0 {
            break;
        }
    }
},

4 =>
// XOR MUL ADD
{
    let mut fator : i32 = 20;
    loop {
    ret = ret ^ &ichave[(2+fator) as usize];
    ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
    ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
    fator = fator - 4;
        if fator < 0 {
            break;
        }
    }
},

5 =>
// XOR ADD MUL
{
    let mut fator : i32 = 20;
    loop {
    ret = ret ^ &ichave[(2+fator) as usize];
    ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
    ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
    fator = fator - 4;
        if fator < 0 {
            break;
        }
    }
}
```

```
    6 =>
    // XOR MUL ADD
    {
        let mut fator : i32 = 8;
        loop {
        ret = ret ^ &ichave[(2+fator) as usize];
        ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
        ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
        fator = fator - 4;
            if fator < 0 {
                break;
            }
        }
    },

    7..=255 =>
    // XOR ADD MUL
    {
        let mut fator : i32 = 8;
        loop {
        ret = ret ^ &ichave[(2+fator) as usize];
        ret = (ret + &ichave[(0+fator)  as usize]) % &ichave[(3+fator)  as usize];
        ret = (ret * (&ichave[(1+fator)  as usize])) % &ichave[(3+fator)  as usize];
        fator = fator - 4;
            if fator < 0 {
                break;
            }
        }
    }

};

    return ret;

}

/*
=====================================================================
function cipher_database()
Function for cipher the intermediate database
The key is passed as parameter
---------------------------------------------------------------------
Parameters:
    database2: database intermediate format : Vec<[String;TOT_COL]>
    ichave: vector with the keys and modules
Return:
    x : return database encrypted: Vec<[String;TOT_COL]>
======================================================================
*/
pub fn inv_cipher_database3(database2 : &Vec<[String;TOT_COL]>, ichave : &Vec<BigUint>) -> Vec<[String;TOT_COL]> {

    // Sbox for decryption
    let isbox = controle_chave().3;

    // attempt to speed up the code
    let mut bigu : Vec<BigUint> = vec![big_integer("0",false);512];
    for ct in 0..512{
        bigu[ct] = big_integer(&(ct+0).to_string(), false);
    }

    // size block - exponent:
    let big_exp = big_integer(&(&LEN_BLOCK-1).to_string(), false);
    // base 256 in format bigUint
    let base = big_integer("257", false);
    let zero = big_integer("0", false);
    let guarda = pow_big_min(&base,&big_exp);

    // attempt to speed up the code
    let mut bigu_pot : Vec<BigUint> = vec![big_integer("0",false);512];
    for ct in 0..512{
        bigu_pot[ct] = pow_big_min(&base,&big_integer(&(ct+0).to_string(), false));
    }

    let mut ret = Vec::new();
    // how to populate a String vector or declare the vector with default element
    let mut cifra: [String; TOT_COL] = [EMPTY_STRING; TOT_COL];

    // Map to decompress
    let mapa = COMPRESSAO.as_bytes();

    // Here we start the actual decryption
    for database in database2 {

        let mut database3 = database.clone();

        // Decompressing the data before decrypting
        for ct2 in 0..TOT_COL{
            let tmp = (database[ct2]).as_bytes().to_vec();
            database3[ct2] = transform_vetor_dec(tmp,mapa.to_vec());
        }

        for tipo in 0..8 {  // decryption variations
            for campo in 0..TOT_COL {
                // create the string for reading data
                let mut resultado = "".to_string();
                // Table field to be encrypted
                let registro = database3[campo].split(".");
                // temporary vector
                let mut y: Vec<BigUint> = Vec::new();

                // transform String Vector in type BigUint
                for ct3 in registro{
                    y.push(ct3.parse::<BigUint>().unwrap());
                }

                // decryption
                let tam2 = y.len();
                for ct4 in 0..tam2{
                    y[ct4] = decifrar(&y[ct4], ichave, &tipo);

                    resultado.push_str(&y[ct4].to_string());
                    if ct4 != (tam2-1) {
                        resultado.push_str(".");
                    }
                }

                // storing field information in the accumulator
                cifra[campo] = resultado;
            }

            // Checking the decryption control:
            let tmp = &transform_database_dec2(cifra.clone(),&zero,&bigu,&bigu_pot,&guarda,&isbox);

            if verify_hash(tmp) == true {
                break;
            }
        }

        // Determining the correct decryption
        ret.push(transform_database_dec2(cifra.clone(),&zero,&bigu,&bigu_pot,&guarda,&isbox));
    }

    return ret;
}
```

[7] read_write.rs

```
/*
----------------------------------------------------------------------
READ AND WRITE FILES
----------------------------------------------------------------------
*/

use std::fs::File;
use std::io::{BufReader, BufRead};
use std::io::Write;

use crate::FILE_PATH;
use crate::NUM_COL;
use crate::TOT_COL;
use crate::EMPTY_STRING;

/*
======================================================================
Function: read_database()
Read a text file and put the memory
----------------------------------------------------------------------
Parameters:
    filename: filename : &str
Return:
    array with fields read from text files : Vec<[String;5]>
======================================================================
*/
pub fn read_database(lfilename : &str) -> Vec<[String;NUM_COL]> {

    // Preparing to read the file path
    let fpath = FILE_PATH.to_string();
    let mut filename = String::from("");
    filename.push_str(&fpath);
    filename.push_str("/");
    filename.push_str(&lfilename);

    // Create the vector for processing
    let mut read_vector = Vec::new();

    // Create a return vector:
    let mut ret_database = Vec::new();

    // Open the file in read-only mode (ignoring errors).
    let file = File::open(filename).expect("Error reading file. Set the FILE_PATH variable");
    let reader = BufReader::new(file);
    // Read lines from file
    for line in reader.lines() {
        let line = line.unwrap(); // Ignore errors.
        read_vector.push(line.to_string());
    }

    // how to populate a String vector or declare the vector with default element
    let mut y: [String; NUM_COL] = [EMPTY_STRING; NUM_COL];

    // Main processing
    for line in read_vector {
        // transform a string in a vector
        let word_01 = line.split("|");

        let mut campo = 0;  // variable to control the divisions in each field
        for ct in word_01 {
            y[campo] = ct.trim().to_string();
            campo = campo + 1;
        }

        // put new element in matrix
        ret_database.push(y.clone());

    }
    // Return matrix with data
    return ret_database;
}

/*
======================================================================
Function: write_database()
write a text file in disk
----------------------------------------------------------------------
Parameters:
    filename: filename : &str
    database : table with fields : Vec<[String;NUM_COL]>
Return:
    std::io::Result<()>
======================================================================
*/
pub fn write_database(lfilename: &str, database : & Vec<[String;NUM_COL]>) -> std::io::Result<()> {

    // Preparing to write the file path
    let fpath = FILE_PATH.to_string();
    let mut filename = String::from("");
    filename.push_str(&fpath);
    filename.push_str("/");
    filename.push_str(&lfilename);
    let mut file = File::create(filename)?;

    let tam = database.len();

    for ct in 0..tam {
        let mut registro = String::from("");
        for ct2 in 0..NUM_COL{
            registro.push_str(&database[ct][ct2]);
            if ct2 != (NUM_COL-1) {
                registro.push_str(" | ");
            }
        }

        registro.push_str("\n");
        file.write_all(registro.as_bytes())?;
    }

    Ok(())
}

/*
=====================================================================
Function: write_database_encrypted()
write a text file in disk
---------------------------------------------------------------------
Parameters:
    filename: filename : &str
    database : table with fields : Vec<[String;NUM_COL]>
Return:
    std::io::Result<()>
=====================================================================
*/
pub fn write_database_encrypted(lfilename: &str, database : & Vec<[String;TOT_COL]>) -> std::io::Result<()> {

    // Preparing to write the file path
    let fpath = FILE_PATH.to_string();
    let mut filename = String::from("");
    filename.push_str(&fpath);
    filename.push_str("/");
    filename.push_str(&lfilename);
    let mut file = File::create(filename)?;

    let tam = database.len();

    for ct in 0..tam {
        let mut registro = String::from("");
        for ct2 in 0..TOT_COL{
            registro.push_str(&database[ct][ct2]);
            if ct2 != (TOT_COL-1) {
                registro.push_str("|");
            }
        }

        registro.push_str("\n");
        file.write_all(registro.as_bytes())?;
    }

    Ok(())
}
```

[8] cargo.toml

```
[package]
name = "projeto"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
rand = "0.7.3"
base64 = "0.13.0"
num-bigint = "0.2"
sha3 = "0.10.1"
bigdecimal = "0.1"
num = "0.1"
```

[9] database_min.txt

```
Helda Fabrizio Corto | 576837751-29 | (27)94640-5409 | 23/03/2013 | 2886.22
Charlles Dockhorn Cossio Campara | 712237047-62 | (35)98394-9535 | 26/03/2022 | 26772.52
Guimar Acauan Azmus Brinke Caurrinhos Gabani | 222770563-79 | (17)92477-7699 | 25/06/2020 | 242.59
Iliana Fiorese Azzalini | 438695869-02 | (48)98827-9027 | 13/06/2010 | 2115.71
Diulia Albeck Grassetto Tessaro Frota Goodwin | 067880700-09 | (68)96658-8199 | 17/12/2012 | 3873.97
Nei Gottschalk | 053327513-38 | (65)92373-4258 | 07/04/2007 | 540.83
Denilda Ancinelo Efrem | 957259222-55 | (97)92088-8319 | 14/11/2015 | 19850.56
Weldson Biancardi Andersson | 252363691-66 | (93)90986-1043 | 04/06/2018 | 1982.85
Andreisa Bettger Beghi | 933807975-77 | (42)98156-2205 | 13/09/2014 | 15600.99
Manuela Brostolon Annunziato | 044044698-51 | (68)97933-6839 | 08/09/2015 | 809.54
Waderson Corpas | 559673060-30 | (69)94085-9162 | 29/05/2016 | 2102.14
Genisson Fisher Benassatto | 385677859-21 | (75)98790-8784 | 29/02/2020 | 833.54
Kenya de Albernas | 124195273-65 | (79)90834-0095 | 14/02/2009 | 8454.10
Daira Grossu Alexandre Chioca Giuduce | 810065369-15 | (69)92933-3889 | 26/05/2018 | 6452.43
Joventino Goffad Eichendorf | 673812302-46 | (71)98164-5349 | 28/07/2012 | 5515.55
Eliosmar Bartici | 010206101-57 | (14)92937-7096 | 23/05/2014 | 14363.99
Cleyde Bonsegno | 658708865-32 | (17)90338-9135 | 12/11/2009 | 19779.87
Mariliane Agnetti Hooper Mamiro | 903954380-09 | (61)97100-6801 | 30/01/2009 | 9827.69
Dalessandro Fischborn | 084775090-57 | (2)90918-5500 | 09/03/2005 | 11463.12
Milana Gallagher Graeber Brunel | 307368986-02 | (18)94207-4931 | 03/07/2013 | 7474.54
Thaiza Bizzo | 433528700-09 | (6)93230-0841 | 14/02/2007 | 5281.97
Edso Adornes | 522440324-82 | (18)98719-2635 | 02/11/2012 | 11772.34
Dinara Bordina | 666403069-08 | (19)98194-5912 | 14/09/2010 | 9015.75
Veleda Deleon | 489682721-70 | (4)97794-6577 | 25/02/2017 | 11497.68
Alessandra Fruhan | 026879622-84 | (64)91386-5413 | 07/03/2021 | 29204.55
Kevilin Cicarini | 675339496-82 | (53)94512-1129 | 10/05/2010 | 4400.94
Teuma Gagetti | 577271378-60 | (24)94922-2718 | 16/11/2017 | 2591.46
Tertuliano Nogueira | 542293619-52 | (47)96606-6691 | 12/05/2015 | 1128.66
Edleni Borella | 573910000-00 | (6)96148-7154 | 06/04/2006 | 3172.58
Marcondes Goncalves Invernizzi | 814251795-11 | (93)96488-6412 | 03/09/2019 | 9892.98
Oracio Dellonardo Benfatti Burkhard Rassato Groehs | 619610284-78 | (38)96902-3369 | 19/01/2017 | 7130.25
Arival Romero Guglielmi | 277339799-00 | (51)91348-1062 | 15/04/2022 | 1922.99
Kele Amando Calimam Ferdinandi Lamha | 106388769-40 | (53)92471-4046 | 08/09/2004 | 9829.03
Dineide Canes Caldana | 530842160-46 | (13)98514-9811 | 09/10/2010 | 160.82
Milena Rush | 276606579-01 | (28)91459-2818 | 24/03/2003 | 13064.85
Vicentina Filippini | 249851544-80 | (9)94903-2774 | 28/02/2021 | 1587.54
Guibson Haarstick Medrado Galucci | 325921292-03 | (87)90098-5893 | 29/06/2015 | 14882.66
Mercedes Eickhof Torezani | 796662050-51 | (86)99325-1241 | 13/01/2018 | 7370.43
Olimpo Brackmann Bonese | 576690026-00 | (19)93264-6139 | 06/07/2003 | 4478.52
Joseph Hansen Hooper | 332949425-92 | (98)93336-4199 | 13/03/2019 | 26269.86
Dielen Basseto Baumy | 137307536-10 | (38)90641-6453 | 06/02/2019 | 8465.11
Ubelina Henrichsen | 094461172-01 | (5)98934-1424 | 14/04/2003 | 29672.59
Raylson Cotrim | 507793620-53 | (45)99607-1257 | 15/07/2016 | 3082.02
Valquer Durival | 754068785-00 | (14)92171-5559 | 06/04/2015 | 10903.54
```

Wellisson Faccipieri Brigati	661419240-18	(3)95307-1899	13/03/2017	1253.10
Junia Covalesci Bacello	821422627-06	(6)99994-8329	13/10/2012	3989.39
Anthony Ferezini	193213661-49	(3)99370-3567	28/12/2004	4035.06
Maxel Dothes Davrison Collevati Concari	197281777-80	(7)92845-8966	22/07/2011	8420.54
Evelaine Diuliana Appol Sperotto Altreiter Cunshnir	441433682-93	(98)90803-1733	27/11/2013	27632.18
Gilda Dondi Altenetter	926027157-27	(93)95397-8657	07/05/2010	15480.82
Claudelina Bindchen	393694677-38	(79)95852-0543	26/02/2006	9936.42
Hermelindo Carnas Burkle	389464377-00	(14)90595-5222	13/06/2017	5896.58
Jerfesom Boettner Carturan	639160956-10	(42)93681-0888	22/06/2021	3514.91
Priciane Finkler	738359137-80	(54)91060-6514	23/07/2014	1888.22
Augusto Gasparotto Guzzi	644584713-01	(32)94002-1741	17/02/2012	6157.24
Maria Eduarda Mayer	253775982-40	(75)99151-8798	24/03/2008	4070.19
Gilcemar Basenova Pais Adamo Cappellaro Price	220589100-08	(82)94961-1207	09/06/2013	6167.68
Domiciano Avona Campisano	267052015-62	(65)92130-5471	15/10/2013	4900.23
Zenobio Favari Gonzalez	150921090-57	(12)90768-0145	19/10/2007	2173.97
Angelice Zerbone Goncales Guitarra Pozinni	820656852-01	(81)91228-6841	10/06/2017	25591.18
Semirames Buriani Bazilevitz	515970813-97	(38)99523-3468	21/11/2018	7002.86
Francelino Manfio	306758165-94	(6)92511-9672	12/08/2003	752.88
Elizelda Negri	530334571-99	(51)96020-7853	02/02/2003	4729.22
Leoberto Ferreira Collina	002488228-09	(24)93590-8240	01/11/2020	14179.04
Analice Lunaidi Dompieri Adamska Barbabela de Mattos Eichholz	270491236-29	(79)96628-2192	14/10/2014	6056.81
Diniz Gularte	851958482-21	(15)90821-4967	25/12/2010	28197.80
Glauce Cuadro Gay	069647173-19	(49)90547-1777	26/08/2020	9827.14
Josenita Praba	498921301-35	(4)94805-2797	01/09/2011	7103.62
Maronita Frias	226708101-32	(53)91155-2100	08/07/2014	26293.21
Valmiro Mccormick Borri Cagnin Saccomano	770116621-93	(16)91286-6260	18/05/2021	7361.57
Divanei Jungllut Geanine	148619691-98	(65)97231-5457	13/08/2015	3556.45
Helenir Dechsler Good Deutschle	262414126-07	(67)98576-6817	30/10/2017	2954.49
Gorge Lupino	838556605-52	(94)91727-5409	08/06/2008	2095.84
Jaelson Haynes Kluge	868852555-02	(42)96432-3574	02/02/2020	105.45
Raimuda Bailey Cauzzi	350563621-41	(42)92151-9150	04/12/2008	25666.28
Oliverio Flocke	069221046-60	(12)99694-3983	20/09/2007	4382.77
Melke Gherele Braschi Fiorenza Entringer Dixen	097443834-68	(62)95688-7918	25/03/2011	9457.95
Lisangela Bernar	292645790-55	(42)94942-2105	27/05/2019	9016.15
Naile Aleci Barabani	002037831-72	(87)90956-2350	12/07/2014	1884.80
Cleidir Valani	815054587-47	(98)98285-7133	14/06/2003	18234.37
Claudimara Carpegiani	412974882-39	(31)92616-0470	01/07/2014	3513.99
Lucca Lorenzetto	125347476-90	(28)92250-0784	09/04/2010	8851.75
Kesia Cesarini	218791950-05	(45)92171-3962	12/05/2015	6603.49
Albino Librenti Battilana Hafner Gloeden Girolami	144451013-12	(73)91332-5386	27/05/2017	19945.15
Iracena Gilande	079017033-94	(82)99576-8375	19/09/2011	17562.33
Marcon Camos Cohen	479097814-49	(37)92889-1684	15/08/2006	1758.15
Maria Sophia Ewing Dickhut	071088715-37	(12)94968-0890	13/12/2010	1566.77
Romenique Strong	330054630-59	(8)91860-4120	25/08/2016	28308.64
Zefira Aloisio	668150351-50	(68)98958-9075	10/02/2020	5651.68
Christina Fresh	247400546-03	(84)91423-2110	11/08/2003	6949.80
Walice Astolfo Braccer	584819180-93	(12)98373-1613	02/04/2006	26481.65

```
Sumiko Boarato | 397672799-41 | (42)98357-0431 | 03/05/2008 | 521.79
Kayane Frimm | 345470664-00 | (96)98225-2581 | 04/07/2019 | 4840.16
Genaina Camarin Holanda Devita Pollini Aleci | 832843930-00 | (5)98823-9438 | 20/12/2005 | 1358.03
Welem Dopke Collares Fretta | 442767690-22 | (8)99339-4407 | 27/06/2006 | 3925.28
Devison Christiansdatter Dallemole Anele Frison | 083712909-70 | (83)95211-3284 | 11/11/2020 | 22853.40
Tanara Fabriz | 133997017-35 | (41)96469-8159 | 27/09/2014 | 11436.58
Clailson Cossari Fanto Bombel Brugalli | 452399361-52 | (83)90077-8258 | 05/03/2008 | 383.57
Ozeni Arrieira | 429644396-21 | (79)92299-7634 | 05/02/2009 | 595.79
Glaziele Bortoloti Bocaccio Bradburger Bartmann Gasparotto Celestino | 655612078-70 | (17)90302-7251 | 18/03/2008 | 13171.65
```