# A STUDY ON THE ENCRYPTION TECHNIQUES AND METHODS IMPLEMENTED WITHIN THE CRITICAL INFRASTRUCTURES

Eugen NEACȘU[1], Emil SIMION[2]

*This paper is a study of the niche approach, the use of cryptographic algorithms in critical infrastructures. This newly introduced notion means the existence in a cryptosystem of the three elements necessary and sufficient for its operation: the pseudo-random cryptographic key generator, the key storage and system management, and the encryption and decryption application. The mentioned elements have an interdependent operation in the achievement of the cryptographic process. The work begins with an introduction to the operation and implementation of the XOR-MWC generator, a new generator, obtained from the implementation of two pseudo-random MWC generators (Multiply-With-Carry) and an XOR function (logic function or exclusively) applied to the resulting bits at the output of the two generators. In the application code there are given several pseudo-random generators by means of which we are able to obtain from the database the partial keys to form the encryption key and with the help of which, the ability to established one of its various methods of operation is provided.*

**Keywords**: cryptography, encryption, RPV algorithm, key generation.

## 1. Introduction

In today's context, where interconnected media have become available to the general public, information needs the most effective means of protection. The methods of information protection, specific to contemporary information technology, are varied, depending on the types of attacks to which the information may be exposed. Awareness of the risks in communications networks has led to widespread use of hardware and software solutions such as antivirus, antispyware, antispam, firewall, VPN (Virtual Private Network), intrusion prevention and detection programs (IPS, IDS) or encryption methods. The process of securing information has become an essential component in the contemporary information society, which has led to the achievement of specific international standards, the most important of which are ISO 27001 and BS 7799. [1]

Currently, the most effective methods of conserving the integrity of information are given by cryptographic techniques. This paper makes a foray into

[1] Security Engineer, Advanced Technologies Institute, Bucharest, Romania, e-mail: neacsu.eugen@yahoo.com

[2] Associate Professor, University POLITEHNICA of Bucharest, Romania, e-mail: emil.simion@upb.ro

these information security techniques and proposes, both to the cryptographic community and to the specialists in the administration and security of information systems, a new solution for ensuring the security of confidential data in critical infrastructures.

A critical infrastructure is a service whose functionality is so vital that its destruction would have a debilitating impact on the security of any country. Basically, the system is subject to the following four attributes: availability, integrity, reliability, and security. Disruption of any system could jeopardize the continued operation of the entire infrastructure. In this context, section 2 presents the cryptographic key generation component, providing the representation of the partial key generation and the encryption key. Section 3 is mainly for the encryption and decryption component and section 4 presents the conclusions of the security study conducted. Classical cryptography encompasses most of the algorithms implemented so far in communication systems. They are generally based on simple operations: addition, displacement, multiplication, substitution, subtraction, and division of modules. The security of systems is that an attack on them involves a very large number of mathematical operations, which makes it impossible to find out. With the advent of parallel computing techniques, these systems can be vulnerable to various types of attacks, so they are trying to strengthen them by increasing the size of the keys or by increasing the complexity of the encryption process. Thus, two types of classical cryptographic algorithms were developed and imposed: symmetric algorithms (with secret key) and asymmetric algorithms (with public keys). The efficiency of these solutions has led to their recent implementation even in environments with low computing resources, such as ad-hoc sensor networks. [2]

Recent research in cryptography is attempting a new approach to the encryption process by introducing new operations in addition to the classical ones, as well as independence from increasing computing power. It is assumed that with the advent of the quantum computer, computing resources will be virtually unlimited, which will lead to the situation where data security systems can be overcome much more easily by brute force attack, compromising data confidentiality. This has led to the development of new research directions like cryptography based on chaotic dynamic systems and quantum cryptography. [3] The application of chaos in cryptography is suggested by its properties: sensitivity to initial conditions and loss of information about the initialization point. The main advantage of this method is the high degree of security offered. The disadvantages are due to the low working speed and the complexity of the floating-point calculations, which would make this technique difficult to implement for real-time use.

## 2. Cryptographic key generation component

The implementation is made using a computer network in which encryption devices and storage media (stations and servers) are considered secure (by using firewall policies as well as antivirus and antispyware programs). If the stations on which the data encryption is performed were considered unsafe, the whole process would be useless. Two stations are sufficient for testing in a local network on which the encryption application is installed together with the database.

For transmission of the database containing the partial cryptographic keys to users of the critical infrastructure, it is recommended to use a secure data network, or tunneling methods. In the absence of a network, for loading databases to the stations on which this system is installed, mobile storage devices can also be used.

A cryptographic mode combines basic cipher and a few simple operations. The operations are simple, because the security depends on the cipher used and not on the cryptographic mode. There are other security considerations: clear text patterns must be hidden, entry into the encryption algorithm must be random, and encryption of more than one message using the same key must be possible.[4]

Another aspect to consider is efficiency - a cryptographic mode must not be less efficient than the encryption algorithm on which it is based. In some circumstances it is important that the clear text and the ciphertext are the same length.

Another consideration is error tolerance. Some applications require encryption or decryption as parallel processes, while others need to be able to pre-process as much data as possible. It is important that the decryption process can recover bit-level errors from encrypted text, missing bits, or extra bits.

The multiply-with-carry (MWC) generator was proposed by George Marsaglia in 1994 and analyzed by Couture and L'Ecuyer in 1997. MWC has been proposed as a modification of the add-with-carry (AWC) generator.[5]

In the first stage, after declaring the variables, the algorithm calls the classes in its composition (MWC1 and MWC2) to obtain from the system, using the *SetSeedFromSystemTime* method, the seed values used at initialization.

```
namespace RNG
{
    /// <summary>
    /// </summary>
    public class MWC1
    {
        private static uint m_w;
        private static uint m_z;
```

```
            public static void SetSeedFromSystemTime()
            {
                System.DateTime dt = System.DateTime.Now;
                long x = dt.ToFileTime();
                SetSeed((uint)(x    >>    16),    (uint)(x    %
4294967296));
            }
          . . . . . . . . . . . . . . . . . . . . . . . . . . . .
          }
      }
```

The XOR-MWC class in the implemented program takes the values resulting from the running of the MWC1 and MWC2 classes, calculates the result of the XOR operation applied to them and displays in the interface both the calculated values and the character corresponding to the extended ASCII code

```
      namespace RNG {
       class XOR-MWC {
              static void Main(string[] args) {
                  {
                     double CDF, CDF1;
                     double temp, temp1;
                     int E1 = 0, E2 = 0, E3 = 0, E4 = 0;
                     char a, b, c, d;
                     MWC1.SetSeedFromSystemTime();

                     int numReps = 255;
                     double failureProbability = 0.001;
                     int j0;
                     double[] samples = new double[numReps];
                     for (j0 = 0; j0 != numReps; ++j0)
                          samples[j0] = MWC1.GetUniform();

                     System.Array.Sort(samples);
                     double K_plus = -double.MaxValue;
                     for (j0 = 0; j0 != numReps; ++j0)
                          {
                              CDF = samples[j0];
                              temp = (j0 + 1.0) / numReps - CDF;
                              if (K_plus < temp)
                              {
                                  K_plus = temp;
                                  E1 = j0;
                              }
```

```
                        }
                    MWC2.SetSeedFromSystemTime();
                    int numReps1 = 255;
                    double failureProbability1 = 0.001;
                    int j1;
                    double[] samples = new double[numReps1];
                    for (j1 = 0; j1 != numReps; ++j1)
                         samples[j1] = MWC2.GetUniform();
                    System.Array.Sort(samples);
                    double K_minus = -double.MaxValue;
                    for (j1 = 0; j1 != numReps; ++j1)
                         {
                             }
                             Temp1 = CDF - (j1 + 0.0) / numReps1;
                             if (K_minus < temp1)
                             {
                                 K_minus = temp1;
                                 E2 = j1;
                             }
                         }
                     double            sqrtNumReps            =
Math.Sqrt((double)numReps);
                     double           sqrtNumReps1            =
Math.Sqrt((double)numReps1);
                     K_plus *= sqrtNumReps;
                     K_minus *= sqrtNumReps1;
                     a = Convert.ToChar(E1);
                     string a1 = Convert.ToString(a);
                     Console.WriteLine("Generator  Val.Zecimala
Val.ASCII");
                     Console.WriteLine("     MWC1:          {0}
{1}", E1, a);
                     b = Convert.ToChar(E2);
                     string b1 = Convert.ToString(b);
                     Console.WriteLine("     MWC2:          {0}
{1}", E2, b);
                     System.Threading.Thread.Sleep(200);
                     int key = E2;
                     char aa1 = (char)(a ^ key);
                     int aa11 = Convert.ToInt16(aa1);
                     string straa1 = Convert.ToString(aa1);
                     string gval1 = Convert.ToString(a1);
                     string gval2 = Convert.ToString(b1);
```

```
                    Console.WriteLine("\n (" + gval1 + ") XOR
" + "(" + gval2 + ") ==> Val. ASCII a Gen. XOR-MWC:  " + straa1
+ "\n");
      Console.WriteLine("=================================
================[" + idrow + "]\n");
              . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . .
                  }
                  Console.ReadLine();
           }
      }
```

In order to be used by the RPV8x application, the results of running the XOR-MWC generator are inserted in the *Keys* table of the *AES-DB* database made in MySQL.

```
      namespace RNG
      {
          static void Main(string[] args)
              {
                  int idrow = 0;
                  for (idrow = 0; idrow < 1000; idrow++)
                  {
          . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . .
                   SqlConnection      conex_DB      =      new
                   System.Data.SqlClient.          SqlConnection
                   ("Server=localhost;DataSource=.\\SQLEXPRESS;D
                   atabase=AES-DB; Trusted_Connection= True");
                   System.Data.SqlClient.SqlCommand comanda = new
                   System.Data. SqlClient.SqlCommand();
                   comanda.CommandType                          =
                   System.Data.CommandType.Text;
                   comanda.CommandText = "insert into dbo.Keys
                   (ID, PK1, PK2, PK3, PK4) values (" + idrow +
                   ", '" + gval1 + "', '" + gval2 + "', '" + gval3
                   + "', '" + gval4+"')";
                    comanda.Connection = conex_DB;
                    conex_DB.Open();
                    comanda.ExecuteNonQuery();
                    conex_DB.Close();
                   }
               }
```

The *idrow* variable sets the number of rounds for the algorithm and also the number of encryption keys generated. Because the RPV (Rijndael with Variable Parameters) encryption algorithm uses 128-bit (16-character ASCII) encryption keys, and the XOR-MWC generator generates only 8 bits after each run (one ASCII character). A 16-generator algorithm variant (XOR-MWC16x) was used to load the Keys table from the database. [6]

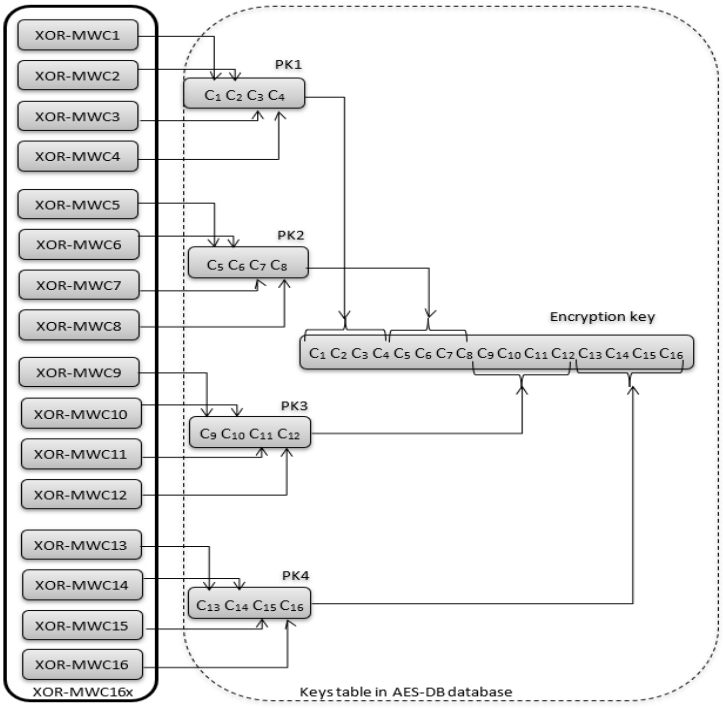


Fig. 1. Representation of partial key generation and encryption key

The XOR-MWC1-4 generators form the partial cryptographic key 1 (PK1), the XOR-MWC5-8 generators form the partial cryptographic key 2 (PK2), the XOR-MWC9-12 generators form the partial cryptographic key 3 (PK3) and the XOR-MWC13 generators forms the partial cryptographic key 4 (PK4). The partial cryptographic keys are entered in the *AES-DB* database, and by concatenation form the 16-character encryption key (C1-C16). Since the 16 generators use the same source to obtain the initialization values, they work in time lag, each with a delay of 0.4 seconds compared to the previous generator. This gives a time of 1.6 seconds for generating a partial key (PK1, PK2, PK3, PK4) and a total time of generating an encryption key of 128 bits of 6.4 seconds.

The *AES-DB* database is installed locally or on a secure server. In the tested version we used a *MySQL* sever installed on the same machine as the RPV application.

*Table 1*

**Keys table contents**

| ID | PK1 | PK2 | PK3 | PK4 |
|---|---|---|---|---|
| 0 | this | isas | ecre | tkey |
| 1 | r8`d | zMo[ | U?]v | EdTq |
| 2 | k1)5 | E0P< | WDZj | &x"a |
| 3 | :EZ[ | \|RA@ | &;8y | E*Vo |
| 4 | %P"u | #?\|h | bFd6 | o)U+ |
| 5 | !;I5 | V#w4 | 8LA~ | cxQ} |
| 6 | }e2u | F)S$ | cATt | s+$F |
| 7 | G%;m | lJY& | o!t0 | <Bt! |
| 8 | dG_y | dFhq | g^ a | Re9y |
| 9 | [T`v | 3nOu | j. E | xK]d |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 998 | ~\|s! | {0$g | "0-U | Zg c |
| 999 | "zgU | `g$% | E&Kf | >Z{n |

*Keys* table keeps the partial cryptographic keys (PK1, PK2, PK3, PK4) from which the encryption keys are formed.

*Table 2*

**Parameters for modifying the structure of the RPV algorithm**

| ID | Param_SR | Param_MC |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

The values in *Table 2* are read by the RPV8x algorithm during its run, setting the working modes for the *ShiftRow* (Param_SR column parameters) and MixColumns (Param_MC column parameters) transformations of the algorithm. Microsoft SQL Server Management Studio Express can be used to view and edit tables and their contents, or any other application capable of managing the content of SQL databases.

Unlike classic implementations of the Rijndael algorithm, which require you to upload a cryptographic key from a user-friendly file, RPV8x provides an operational security measure, with users not having access to the encryption key or the parameter IDs. [**7**]

## 3. Encryption and decryption component

The RPV8x encryption and decryption program is based on the RPV algorithm and has a friendly, easy-to-use interface, and in the test version provides detailed information on how to choose pseudo-random operation using implemented PRNGs. The two implemented modules, the *Encryption Module* and

the *Decryption Module* have search and save file buttons (Open File and Save File), text fields showing the path of selected or saved files, as well as encryption / decryption and reset application option. Also, in the test version there are buttons to check the parameters used (Check Encryption / Decryption Parameters). In addition, the test interface provides information about:

- ✓ S-Box pair ID used in encryption;
- ✓ *ShiftRows* and *MixColumns* transform run parameter IDs;
- ✓ Partial key IDs selected from the database;
- ✓ Session ID.

The RPV algorithm is a symmetric cipher that works with 128-bit-long blocks of data and encryption keys and is a development of the Rijndael algorithm developed in 1998 by two Belgian cryptographers. Together with the MWC generators that have the role of introducing the pseudo-random character of the operation and the *AES-DB* database in which some of the parameters necessary for running are stored, the RPV algorithm forms a complex cryptographic system.

Due to the properties of the invertible polynomials underlying the formation of S-Boxes and the possibility of implementing different ways of working in the *ShiftRows* and *MixColumns* transformations of the Rijndael algorithm, the RPV algorithm can work in 8 different modes (of these 8 modes of operation, one is similar to that of Rijndael). [8]

By encrypting with the RPV8x application, encrypted texts are obtained for the same clear text and the same encryption key, modifying the diffusion and confusion factors of the algorithm both by changing the pseudo-random parameters of the *ShiftRows* and *MixColumn* transformations, and by accessing in direct or reverse order of S-Boxes. At the same time, even if in the case of a brute force attack on the algorithm the time remains the same, being determined only by the computing power of the attacker, in the case of linear and differential cryptanalysis there is the problem of determining how the algorithm works, thus increasing 8 times the theoretical resistance time of the algorithm in the case of these types of attacks (hence the name of the application -RPV8x).

There are three fundamental criteria taken into account for choosing the Rijndael cipher as the basis for the development of the RPV algorithm: [9]

- resistance against all known types of attack;
- speed and compact code structure on many platforms;
- simplicity of design.

In the case of most block digits, the round transformation has a Feistel structure. This assumes that some of the intermediate state bits are transposed unchanged to another position. The transformation of the round into Rijndael does not have such a structure but is composed of three uniform invertible transformations, called layers. The specific choices of the different layers are largely based on the application of the Wide Trail Strategy, a design method

designed to withstand linear and differential cryptanalytic attacks. In Wide Trail Strategy, each layer has its own property: [10]

- Linear mix layer: guarantees a high degree of diffusion over multiple rounds;
- Non-linear layer: ensures the parallel application of S-Boxes with optimal properties for non-linearity cases;
- Key addition layer: performs a simple XOR between the round key and the intermediate state.

A key addition is made before the first round, as any of the layers can be discovered without the need to know the key in case of attacks on known plain text. The addition of the key in the initial or terminal stage is successfully applied in several designs such as: IDEA, SAFER and Blowfish. [11] Through the *SqlCommand* and *SqlDataReader* functions, the application connects to the Keys table in the database and reads the 4 ASCII characters corresponding to the $ID_{k1-4}$ index line and $PK_{1-4}$ columns.

```
            SqlCommand comanda_PK1 = connex.CreateCommand();
            SqlDataReader citeste_PK1;

            connex.Open();
            commanda_PK1.CommandText = "select PK1 from dbo.Keys
                  where ID=" + IDPK1;
                  citeste_PK1 = commanda_PK1.ExecuteReader();
                  PK1 = Convert.ToString(citeste_PK1.Read());
                  for (int tk1 = 0; tk1 < citeste_PK1.FieldCount;
tk1++)
                      linie_PK1                                   +=
citeste_PK1.GetValue(tk1).ToString();
                  connex.Close();
          . . . . . .
. . . . .
      private void FormEDK()
              {
                  EDK  =  linie_PK1  +  linie_PK2  +  linie_PK3  +
linie_PK4;
              }
```

The codes for reading the other 3 partial keys (PK2, PK3, PK4) are similar to this one. After reading the partial keys, the encryption / decryption key EDK (Encryption / Decryption Key) is made by concatenation.

In the idea of using a unique encryption key each time the application is run, the C# implementation of the MD5 algorithm is used to hash the generated key. The thus obtained hash is checked in the BlackList table of the database and if it is

identified in the list, the PRNGs are automatically reset, performing a new encryption key.

```
SqlConnection connex1 = new System.Data.SqlClient.SqlConnection
("Server= localhost;DataSource=.\\SQLEXPRESS; Database=AES-DB;
Trusted_Connection= True");

                  SqlCommand            commanda_BK            =
connex1.CreateCommand();
                  SqlCommand            commanda_BK1           =
connex1.CreateCommand();
                  SqlCommand            commanda_BK2           =
connex1.CreateCommand();
                  SqlCommand            commanda_BK3           =
connex1.CreateCommand();
                  SqlDataReader citeste_BK;
                  SqlDataReader citeste_BK1;
                  SqlDataReader citeste_BK2;
                  SqlDataReader citeste_BK3;

System.Security.Cryptography.MD5CryptoServiceProvider EDKx = new
System.Security.Cryptography.MD5CryptoServiceProvider();
                  byte[]                EDKbs                  =
System.Text.Encoding.UTF8.GetBytes(EDK);
                  EDKbs = EDKx.ComputeHash(EDKbs);
                  System.Text.StringBuilder   EDKsMD   =   new
System.Text.StringBuilder();
                  foreach (byte b in EDKbs)
                  {
                      EDKsMD.Append(b.ToString("x2").ToLower());
                  }
                  EDKMD5 = EDKsMD.ToString();
                  connex1.Open();
                  commanda_BK.CommandText  =  "select  ID  from
dbo.BlackList where RejectedKeys ='" +EDKMD5 + "'";
                  citeste_BK = commanda_BK.ExecuteReader();
                  bool IDkbl = citeste_BK.Read();
                  connex1.Close();
                  if (IDkbl == false)
                  {
                      connex1.Open();
                      commanda_BK1.CommandText = "select max(ID)
from dbo.BlackList";
```

```
                    citeste_BK1                                =
commanda_BK1.ExecuteReader();
                    string              valIDst                =
Convert.ToString(citeste_BK1.Read());
                    for    (int    idbl    =    0;    idbl    <
citeste_BK1.FieldCount; idbl++)
                        valID                                  =
citeste_BK1.GetValue(idbl).ToString();
                    int maxIDbl = Convert.ToInt16(valID);
                    connex1.Close();
                    connex1.Open();
                    maxIDblp1 = maxIDbl + 1;
                   commanda_BK2.CommandText   =   "insert   into
dbo.BlackList (RejectedKeys) values ('" + EDKMD5 + "')";
                    citeste_BK2                                =
commanda_BK2.ExecuteReader();
                    connex1.Close();
```

Encryption is performed this way: [12]

1. PRNG creats $ID_{k1}$, $ID_{k2}$, $ID_{k3}$, $ID_{k4}$, $ID_s$ and $ID_{srmc}$ indexes in the database tables;
2. AE reads the partial keys corresponding to the indexes $ID_{k1}$, $ID_{k2}$, $ID_{k3}$, $ID_{k4}$ from the *AES-DB* database and composes the encryption key *K*;
3. AE reads the pair of replacement tables using $ID_s$;
4. AE reads working parameters with $ID_{srmc}$ identification number for *ShiftRows* and *MixColumns* transformations in the database;
5. AE uses the encryption key formed in step 1, the pair of substitution tables, and the *ShiftRows* and *MixColumns* transform variants to encrypt plain text;
6. AE creates an $ID_{ses}$ session identification number, consisting of the concatenation of $ID_{k1}$, $ID_{k2}$, $ID_{k3}$, $ID_{k4}$, $ID_s$ and $ID_{srmc}$, which it adds to the encrypted message to be transmitted.

Encryption is done by calling the methods in which the transformations are implemented, according to the source code presented below.

```
        int Encrypt(int[][] a, int[][][] rk)
              {
                  int r;
                  AddRoundKey(a, rk[0]);
                  for (r = 1; r < ROUNDS; r++)
                  {
                      SubBytes(a, S);
                      ShiftRows(a, 0);
```

```
                MixColumns(a);
                AddRoundKey(a, rk[r]);
            }
            SubBytes(a, S);
            ShiftRows(a, 0);
            AddRoundKey(a, rk[ROUNDS]);
            return 0;
        }
```

The information contained in the previously formed $ID_{ses}$ is required in the decryption process and even if the $ID_{ses}$ are not added to the encrypted message, other methods of its secure transmission between the corresponding entities can be used. For added security, data packets can be overwritten using a secret key and a pair of predefined replacement tables. [13]

## 4. Conclusions

Depending on the situation and requirements, information security methods need to be constantly adapted to respond effectively to individual needs. In order for transactions to be secure and information security objectives to be met, increasingly complex cryptographic protocols and techniques have been developed. These, in conjunction with compliance with procedural techniques, can ensure the desired level of security.

Today, cryptographic applications are part of everything that happens in communications networks, regardless of the medium of transmission (terrestrial radio networks, metal cable, fiber optics or satellite networks). The algorithms used so far have been and are constantly "bombarded" by attempts made by specialists to prove their vulnerabilities, before hackers take advantage of their weaknesses.

Cryptographic solutions based on symmetric block algorithms are a special category, characterized in particular by a very good encryption speed. Their design (and not just this type of algorithm) must meet two essential conditions: they must be *secure,* and they must be *fast*. In recent years, great efforts have been made to design algorithms that best meet the two stated conditions. The partial key generation component (XOR-MWC generator) used to load the database and to create the encryption keys, based on the MWC generator, is distinguished by the fact that it is able to pass NIST tests, despite being taken individually, the two generators in its composition do not succeed in this. The RPV algorithm, as part of the complex cryptographic system, is based on the robustness demonstrated by the Rijndael algorithm in the face of linear and differential cryptanalytic attacks and is an optimized variant of it, by implementing different ways of working. These modes allow you to obtain 8 variants of encrypted text, for the same clear text and the same encryption key, thus increasing the theoretical cryptanalysis time for

attacks on the algorithm. In the case of brute force attacks, there is no improvement, which is done by checking all possible encryption keys. [14]

The constant evolution of cryptanalytic methods, supported by continuous technological development, determines the entire community of cryptologists to constantly look for ways to optimize current security solutions, or to develop new ones.

## R E F E R E N C E S

[1]. Information Security Management, ISO/IEC 27001, https://www.iso.org/isoiec-27001-information-security.html, accesed on October 2021.
[2]. Christoph D., Maria E., Hannes G., Stefan M., Florian M., Robert P., Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures. Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11273). Springer, 315–342, 2018.
[3]. Fan Z., Xiaoxuan L., Xinjie Z., Shivam B., Wei H., Ruyi D., Samiya Q., Kui R., Persistent Fault Analysis on Block Ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018, 3, 150–172, 2018.
[4]. Zeng G., Quantum Private Communication. Higher Education Press, Beijing and Springer-Verlag Berlin Heidelberg, pp. 135-137, 2010.
[5]. Marsaglia G., Yet another RNG. Posted to electronic bulletin board at sci.stat.math, August, 1994.
[6]. Galice S., Minier M., Improving Integral Attacks Against Rijndael-256 Up to 9 Rounds. Africacrypt 2008, LNCS 5023, pp.1–15. Springer–Verlag, 2008.
[7]. Nakahara J., Freitas D., Phan R., New Multiset Attacks on Rijndael with Large Blocks. Advances in Cryptology — Mycrypt 2005, LNCS 3715, pp. 277–295, Springer–Verlag, 2005.
[8]. Dassance F., Venelli A., Combined Attacks on the AES Key Schedule. Cryptology ePrint Archive: Report 2012/098, http://eprint.iacr.org/2012/098, accessed on October 2021.
[9]. Koblitz A. H., Koblitz N., Menezes A., Elliptic curve cryptography, the serpentine course of a paradigm shift. eprint.iacr.org/2008/390, 2008.
[10]. Schwartz B., Zaitsev P., Tkachenko V., Zawodny J. D., Lentz A, Balling D. J., High Performance MySQL, Second Edition. O'Reilly Media, Inc. ISBN: 978-0-596-10171-8, Iunie, 2008.
[11]. Ji-Peng X., Xue-Cheng Z., Xu G., Ultra-low power S-boxes architecture for AES. The journal of China Universities of post and telecommunications. Vol. 15, issue1, Martie, 2008.
[12]. IEEE P1363, IEEE Standard Specifications for Public-Key Cryptography. Computer Society, New York, USA. Disponibil online la: http://grouper.ieee.org /groups/1363, 2004.
[13]. Behrouz A. F., Cryptography and network security, TATA-McGraw Hill Publication 2007.
[14]. NIST Special Publication 800-131, Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes, Federal Information Processing Standards Publication (FIPS PUB) 197, National Institute of Standards and Technology (NIST), Januarie, 2010.