Evaluation Framework for Security and Resource Consumption of Cryptographic Algorithms

Cristina-Loredana Duta, Bogdan-Costel Mocanu,
Florin-Alexandru Vladescu, Laura Gheorghe, Nicolae Tapus

Department of Computer Science and Engineering, University Politehnica of Bucharest, Romania
E-mail: cristina.duta.mapn@outlook.com, bogdan_costel.mocanu@cti.pub.ro,
florin_alexandru.vladescu@cti.pub.ro, laura.gheorghe@cs.pub.ro, nicolae.tapus@cs.pub.ro)

Abstract: Today, security is necessary when transmitting confidential information over the network. One of the most important ways to provide data confidentiality is through cryptography. In this paper, we present a framework for testing and evaluating cryptographic algorithms. When evaluating block and stream ciphers, some basic properties should be tested: correct functional testing, passing statistical randomness testing, having good substitution boxes (S-boxes) and good permutation boxes (P-boxes) in their construction and providing a good throughput. The proposed framework receives the cryptographic algorithm as input and evaluates: the provided test vectors, the randomness of the generated data, the properties of the S-boxes and the P-boxes, the performance in terms of speed and throughput and provides a result, whether or not the algorithm is secure and can be used in cryptographic applications.

Keywords: cryptography, randomness tests, S-boxes, P-boxes, performance evaluation

1. INTRODUCTION

Nowadays, due to the large amount of data that is processed or transferred between individuals or companies, the security domain is becoming one of the most important issues. Data must be kept secure and nobody, except the authorized entities, can have access to confidential information. The solution for this requirement is to use cryptography.

Cryptography protects information by transforming it into unreadable format (encryption). This format hides a message that should be understood only by the intended receiver (after decryption). Also, with the help of cryptography, transmission channels that carry sensitive information can be protected against unauthorized access.

Because of the increasing importance of data that is being exchanged over the Internet, a solution must be found in order to provide the necessary protection against the attacks. Because of this, evaluating cryptographic algorithms is an important task in order to achieve a high level of information security.

Security for cryptographic systems is ensured if some basic properties are taken into consideration when developing a cryptographic algorithm. For instance, the impossibility of distinguishing a primitive (block or stream cipher) from a random mapping is an important aspect. Based on the results of statistical randomness tests, it can be determined the suitability of the algorithm to be used in specific applications as a random number generator.

The purpose of this project is to determine whether the evaluated cryptographic algorithm is capable of providing a certain level of security for the transmitted information. Our goal is to create a complex evaluation framework that incorporates testing functions for all important cryptographic properties. In this way, the users of the framework can see if their algorithm is vulnerable or not and decide if it can be used in the application it was designed for. For instance, statistical testing module can show if an algorithm is suitable to be a random number generator, the S-boxes testing module can identify a weakness in the construction of the algorithm, the performance evaluation module can specify if the algorithm is good in software or in hardware implementations.

The paper is organized as follows. Related work is presented in Section 2. Section 3 gives an overview of the architecture for the evaluation framework. Section 4 offers details about our implementation of the framework. In Section 5 various metrics taken into consideration for the algorithms are given and the experimental results are presented. Section 6 describes our conclusions and future work.

2. RELATED WORK

To offer a better perspective about the utility and importance of the generic evaluation framework we developed, this section presents various methods and techniques for evaluating cryptographic algorithms and the results obtained by other solutions.

2.1 Cryptographic Algorithms

Cryptographic algorithms are classified in symmetric algorithms (also known as secret key algorithms) and asymmetric algorithms (public key algorithms).

When symmetric algorithms are used, the sender and the receiver have the same key. The key is agreed upon between the both of them and it is used for encryption and decryption of the message. Some of the best known cryptographic algorithms included in this category are: Advanced

Encryption Standard (AES), Data Encryption Standard (DES) and 3DES.

When asymmetric algorithms are used, each user has a public and a private key. The public key of the receiver is known to everyone and is used by the sender to encrypt the message and the private key is used by the receiver to decrypt message. The public-key system is constructed in a way that calculating one key, for instance the private key, from the other key, the public key, is computationally infeasible. Some of the most important applications of public-key cryptography are Elliptic Curve cryptography (ECC), PGP and the public-key infrastructure (PKI).

Symmetric algorithms can be divided into block ciphers and stream ciphers. Block ciphers are algorithms that permute N-bit blocks of plaintext data, combine them with the secret key and generate at the end N-bit blocks of encrypted data. Stream ciphers typically operate serially by generating a stream of pseudo-random key bits, called keystream. This keystream is XOR-ed with the data to encrypt or decrypt in a bit by bit manner. In this paper, we focus on evaluating symmetric algorithms.

2.2 Performance Evaluation

(Tamimi, 2005) makes a performance comparison between symmetric key algorithms such as DES, 3DES, AES and Blowfish taking into consideration different sizes and contents of input data. The evaluation was performed on two different hardware platforms: P-II 266 MHz and P-4 2.4 GHz.

(Shah et al., 2011) propose different performance factors such as: visual degradation after encryption, tunability and computational speed. (Singh et al., 2011) discuss the performance metrics for the symmetric key algorithms and present their results.

(Arora et al., 2012) has chosen algorithms such as AES, DES and Blowfish. In this case, the evaluation is dependent on the type of data. The input files can be of type .exe, .doc, .wmv and .avi and the comparison is based on the calculated throughput.

(Ramesh et al., 2012) try to determine how algorithms perform on web servers. The tests were performed on platforms such as Internet Explorer, Mozilla Firefox, Opera and Netscape Navigator. They reached the conclusion that DES is suitable for Internet Explorer, RC6 for Mozilla Firefox and UR5 for Opera, in terms of provided throughput.

Performance comparison between different encryption algorithms implemented in .NET framework was made by (Dhawan, 2002). The algorithms compared were DES, 3DES, RC2 and AES. The parameters used for comparison were the number of requests processed per second and the response time for different user-load situations. The best result is obtained in the case of AES.

Compared with all these papers, our framework brings new and original functionalities in terms of performance evaluation: it can evaluate any symmetric cryptographic algorithm (block cipher or stream cipher) requiring as input only the .dll file containing the algorithm. It calculates its execution time, speed and throughput for different file sizes such as 1KB, 100 KB, 500 KB, 1 MB, 100 MB, 500 MB, 1 GB, compares the results with threshold values from Crypto++ benchmarks[1] and specifies if the evaluated algorithm has a small/good/high throughput.

2.3 S-box Testing

The foundations of any cipher system are two fundamental concepts: confusion and diffusion, which were identified by (Shannon, 1949). All modern cryptographic primitives have in their construction a collection of S-boxes that ensure confusion and P-boxes, which provide diffusion, by spreading out the output bits to S-boxes included in the next round of the algorithm. It can be said in this context, that the S-boxes are an integral part of symmetric key cryptosystems, which are used to obscure the relation between the plaintext and the cipher text. In general, the strength of symmetric ciphers is determined by "properly" designed S-boxes.

The theory regarding S-boxes has appeared as an attempt to formalize defenses that can be included into S-boxes to strengthen the algorithm against cryptographic attacks. For instance, if a new attack is discovered, this leads to a new design criterion that shows what proprieties the S-box must have to resist it. When creating cryptographic algorithms, there is a set of design criteria, which are considered to be essential.

If the S-box does not satisfy one of the criteria, the design of the algorithm based on that S-box may be cryptographically weak (it can be attacked). The set of design criteria essential for S-boxes are defined further on.

Completeness criterion - This criterion was discovered by (Kam and Davida, 1979) and is, in general, intended for the entire cryptographic design rather than a single S-box. If the criterion is not satisfied, the attacker can use methods such as "divide and conquer" to inspect the design.

Balance criterion - According to the balance criterion each Boolean vector responsible for the S-box has the same number of 0's and 1's. If this criterion is not satisfied, then some output strings are more probable than others, which leave the design vulnerable to attacks which exploit the non-uniformity of output strings using probabilistic distributions.

Nonlinearity criterion - This criterion requires the S-box not to be a linear mapping from input to output. If this criterion is not satisfied, then the cryptosystem will be susceptible to attacks.

Propagation criterion - (Webster and Tavares, 1986) introduced the Strict Avalanche Criterion (SAC). According to this criterion if only one input bit is changed, than half of the output bits will be changed. If the SAC criterion is not satisfied, an attacker can determine correlations between the plaintext and the cipher text using methods such as known plaintext-cipher text attack.

[1]http://www.cryptopp.com/benchmarks.html

Good XOR profile - The XOR profile is a table, which presents the differences between the input and output of the S-boxes. This criterion is not very restrictive, because the designer of the S-boxes is the one that has to make sure that the XOR profile does not contain entries with large numbers. If the XOR profile is not very good, then an attacker can apply a differential cryptanalysis attack (Heys, 2004), and recover pieces of the plaintext.

If a designer understands how to create cryptographically good S-boxes, the new S-boxes can be used to develop new private-key cryptosystems and new methods of generating cryptographically good S-boxes are always in demand.

(Stoianov, 2010) has developed software for testing square S-boxes. In this project, the author divides the software into two components entitled "Data processing" and "Checks and Tests".

(Angraini et al., 2013) have analyzed the S-boxes from Whirlpool and SEED algorithms, but only in terms of the strict avalanche criterion (SAC) test. Their study tries to create a new S-box, based on AES's one and to determine whether or not if satisfies SAC.

(Saarinen, 2011) presents the results obtained after analyzing all 4x4 bit S-boxes. The paper includes a detailed description of the properties of the S-boxes and presents a set of S-boxes that have ideal cryptographic properties.

Our framework includes a module for testing S-boxes, because as far as we know, there is no software available today that can be used to verify all the properties of any S-box. Testing the S-boxes is very important because it allows detecting and identifying vulnerabilities in the construction of the cryptographic primitive if they exist.

2.4 P-box Testing

A domain in which little research was made is the one of analyzing the properties of Permutation boxes (P-boxes), related to cryptography area. A permutation box is a method of bit shuffling used to permute bits across S-boxes inputs.

(Brown et al., 1990) have studied the design of permutation boxes. In the article, they explain the design criteria for the permutations in DES type cryptosystems and they establish some rules that must be respected in order for the permutation to be correct. Due to the small number of publications regarding the properties of the permutations, we explored this research subject and our evaluation framework includes a module for testing P-boxes.

2.5 Statistical Testing

In general, the security of cryptographic system is strongly related to randomness, because the output of these systems can be observed by any adversary and should be seen as a sequence of random values, which hold the secret, but don't reveal any sensitive information. Generating high-quality randomness is an essential step of many cryptographic operations and sometimes, the significance of well designed cryptographic pseudo-random data generators is deprecated.

Because some of the proprieties of random sequences are statistical, these can be measured and evaluated using statistical randomness tests.

In general, statistical tests use a binary sequence as input and determine whether or not the null hypothesis, denoted H0, is accepted or rejected. H0 considers the input sequence to be random. Because randomness tests are probabilistic, two types of errors can occur: type I error (the data is random, but H0 is rejected) and type II error (the data is non-random, but H0 is accepted).

The probability for a type I error to occur is called *level of significance* of the test, and is denoted by α. In general, the statistical tests return a number between 0 and 1, which is called P-value. If P-value is greater than α, then the H0 is accepted, else it is rejected. Based on these notions, it can be seen that the level of significance can have different values, according to the specific of the application.

A collection of statistical randomness test that are created to evaluate the randomness proprieties of sequences is called a test suite. Several test suites are available today such as: NIST (Rukhin et al., 2010), TestU01 (L'Ecuyer et al., 2007), Diehard (Marsaglia, 2003), and ENT (Walker, 2008). Diehard suite was developed by George Marsaglia, while the NIST test suite was developed by the Computer Security Division and The Statistical Engineering Division at National Institute of Standards and Technology (NIST). Crypt-XS suite (Gustafson et al., 1994) was developed at the Information security Research Center at Queensland University of Technology in Australia.

It is very important to evaluate the outputs generated by cryptographic algorithms using statistical randomness tests. For instance, even when the AES competition took place, the candidate block ciphers were evaluated by (Soto, 1999). In order for the results of the tests to be relevant, sequences of at least 106 bits length were necessary. This was achieved by concatenating the outputs of the candidate algorithms. Nine different methods were proposed to generate large number of data stream from a block cipher and then the resulting streams were tested using the statistical tests from NIST (Rukhin et al., 2010).

(Cook et al., 2009) have created a new method entitled "elastic block cipher method" and they present in the paper examples based on AES, Camellia, MISTY1 and RC6. The two versions, the original and the elastic one, were evaluated using NIST statistical tests and randomness tests used for AES candidates.

(Chen et al., 2009) present a new statistical test for block ciphers. The new test is applied to Rijndael, Camellia and SMS4 algorithms in order to determine if these have good statistical properties.

The advantage brought by our project is that, as far as we know, a generic framework to evaluate also the randomness properties of any cryptographic algorithm (stream cipher or block cipher) has not been publicly presented or described.

2.6 Functional Testing

Another important feature provided by our framework is the module of functional testing, for which as far as we know, there isn't any software instrument developed. Having the test vectors provided by the developers, the application automatically compares them with the results obtained by it. It uses the evaluated algorithm for encryption/decryption of the specified text and can determine if they coincide or not. In this way, the user can perform functional testing to make sure the algorithm is implemented correctly before he proceeds to the other evaluation mechanisms.

Using our framework, we managed to evaluate algorithms such as AES, DES, 3DES, TEA, Camellia, LEX, Sosemanuk, HC-128and RC4 taking into consideration performance, S-box testing, P-box testing, statistical testing and functional testing.

3. FRAMEWORK ARCHITECTURE

This section presents the architecture and the design principles of the evaluation framework.

The proposed framework for cryptographic algorithms has four main functionalities. The first one is performance evaluation of the cryptographic algorithms. This functionality consists of measuring the speed, throughput and clock cycles. The second functionality of the framework is the evaluation of the randomness properties of cipher outputs. For this functionality, the NIST statistical suite was implemented. Another set of tests is the set of statistical tests used in evaluating AES candidates. The third functionality of the framework is the evaluation of substitution boxes (S-boxes) and the last functionality provided is the evaluation of the P-boxes.

The framework includes several modules such as functional testing module (which verifies the test vectors provided by the developer of the algorithm), statistical testing module (which verifies if the output of the cryptographic algorithm satisfies randomness properties), S-box testing module (which evaluates the properties of the S-boxes included in the algorithm construction, if they exist), P-box testing module (which verifies the properties of the P-boxes included in the algorithm) and the performance evaluation module (which compares the speed and throughput of the evaluated algorithm with the speed and throughput of well known cryptographic primitives such as AES and RC4).

Concerning the interactions between the user and the framework a use case diagram is presented in Figure 1. The use case diagram includes 6 actions.

In the initialization action, the operator loads the cryptographic algorithm into the framework and sets the parameters necessary for the evaluation. The initialization and configuration process implies the loading of the target algorithm to be evaluated in a specific form. The algorithm must be implemented in a common programming language such C/C++, C# and compiled as a dynamic library (.dll). The parameters are set by the user in the GUI depending on the description of the algorithm. The parameters that must be configured are standard parameters for cryptographic algorithms such as key size, block size, IV size, if necessary.

After initialization, the user can perform the evaluation. The processing activity represents the effective testing for the algorithms. Therefore, in this activity are performed the functional tests of the algorithm implementation by using test vectors provided by the developer of the algorithm. The performance evaluation module tests the speed and throughput of the algorithm. The statistical evaluation module ensures testing of the statistical properties of the algorithm output. This task is performed by applying the NIST statistical suite. The evaluation of the S-Boxes is performed by testing the properties of good S-boxes such as: balance, nonlinearity, completeness, propagation criteria and good XOR profile.

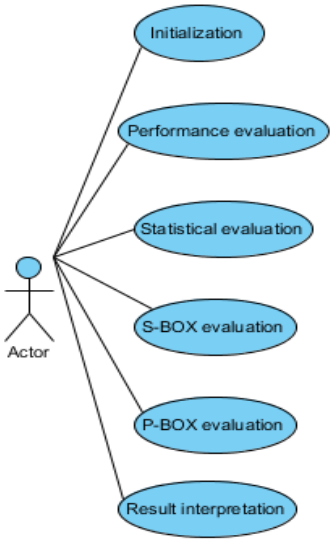

Fig. 1. Use Case Diagram.

Finally, the evaluation of P-boxes is a bit different because permutations alone do not have good cryptographic properties. Because permutations are from mathematical point of view bijective functions, the framework will test their properties. The most important test is to determine whether the permutation has an inverse. To do this, the module verifies if the permutation used in the algorithm has fixed points and if it is circular.

After the algorithm is tested, the results of the evaluation are analyzed and the results are obtained. Based on the final results, the user can decide if he can use the algorithm in cryptographic applications and it is secure or not.

The evaluation framework is structured as a finite state machine (FSM). The FSM is composed of 7 states which are shown in Figure 2. The basic idea of the FSM is that when the framework is properly configured for a specific cryptographic algorithm all the program logic can be applied for testing.

The states of the framework are: load algorithm, configure parameters, functional testing, performance evaluation, statistical evaluation, S-box evaluation, P-box evaluation.

In the load algorithm state, the algorithm for evaluation is loaded in a .dll format. After this state, the FSM enters in the configuration state. Here are configured the following parameters: algorithm type, key size, block size and IV size.

The next state represents the functional testing phase. In this state, the algorithm is functional tested with test vectors.

In the statistical evaluation state are performed statistical tests from the NIST statistical suite to evaluate the randomness of the output of the algorithm being tested.

In the performance evaluation state, the speed and throughput of the evaluated algorithm is calculated and compared with threshold values.

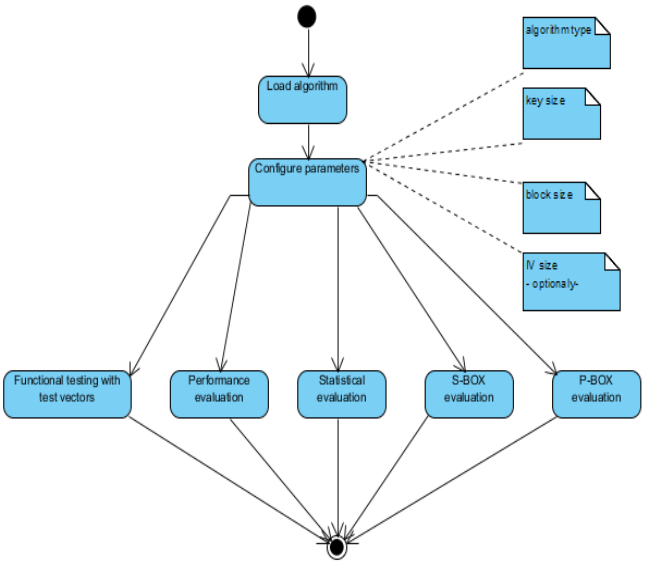

Fig. 2. State machine diagram.

In the S-box evaluation state are tested the properties of S-boxes and finally in the P-box evaluation state are tested the properties of the P-boxes.

4. IMPLEMENTATION

In this section, we present the implementation of the evaluation framework of cryptographic algorithms.

The framework was designed as a thick client application developed under Microsoft Windows Operating System and with .NET 4 Framework. The thick client design was chosen based on the following reasons: the framework is deployed on a standalone computer, the framework has to be easy to use and the cost of deployment and maintenance has to be low.

The framework is developed in C# under the .NET 4 Framework as a standalone Windows Forms application. The framework is implemented using the MVP (Model View Presenter) design pattern.

The MVP design pattern is an evolution of the Model View Controller (MVC) design pattern. MVP consists of the following layers: Graphical user interface (GUI), Application logic, Model business objects.

In MVP design pattern, the model layer is isolated from the view layer, in contrast with MVC where the model and view layers have direct links between each other.

4.1 GUI Description

The GUI is designed using general Windows Forms with several tabs for the main activities of the application. The GUI is depicted in Figure 3.

Each tab from GUI represents a module of the framework. The first tab is for evaluating symmetric algorithms, as it can be seen in Figure 3. The user must select the type of the algorithm (stream or block cipher), introduce the key, block and optionally the IV length, load the .dll file and select the format of the data.

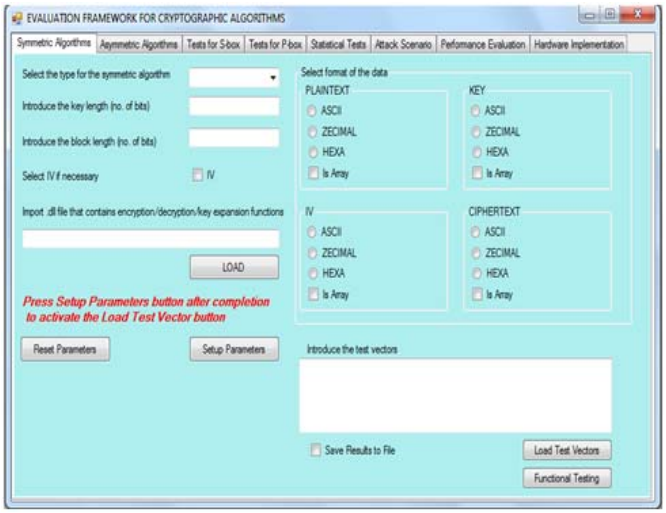

Fig. 3. GUI of the Framework.

The third tab is for the S-boxes testing module, shown in Figure 4, and the next tab is for P-boxes testing module, presented in Figure 5 (the user loads the S-box or P-box from a text file).

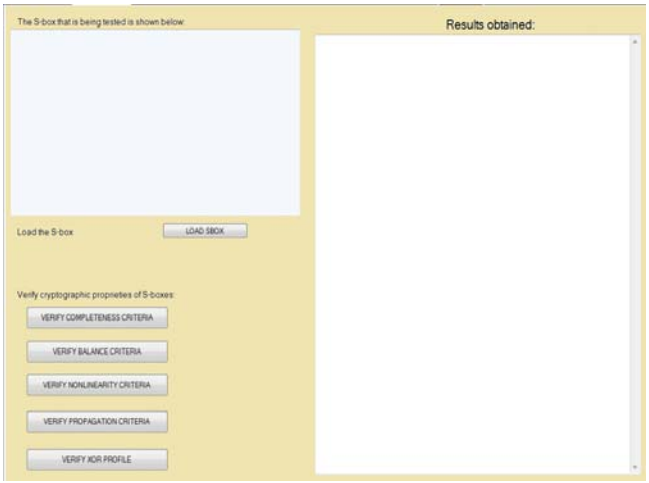

Fig. 4. S-box testing module.

The fifth tab, shown in Figure 6, includes the module for statistically testing the algorithms.

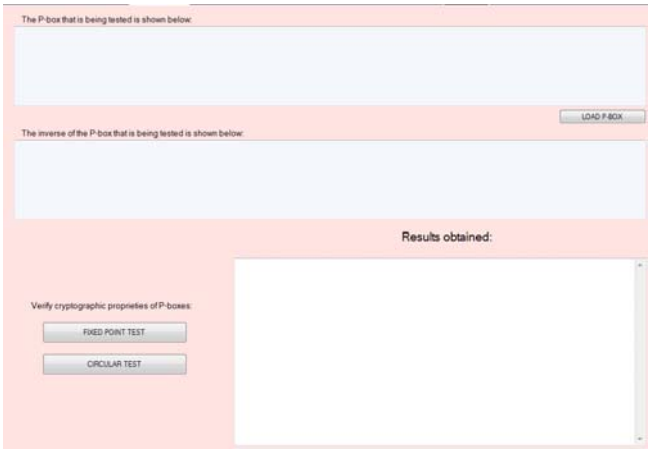

Fig. 5. P-box testing module.

The sixth tab, presented in Figure 7, represents the Performance evaluation module, which is used to determine the speed and throughput of the tested algorithm and compare it with the speed and throughput of standardized algorithms such as AES, RC4.

Fig. 6. Statistical testing module.

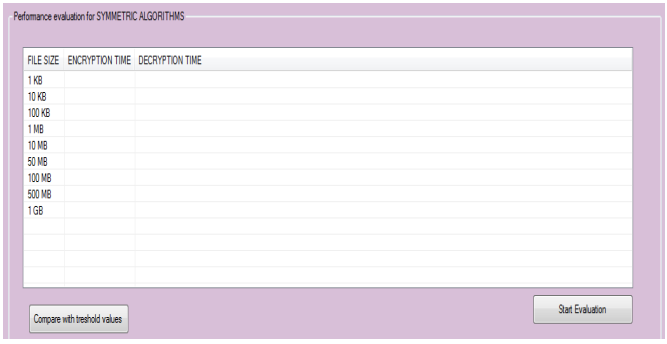

Fig. 7. Performance evaluation module.

4.2 Application logic

This layer contains the logic of the framework, such as the implementation of the NIST Suite for statistical properties of randomness, the implementation of statistical tests applied for AES candidates, the implementation of functions to calculate the speed and throughput of the algorithm, the implementation of functions to verify the properties of good S-Boxes and the implementation of functions to verify the properties of permutations.

For the functional testing module to work, the user must configure the parameters. For example, if we want to test DES algorithm, first, the type of algorithm is selected, which in our case is "block cipher". Then the length of the key is written (64 bits), the length of the block (64 bits) and the length of the IV if necessary (which in our example is not used). Then the user must import the .dll file containing the cryptographic algorithm being tested. Also, mandatory for this part, is to select the format of the data (it can be in ASCII format, in decimal format or in hexadecimal format).Taken our example, with DES, the DECIMAL radio button is selected for all three parameters – plaintext, key and ciphertext.

To activate the "Load Test Vectors" button, the user must press the "Setup parameters" button. If the parameters are not correct, for instance, the file loaded is not a .dll file, or if instead of numbers for the key/block length are letters introduced, the application will show corresponding messages and will not enable the "Load Test Vectors" button.

If the parameters are correct, the user can load the file containing the test vectors and can apply the functional testing function. This module allows the user to save the results obtained into a text file.

The statistical testing module includes implementations for the NIST statistical suite and for the randomness tests used for AES candidates.

There are 15 NIST tests implemented, each verifying different properties of the output of the cipher. These are: frequency test, block frequency test, runs test, cumulative sums test, longest run of 1's test, binary matrix rank test, discrete Fourier transform test, non-overlapping template test, overlapping template test, universal statistical test, linear complexity test, serial test, approximate entropy test, random excursions test, and random excursions variant test.

The nine tests used to evaluate the AES candidates are: key avalanche test, plaintext avalanche test, plaintext-ciphertext correlation test, cipher block chaining mode test, random plaintext-random key test, low density key test, low density plaintext test, high density plaintext test, and high density key test.

The user can select which tests he wants to apply, by checking the box corresponding to it, or it has the option to run all of them. The framework stores all the results of the statistical tests in files and provides charts for each NIST test that show the uniform distribution of the obtained values.

The S-box testing module includes five functions, each corresponding to a property of S-boxes. The module allows the user to load the S-box written as a matrix in a text file and verifies the properties that the user selects. It verifies the completeness criteria, the balance criteria, the nonlinearity criteria, the propagation criteria and the XOR profile. The results are displayed in a textbox in detail.

P-box testing module allows loading the P-box as a text file. The application verifies at the beginning if the permutation

has an inverse and displays it in a text box. Then, the user can select which property of the permutation to test: if it has fixed point or if it is circular.

For the performance evaluation module, random files of 1KB, 10 KB, 100 KB, 1 MB, 10 MB, 50 MB, 100 MB, 500 MB and 1 GB were generated and used for all algorithms. To measure the throughput and speed, objects specific to .NET framework were used, such as "PerformanceCounter". After calculating these values, they are compared with the predefined threshold values.

5. EXPERIMENTAL EVALUATION

To demonstrate the functionalities provided by the framework, we used a system with Intel Core Dual CPU 1.86 GHz, 2 GB of RAM, Windows 8 32-bits and video card integrated. The purpose of this paper is to show which instruments can be used to evaluate a cryptographic algorithm, and to determine whether an algorithm is suitable to be used in secure communications or as a pseudo-random generator.

The experiments that were performed are presented in detail in the following sections.

5.1 Functional testing using test vectors

In order for an algorithm to be evaluated, a .dll file containing its functions must be provided as input to the framework, as mentioned in the Section 4.

The developer of the cryptographic algorithm has to provide a file, which contains test vectors, so that the evaluator can perform a functional testing before starting with the other evaluation mechanisms.

In the framework's layout, a separate box specifies the format necessary for the test vectors, as it can be seen in Figure 8. This ensures that the framework correctly interprets the obtained results.

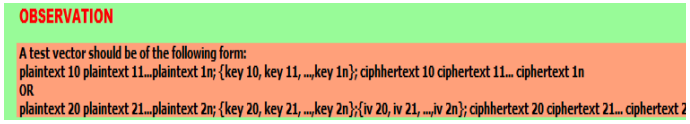

Fig. 8. The format for the test vectors.

For instance, consider the test vectors from Figure 9 for AES with 128-bit key.

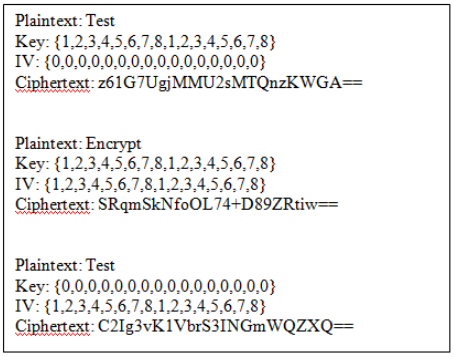

Fig. 9. Test vectors for AES-128 algorithm.

When loading the file with the values previously mentioned and in the format specified in the framework the results will be printed in a box of the framework and saved in a file (if the check box for this is selected). For the example above, the results are shown in Figure 10.

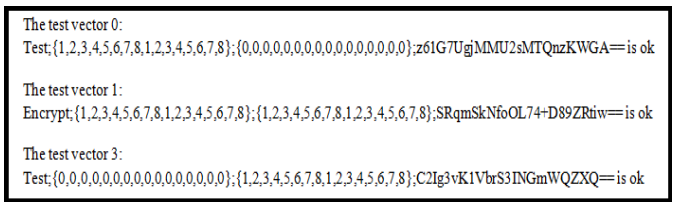

Fig. 10. Results for functional testing for AES-128.

In addition, if the values of the test vectors do not coincide, the framework specifies the incorrect value and which is the correct value. If the functional testing does not provide the correct results, the rest of the evaluation mechanisms still work, because the modules are independent of each other.

This functionality was tested on block cipher algorithms such as AES, DES, 3DES, TEA and Camellia and on stream ciphers such as RC4, Sosemanuk, LEX, and HC-128.

5.2 Statistical testing

As mentioned before, the framework ensures the possibility to apply the NIST statistical suite and the randomness tests used for AES candidates as described in previous sections.

In general, there are four values for the level of significance (α) that can be used: 0.05, 0.01, 0.005 and 0.001 and six different lengths for the sequences being tested: 128 KB, 256 KB, 512KB, 1MB, 2MB and 4MB. The minimum number of sequences/files, denoted with N, that have to be tested to obtain statistically relevant results depends on the value of α, as it can be seen in (1).

$$N \geq \frac{5}{(\alpha-(1-\alpha))} \quad (1)$$

Since the level of significance is 0.01 set as default in the framework implementation, the minimum number of files/ sequences necessary for testing with NIST statistical suite is 506. This means that when the user selects tests applied to AES candidates, the framework will generate automatically 510 files of 128 KB each, with the characteristics corresponding to selection made.

The interpretation of empirical results is made using two approaches: 1) examining the proportion of sequences that pass a statistical test, 2) analyzing the distribution of P-values to check uniformity.

Using the results obtained, calculate the proportion of sequences (files) that pass. For instance, if 510 binary sequences are tested and only 506 of them have the P-value greater than 0.01, then the proportion is 506/510=0.9921.

The range of acceptable proportions is given by the confidence interval, which is defined as in (2), where$\hat{p} = 1 - \alpha$ and m is the sequence size. For the example above, the confidence interval is 0.99 ± 0.00943. If the proportion is outside of this interval, then the data is non-random.

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}} (2)$$

The interval between 0 and 1 is divided into 10 sub-intervals and the P-values included in each sub-interval are counted and displayed in a chart, called a histogram.

The uniformity of the values is also verified by applying a χ^2 test on the P-values obtained for an arbitrary statistical test. Equation (3) shows how to compute this, where F_i represents the number of P-values in each sub-interval *i* and *s* is the sample size (in our case 510).

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10} (3)$$

The P-value is calculated with (4), using the incomplete Gamma function (igamc). If the P-value is greater than 0.0001, then the sequences are uniformly distributed.

$$P - value = igamc\left(9/2, \chi^2/2\right) (4)$$

The files which contain the results of the NIST statistical tests have the structure depicted in Figure 11.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------

------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE   PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  6  12   9  12   8   7   8  12  15  11  0.616305   0.9900     Frequency
  6  10   8  14  16  10  10   6   5  15  0.129620   0.9900     Cusum
  7   9   9  11  11  11   8  12  12  10  0.978072   0.9900     Serial
```

Fig. 11. Interpretation of the statistical tests results.

The first ten columns represent the frequency of P-values, and how they are distributed in the unit interval that has been divided into ten discrete bins. The next column is the P-value that is obtained after applying the chi-square test (this indicates the uniformity of the P-values for a specific statistical test). The column 12 indicates the proportion of the binary sequences that has passed (rate of success) and the last column indicates the name of the statistical test applied.

An example of a chart generated by the framework, when applying the frequency test is depicted in Figure 12.The frequency test was applied on 510 files of 128 KB generated by concatenating the ciphertexts obtained by using AES algorithm, with 128 bits key. The chart shows the uniformity of the P-values obtained. The interval between 0 and 1 is divided into 10 sub-intervals and the P-values that are between each sub-interval are counted and then displayed.

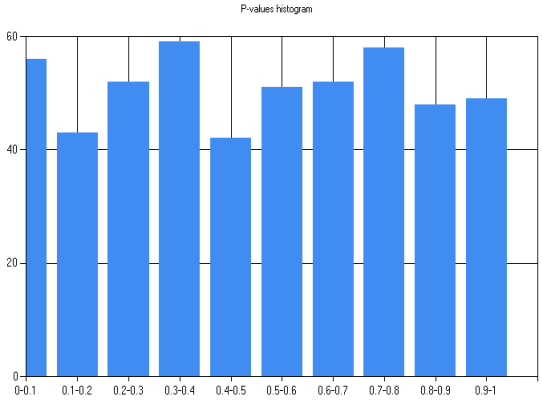

Fig. 12. Histogram of P-values for Frequency Test.

This module was used to test the output generated by algorithms such as AES, DES, 3DES, Camellia, TEA, LEX, Sosemanuk, HC-128 and RC4. In (Duta et al., 2014) we have applied studied the randomness properties of Camellia, TEA and LEX using the statistical module included in our generic framework. We have shown that TEA, Camellia and LEX are algorithms that pass with high score the statistical randomness testing demonstrating in this way their suitability to be random number generators.

5.3 Testing S-boxes

As mentioned in previous sections, analyzing the S-boxes of a cryptographic algorithm is also very important to determine whether or not the primitive can be used in applications to ensure security.

For this module, the framework asks the user to load the file containing the S-box (which has to be written in hexadecimal format in order to correctly interpret the results). As mentioned before, the user has the possibility to select which properties of the S-box he wants to test. The results are displayed in a separate window and can be written in a file, if the corresponding option is checked.

For instance, when considering the S-box of DES algorithm, shown in Figure 13, the framework will display the results presented in Figures 14, 15, 16 and 17.

```
0x9B 0x84 0xFE 0x22 0x29 0x40 0x47 0x56 0xC2 0x6C 0x9C 0xCD 0xF5 0x89 0x7D 0x2D
0x37 0x45 0x8A 0x8C 0xF2 0x57 0x5A 0x6A 0xAC 0x65 0x92 0x7A 0x02 0xA9 0x2A 0xBB
0x27 0xC8 0xF4 0x8F 0x13 0xA6 0xFA 0x39 0xC5 0x52 0xFD 0x82 0x91 0x3B 0x63 0x73
0x00 0xEA 0x0E 0xD1 0x1F 0x96 0x6B 0x99 0xBD 0x87 0x68 0x0A 0x43 0x30 0x61 0x20
0x2F 0x0D 0x32 0x21 0x2C 0x7E 0x7B 0xB1 0x35 0xCF 0x05 0x23 0xC1 0xA7 0xCA 0xC0
0x5E 0xAF 0x95 0x4B 0xA8 0x11 0x17 0xCC 0x9E 0xD6 0xCB 0x86 0xE7 0x5F 0xEF 0xD3
0xBC 0xE9 0xC9 0x55 0x51 0x97 0xBE 0x9D 0xD9 0x49 0xF7 0x10 0xE3 0x8E 0x9A 0x3C
0x88 0xC3 0x7F 0xAB 0x62 0x3E 0xB9 0x04 0x1D 0xF1 0x4E 0xDC 0xE6 0xDF 0x46 0x90
0x2B 0x2E 0xD0 0xA4 0x4F 0x15 0xC6 0x58 0xAA 0x06 0x83 0x42 0xEB 0x59 0x64 0x01
0x26 0x3A 0x5B 0xE0 0xD4 0x74 0xC7 0xDE 0xB6 0xEE 0x85 0x1A 0x33 0x3F 0x36 0xF8
0x79 0x0F 0x81 0x4A 0x76 0x6D 0x19 0xEC 0xE8 0x66 0x34 0x0B 0xDD 0x7C 0xDA 0x9F
0xF3 0xB5 0xD7 0x78 0xB3 0x8D 0xFF 0xDB 0xA5 0x3D 0xF9 0xD5 0xA1 0xE2 0xBF 0xF6
0x08 0xF0 0x48 0x16 0xB2 0x41 0xFC 0x24 0x14 0x31 0x77 0x50 0x09 0xED 0x72 0x8B
0xBA 0xE1 0x18 0x4C 0xAD 0xB0 0xB4 0x70 0x4D 0x80 0xFB 0xE4 0x1B 0x0C 0xA2 0x6E
0x12 0x5C 0xB7 0xCE 0x03 0x07 0x69 0x94 0x67 0x38 0xA3 0xA0 0x60 0x6F 0x54 0x93
0xD8 0xB8 0x44 0xC4 0x71 0x98 0xD2 0x5D 0x25 0xE5 0x53 0x1C 0x75 0x28 0x1E 0xAE
```

Fig. 13. DES S-box.

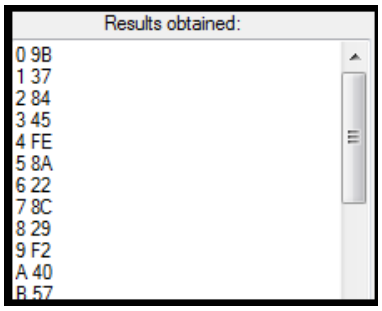

Fig. 14. Results for balance criteria for DES S-box.

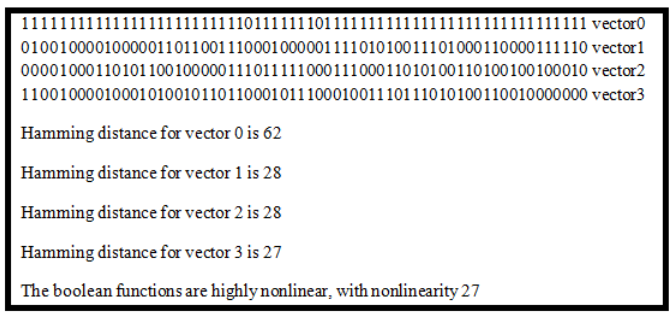

Fig. 15. Results for nonlinearity criteria for DES S-box.

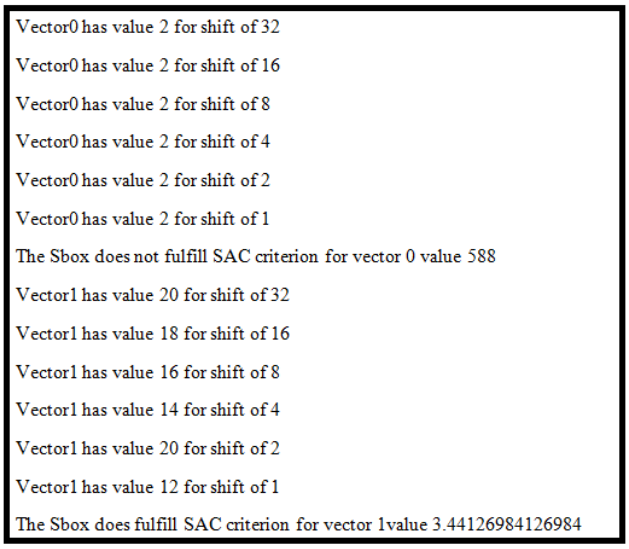

Fig. 16. A part of the results for propagation criteria for DES S-box.

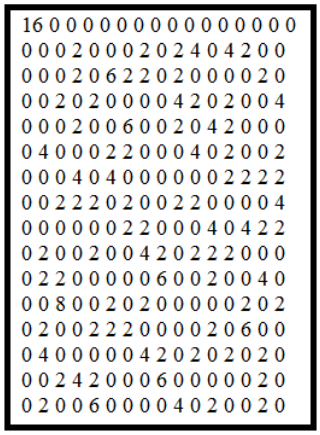

Fig. 17. An XOR profile generated automatically.

In addition, the S-boxes used for AES and Camellia were verified and we can conclude that they satisfy all the properties necessary for a good cryptographic S-box.

5.4 Testing P-boxes

As described in the framework architecture section, another functionality of the framework is to test the properties of the P-boxes used by cryptographic algorithms. P-boxes, as independent constructions do not provide too much security to the cryptographic algorithms, because the only property that they achieve is diffusion.

In combination with other mathematical transformations such substitutions, XOR-ing and other, they contribute to the security of the algorithm. In this paper, we present two properties of independent P-boxes.

First of all, having a permutation function, *f*, such as in (5), we want to test if the permutation has fixed points. If (6) is true, the permutation has fixed points, otherwise it doesnot have fixed points.

$$\Pi = \begin{pmatrix} x_i \\ f(x_i) \end{pmatrix}, i = 1..n; \quad (5)$$

$$x_i = f(x_i) \quad (6)$$

The second test verifies the fact that the permutation function, f, described in (1) is not circular. Considering the inverse permutation function from (7), called *g*, if (8) is true, then the permutation Π is circular, otherwise is not circular.

$$\Pi^{-1} = \begin{pmatrix} x_i \\ g(x_i) \end{pmatrix}, i = 1..n; \quad (7)$$

$$g(x_i) = f(x_i) \quad (8)$$

An example for testing our approach upon the P-boxes properties we have chosen the 32 bit P-box used by DES for shuffling the bits of a 32 bit half block. Therefore the results obtained by using the framework described in this paper are shown in Figure 18.

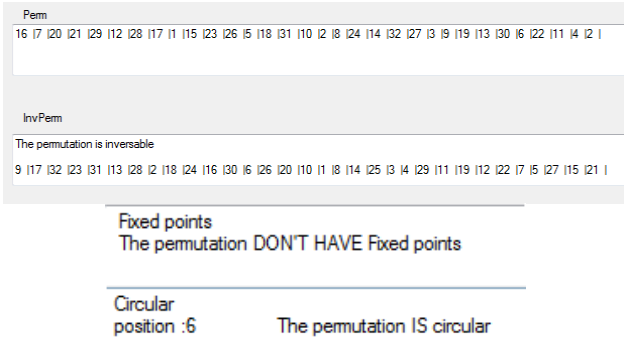

Fig. 18. Results for DES P-box- 32 bit half block.

Also, the permutation functions from DES, Camellia, and Blowfish have been tested. We can mention the fact that the permutation included in DES algorithm is not good, it does not satisfy the previously mentioned properties. This does not mean that the construction of the algorithm is not secure, because as we already said, the permutations ensure diffusion, but combined with S-boxes and other structures that ensure confusion, the algorithm can be cryptographically good.

5.5 Algorithm Performance Evaluation

Every cryptographic algorithm is created in order to fulfill security needs in a specific application. Sometimes, they are used in embedded devices and other times in software modules. When selecting the best cryptographic algorithm for these places, we need to take into consideration the speed and throughput provided by them.

The framework has a module entitled "Performance evaluation", which measures the throughput of the cryptographic algorithms that is being evaluated, when encrypting files of 1 KB, 100 KB, 500 KB, 1MB, 100MB, 500MB and 1 GB. The values obtained are compared with threshold values such as: throughput for AES (128, 192, 256 bits according to the key size) and throughput for RC4 (for stream ciphers in particular). Based on the comparison with these values, the framework returns the following results, as represented in Figure 19:

- The algorithm has a good throughput, but smaller than the one provided by AES/RC4;

- The algorithm ensures a high throughput, better than the one provided by AES/RC4;
- The algorithm ensures a small throughput, much smaller that the throughput of AES/RC4 algorithm.

In previous research, we have calculated the performance in terms of throughput, speed and clock cycles for AES, DES and 3DES, in order to establish which should be considered the standard threshold value used to compare other algorithms performance.

In (Duta et al., 2013), we calculated the performance in terms of throughput, speed and clock cycles for stream ciphers such as HC-128, RC4 and Sosemanuk, in order to establish which should be considered the standard threshold value.

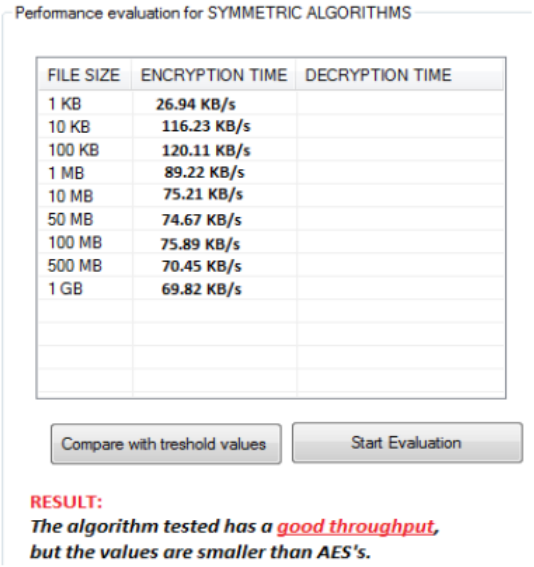

Fig. 19. Results from Performance Evaluation Module for Camellia algorithm.

We have chosen AES's throughput as the standard value for block ciphers and the RC4's throughput as the standard value for stream ciphers.

For instance: Camellia algorithm has a good throughput, but the values are smaller than AES's; TEA algorithm has a high throughput, the values are greater than AES's; LEX algorithm has a good throughput, but the values are smaller than RC4's.

6. CONCLUSIONS

The aim of this paper is to present a framework, which can be used to evaluate any symmetric encryption algorithm by taking into consideration different elements such as: properties of S-boxes, properties of P-boxes, test vectors for the algorithm, statistical properties of the generated ciphertext (applying randomness tests such as NIST and the tests used for the AES candidates), speed and throughput provided by the algorithm.

From the evaluation results, we can observe certain characteristics. First of all, we can ensure functional testing for the algorithms if test vectors are provided. This is a necessary functionality needed every time a new cryptographic algorithm is developed. We performed functional testing for block cipher algorithms such as AES, DES, 3DES, TEA and Camellia and for stream ciphers such as RC4, Sosemanuk, LEX, and HC-128.

Secondly, the properties of the S-boxes and P-boxes used in the construction of the algorithm can be verified to identify the existence of any vulnerability. For instance, the S-boxes used by AES and Camellia were verified as well as the permutation functions from DES, Camellia, and TEA.

Thirdly, the framework can be used to statistically test any cryptographic algorithm. To verify the correct functionality of this module, the output generated by algorithms such as AES, DES, 3DES, Camellia, TEA, LEX, Sosemanuk, HC-128 and RC4 were tested. As far as we know, a generic framework to evaluate the randomness properties of any cryptographic algorithm (stream cipher or block cipher) has not been publicly presented or described. Also, we are the first to publish in (Duta et al., 2014), the results obtained for TEA cipher regarding its randomness properties.

Fourthly, the performance of the evaluated algorithms in terms of throughput is compared with threshold values obtained for standardized implementations of cryptographic primitives. For instance, Camellia algorithm has a good throughput, but is smaller than AES's. Also, stream ciphers such as Sosemanuk and HC-128 were tested and the performance evaluation module showed that they have a good throughput, but smaller than RC4's.

Our future work involves an improvement of the framework such that it will allow evaluating algorithms based on other criteria such as memory usage, attack scenarios and hardware implementation resources. Our goal is to be able to decide based on an evaluation framework which encryption algorithms to use for different types of applications in a way that consumes less energy, but still ensure high speed.

REFERENCES

Angraini, N., Susanti, B.H. and Magfirawaty (2013). Analysis of the Use of Whirlpool's S-box, S1 and S2 SEED's S-box in AES Algorithm with SAC Test. *Information Systems International Conference*, pp. 56-76.

Arora, R. and Sharma, S. (2012). Performance Analysis of Cryptography Algorithms. *International Journal of Computer Applications*, Vol. 48, No. 21, pp. 35-39.

Brown, L. and Seberry, J. (1990). On the Design of Permutation P in DES Type Cryptosystems. *Advances in Cryptology: Proceedings of EUROCRYPT '89*, pp. 696-705.

Chen, H., Feng, D.G. and Fan, L.M. (2009). A New Statistical Test on Block Ciphers. *Chinese Journal of Computers*, pp. 595-601.

Cook, D.L., Yung, M. and Keromytis, A.D. (2009). Elastic Block Ciphers in Practice: Constructions and Modes of Encryption. *3rd European Conference on Computer Network Defense*, pp. 69-91.

Dhawan, P. (2002). Performance Comparison: Security Design Choices. *Microsoft Developer Network*,

http://msdn2.microsoft.com/en-us/library/ms978415.aspx (Last Accessed: September 2014).

Duta, C.,Mocanu, B.C.,Vladescu, F.A. and Gheorghe, L. (2013). Performance Analysis of Stream Cipher Algorithms. *6th International Conference on Security for Information Technology and Communications*, pp. 112-128.

Duta, C., Mocanu, B.C, Vladescu, F.A. and Gheorghe, L. (2014). Randomness Evaluation Framework of Cryptographic Algorithms. *International Journal on Cryptography and Information Security*, Vol. 4, No.1, pp. 31-49.

Gustafson, H. et al. (1994). A computer package for measuring the strength of encryption algorithms.*Computers and Security*, Vol. 12, pp. 687-697.

Heys, H. M. (2004). A Tutorial on Linear and Differential Cryptanalysis. Technical report. *Electrical and Computer Engineering*, University of Newfoundland, Canada.

Kam, J. and Davida, G. (1979). Structured design of substitution-permutations networks. *IEEE Transactions on Computers*, pp. 747-753.

L'Ecuyer, P. and Simard, R. (2007).TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, Vol. 33, No. 4.

Marsaglia, G. (2003). The diehard test suite, available at http://i.cs.hku.hk/~diehard/(Last Accessed: September 2014).

Ramesh, G. and Umarani, R. (2012). Performance Analysis of Most Common Encryption Algorithms on Different Web Browsers. *International Journal of Information Technology and Computer Science*, Vol. 4, No. 12, pp. 60-66.

Rukhin, A., et al. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST Special Publication 800–22. *National Institute of Standards and Technology*.

Saarinen, M.O. (2011). Cryptographic Analysis of All 4x4-Bit S-boxes, https://eprint.iacr.org/2011/218.pdf (Last Accessed: September 2014).

Shah, J. and Saxena, V. (2011). Performance Study on Image Encryption Schemes. .*International Journal of Computer Science*, Vol. 9, Issue 4, No. 1, pp. 349-355.

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal 28*, Vol.4, pp. 656-715.

Singh, G., Singla, A.K.. and Sandha, K.S. (2011). Performance Evaluation of Symmetric Cryptography Algorithms. *International Journal of Electronics and Communication Technology*, Vol. 2, Issue 3, pp. 141-146.

Soto, J. (1999). Radomness Testing of AES Candidate Algorithms, available at http://csrc.nist.gov/publications/nistir/ir6390.pdf (Last Accessed: September 2014).

Stoianov, N. (2010). One software tool for testing square s-boxes, http://arxiv.org/ftp/arxiv/papers/1009/1009.2476.pdf(Last Accessed: September 2014).

Tamimi, A. (2005). A Performance Comparison of Data Encryption Algorithms. *Information and Communication Technologies, ICICT 2005*, pp. 84-89.

. Walker, J. (2008). ENT-A pseudorandom number sequence test program, available at http://www.fourmilab.ch/random/(Last Accessed: September 2014).

Webster, A.F. and Tavares, S.E. (1986). On the design of S-boxes. *Advances in Cryptology (CRYPTO'85), Lecture Notes in Computer Science No. 218*, Springer, pp. 523-534.