Secure True Random Number Generator in Wireless Network (Ad hoc)

Seyed Masoud Alavi Abhari

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
July 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Zeki Bayram
2. Assoc. Prof. Dr. Alexander Chefranov
3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

During last decade, Wireless LAN (WLAN) has been very important and developed area of technology and science. Nowadays a level of security that can exceed the security of a WLAN is provided by using security protocols. Not only the security of WLAN can increase by cryptography algorithms their own, but also will be more powerful if True Random Numbers are being used in Cryptography algorithms. Today it is hardly possible to disregard the vital role of random numbers in Cryptography consequently security of wireless Networks in case of key exchange algorithms or nonce to convince the other part is trusted.

The aim of thesis is generating secure true random numbers in context of Ad hoc WLAN. To this aim, Diffusion RNG_{Ligth} technique, which is modification of Scatter RNG_{Ligth} technique that produced true random numbers using sensory reading on Wireless Sensor Network (WSN) [1], is implemented. These modifications consist of adding and changing some parts of the secure frameworks and using AES and Triple-DES instead of DES in encryption/decryption and CMAC. In addition, produced random numbers using Diffusion RNG_{Ligth} are evaluated by NIST statistical test. The results show that p-values of Diffusion RNG_{Ligth} using AES 60% have improved in comparison with Diffusion RNG_{Ligth} using Triple-DES, also p-values of Diffusion RNG_{Ligth} using Triple-DES 60% have improved in comparison with Scatter RNG_{Ligth}.

Keywords: Ad hoc, Wireless Local Area Network (WLAN), Network Security, True Random Number Generator (TRNG), Multicasting, Diffusion RNG_{Ligth}

ÖZ

Son on yılda, Kablosuz LAN (WLAN) teknoloji ve bilim çok önemli ve gelişmiş bir alan olmuştur. Günümüzde WLAN güvenlik aşabilir güvenlik düzeyi güvenlik protokolleri kullanılarak sağlanır.WLAN güvenlik Sadece kendi şifreleme algoritmaları ile artırabilir, ama gerçek Rasgele Sayılar Kriptografi algoritmaları kullanılıyor ise de daha güçlü olacaktır. WLAN güvenlik Sadece kendi şifreleme algoritmaları ile artırabilir, ama gerçek Rasgele Sayılar Kriptografi algoritmaları kullanılıyor ise de daha güçlü olacaktır. Bugün dolayısıyla kablosuz ağları güvenlik anahtar değişimi algoritmaları veya nonce diğer kısmı ikna etmek durumunda güvenilir Kriptografi rastgele sayı hayati bir rol göz ardı etmek pek mümkün değildir. Tezin amacı, Geçici WLAN bağlamında güvenli gerçek rasgele sayılar oluşturuyor. Bu amaçla, Difüzyon RNGLigth tekniği için, hangi Kablosuz Algılayıcı Ağ (KAA) [1], uygulanan duyusal okuma kullanarak gerçek rasgele sayılar üretti Dağılım RNGLigth teknik değişiklik olduğunu. Bu değişiklikler ekleme ve değiştirme güvenli çerçeveler bazı parça ve şifreleme / şifre çözme ve CMAC yerine DES AES ve Triple-DES kullanarak oluşur. Buna ek olarak, Difüzyon RNGLigth kullanarak rasgele sayılar NIST istatistik testi ile değerlendirilir üretti. Sonuçlar AES 60% kullanarak Difüzyon RNGLigth p-değerleri de Triple-DES, Triple-DES 60% kullanarak Difüzyon RNGLigth p-değerleri Dağılım RNGLigth ile karşılaştırıldığında düzeldi kullanarak Difüzyon RNGLigth ile karşılaştırıldığında daha iyi olduğunu göstermektedir.

Anahtarkelimeler: Ad hoc, Kablosuz Yerel Alan Ağı (WLAN), Ağ Güvenliği, Gerçek Random Number Generator (TRNG), Multicasting, Difüzyon $\text{RNG}_{\text{Ligth}}$

Dedicated to my family

ACKNOWLEDGMENTS

First and foremost, I offer my sincerest gratitude to my supervisor, dear Dr. Alexander Chefranov, who has supported me throughout my thesis with his patience and knowledge. I attribute the level of my Master degree to his encouragement and effort and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier supervisor.

I would like to express my deepest gratitude and appreciation to Dr. Zeki Bayram and Dr. Gürcü Öz for the useful comments, remarks through the learning process of my Master degree. Last, but not least, I am greatly indebted to my lovely parents for their invaluable supports. I also owe special gratitude to my dear friend Maryam Farajzadeh for her constant encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
DES	Data Encryption Standard
BB	Buffer Block
CMAC	Cipher based Message Authentication Code
PRNG	Pseudo Random Number Generator
TRNG	True Random Number Generator
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
RB	Register Block
LC	Local Clock

Chapter 1

INTRODUCTION

Over the past few years, wireless communications has been fast growing with many devices like laptops, PDAs, and Pocket PCs. It starts to change many aspect of our life, some of these are our approach to business interaction, emails, private communication for mobile or E-commerce; all of these require wireless technology. However, studies indicate that the growth of wireless networks is restricted by their perceived insecurity.
Today the essential need of Wireless Local Area Network (WLAN) security is noticeable. However needs of protecting of such networks are increased while encryption algorithms have a fundamental role in information systems security [2].

Encryption methods and cryptography algorithms require a source of random data, even some symmetric ciphers (where the secret is shared), to generate new either private or public key pairs, for nonce, for session keys, for padding, or for any other reasons [3, 4, 5]. Actually, many Methods for Random Number Generation (RNG) have been invented, physical methods, Computational methods or Generation from a probability distribution.

Random numbers are divided in two categories. Some produced by Pseudo Random Numbers Generators (PRNG) base on a formula, they are very seldom truly random, and they are generally pseudo random also guessable by cryptanalysis. In other words, they appear random, but are not random absolutely. On the other hand, there are also some others called True Random Numbers Generator (TRNG), which produces random numbers measured with some physical phenomenon that is expected to be random.

Moreover, true random numbers are unpredictable and aperiodic. Not only the security of WLAN can increase by cryptography algorithms their own, but also will be more powerful if true random numbers are being used in Cryptography algorithms. Thus, the methods related to secure TRNGs is obviously needed. In addition, a technique that implemented earlier is named Scatter $\text{RNG}_{\text{light}}$. It is about distributed true random number generator in Wireless Sensor Network (WSN). In this case, random numbers derived from the distributed sensor readings are collected using secure frameworks. Accordingly, DES algorithm and SHA-1 are used for providing the data integrity and authenticity for security of the frameworks. After collecting the random numbers, they are authenticated, and then random numbers received from other sensors are combined in the requester side to generate a random number [1].

The main purpose of this study is generating secure true random numbers by using laptops in an Ad hoc WLAN due to Diffusion $\text{RNG}_{\text{Light}}$, and then evaluating randomness of them by NIST statistical tests. To achieve these aims an Ad hoc WLAN is established to prepare a full graph of communication between all available laptops, then the laptop which needs a random number, multicast its request to other laptops, thus the others answer the request by replying a message in form of a protected frame. Finally, the demandant laptop after collecting the authenticated replies, generate a true random number.

Moreover, for the implementation three modifications are applied on Scatter RNG_{Light}; Firstly, AES and Triple-DES mentioned in [6] are used in scramble function and for messages encryption instead of DES used in Scatter RNG_{Light}. Secondly, the IP address, which is used in this study, contains 32 bits, while 12 bits is used for IP address in Scatter RNG_{Light}. Finally, 4bits are reserved in message to declare which type is used for encryption of a frame. Furthermore, randomness of the outputs of Diffusion RNG_{Light} will be tested. A reliable test to evaluate the randomness of these random numbers should cover most of statistical tests. In addition, there are some techniques to test randomness for a binary sequence. One of them called NIST test suite provided in National Institute of Standards Technique [7]. The quality of Diffusion RNG_{light} in terms of the randomness of the produced number sequences is assessed and the results represent that the number of P-values in Diffusion RNG_{Light} using Triple-DES is 60% greater than the P-values of Scatter RNG_{Light}, while Diffusion RNG_{Light} technique using AES achieved 60% greater P-values than Diffusion RNG_{Light} using Triple-DES.

This study divided in four chapters, which, chapter 2 is about related works, and definitions, which will be used in this study. Chapter 3 describes Diffusion RNG_{Light} development in WLAN, while chapter 4 provides discussion about randomness quality of the generated random numbers. After finalizing this study in Conclusion, in Appendix running procedure of the program and the whole view of developed codes are presented.

Chapter 2

DEFINITIONS AND RELATED WORKS

In this chapter, Ad-hoc WLAN is debated in section 2.1.1 to prepare a mesh topology that all laptops in the network are capable of communicating with each other.

Then some issues about cryptography algorithms definitions are explained; Section 2.1.2, the Data Encryption Standard (DES), section 2.1.3 the Advances Encryption Standard (AES) and section 2.1.4, triple DES. These algorithms are used to modify the secure framework for transferring generated random numbers. In addition, one-way function Sha-1 using for message digest and data integrity will be discussed in section 2.1.5. Moreover, section 2.1.6 is about CMAC, which stands for Cipher based Message Authentication Code, is used in random number generation module for assurance authenticity and integrity. In section 2.1.7 true and pseudo random number generators are discussed. However, apart from how random numbers is produced, the quality of generated random numbers in purpose of randomness is a matter of debate. Thus, NIST statistical test (provided by National Institute of Standard and Technology); assesses the random numbers to be sure that they are truly random enough; will be discussed in section 2.2.

Section 2.3 focus on wide variety of parameter that is used and provided based on various processes to generate random number and some related works will be brought up. Moreover, in section 2.4, the needs for TRNG and secure transferring of random numbers in context of Ad hoc Wireless Network are considered and the problem is defined. Actually all these requirements are determined in Diffusion RNG_{Light} technique.

2.1 Definitions

Security is a major critical field in the wireless Local Area Network (WLAN), because it is more flexible than LAN for connection of any laptops, so any cracker and cryptanalysis can connect with a little trouble. For this reason and for increasing Intrusion Detection to prevent cracker and cryptanalysis sabotages, User Authentication, reliability of access control, Data Integrity and confidentiality and so on, security requires for any wireless connection such WLAN. However, Network Security & Cryptography is a concept to protect network and data transition over WLAN.

During the security review, encounter to WEP and WPA is inevitable, thus, to understand these concepts is preferred to be familiar with Wireless security, add-hock networks and wireless sensor networks.

2.1.1 Ad hoc Networks -Wireless Security

Actually there exist many ways to establish a wireless networks these networks introducing as followed.

According to the similarity of characteristic between mesh topology and Ad hoc in comparison with WSN, which is implemented in the [1], it was decided to use Ad hoc in WLAN. In this type one Laptop (node) prepares a context of WLAN used Mesh topology such a full graph illustrated in Figure1. Therefore, the other nodes will be able to connect to this network.

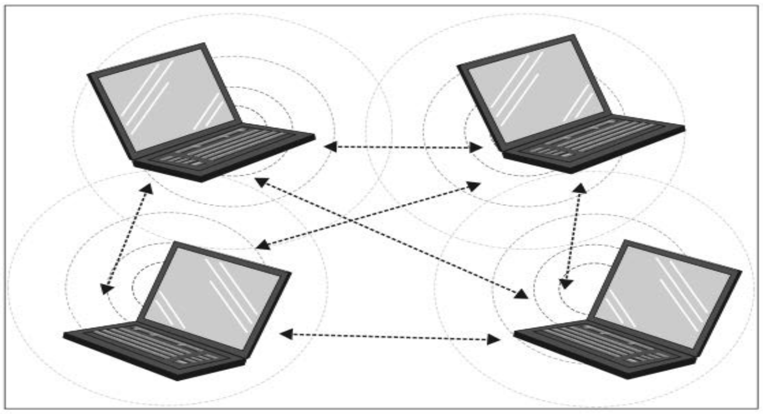


Figure 1: Ad hoc Network

In ad hoc networks, the communicating nodes do not necessarily rely on a fixed structure or formation, which needs to define and consider the necessary security architecture they apply. In addition, as ad hoc networks are often constructed for particular environments and expected to operate with full availability even in difficult conditions, security solutions applied in more networks that are traditional may not be adequate for protecting them. However, Wireless security is the prevention of unauthorized access or damage to computers using wireless networks. Wired Equivalent Privacy or WEP and Wi-Fi Protected Access, which is called (WPA), are the most general kind of wireless security. WEP is a weak security standard. A common laptop by help of an available software tools usually can brock the passwords in a few minutes (Figure1).

According to [16] WEP is an old IEEE 802.11 standard from 1999, which was abolished in 2003 by WPA. Moreover, WPA or Wi-Fi Protected Access was a safer replacement to improve security over WEP. While WPA2 is the current and most popular standard; some hardware cannot support WPA2 without firmware upgrade or replacement.WPA2 uses an encryption method encrypting the network with a 256-bit key; because of the longer key length, security of WPA2 has been improved against WEP.

Nowadays there exist many different methods and algorithms for retaining WLAN security, which rely strongly on cryptographic techniques. In other words, cryptography algorithm and techniques divided into some parts, Asymmetric and Symmetric algorithms (such AES, DES and Triple DES), one-way functions (i.e. SHA-1 and MD5 called Hash function) comes to provides security in WLAN. In the following these issues will be discuss in details.

2.1.2 Data Encryption Standard (DES)

DES is one of the most widely used encryption algorithm is based on the Data Encryption Standard adopted in 1977 by NIST [6]. The input data in DES encryption Algorithms are encrypted in 64-bit blocks using 56-bit key. The procedure transforms 64-bit input into a 64-bit output after passing a series of step using eight substitution tables and S-boxes in each iteration, with the similar key in both encryption or decryption process. However, there are two inputs to the encryption function or for the reverse of it, the plain text (the cipher text) and a 56-bit key as shown in Figure 2.

In addition, there are two important methods of cryptanalysis: Differential cryptanalysis and linear cryptanalysis. DES has been illustrating that is qualified and highly resistant to these kind of attack. DES considered avalanche effect, In other words a small change in one bit of the plaintext or even in one bit of key should produce a significant change in cipher text. Although multiple symmetric ciphers have been expanded and improved since DES was introduced, and although its aim is to be replaced by the Advanced Encryption Standard (AES), DES stays the most serious and principal such algorithm [6].

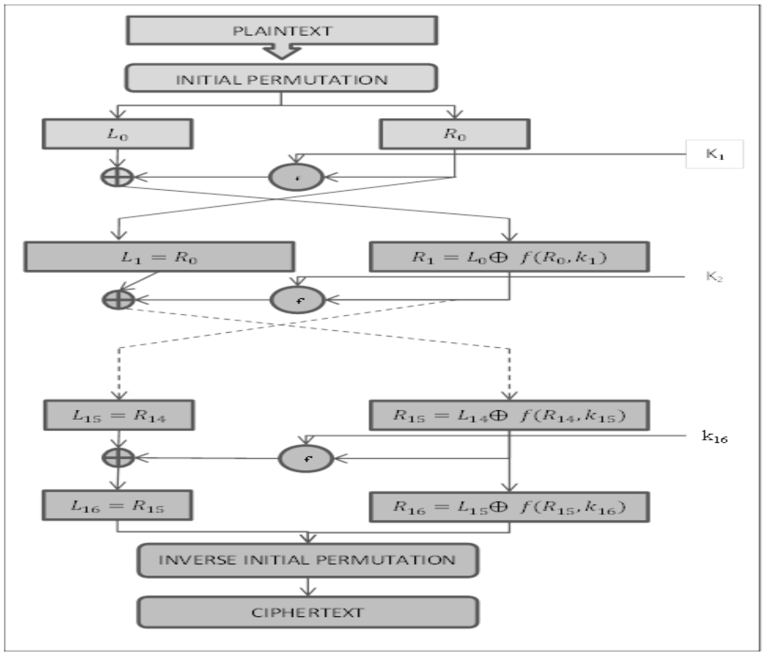

PLAINTEXT
INITIAL PERMUTATION
L_0
R_0
K_1
$L_1 = R_0$
$R_1 = L_0 \oplus f(R_0, k_1)$
K_2
$L_{15} = R_{14}$
$R_{15} = L_{14} \oplus f(R_{14}, k_{15})$
k_{16}
$L_{16} = R_{15}$
$R_{16} = L_{15} \oplus f(R_{15}, k_{16})$
INVERSE INITIAL PERMUTATION
CIPHERTEXT

Figure 2: Data Encryption Standard (DES) [6]

2.1.3 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is symmetric block cipher was published by the National Institute of Standards and Technology (NIST) in 2001 [6]. It is intended to exchange as the approved standard for a vast majority of applications instead of DES. AES uses a 128-bit block size and a key size of 128, 192, or 256 bits, Figure 3 illustrates AES. In comparison with DES, instead of using a Feistel structure [6], each full round of AES consists of four separate functions: byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key.

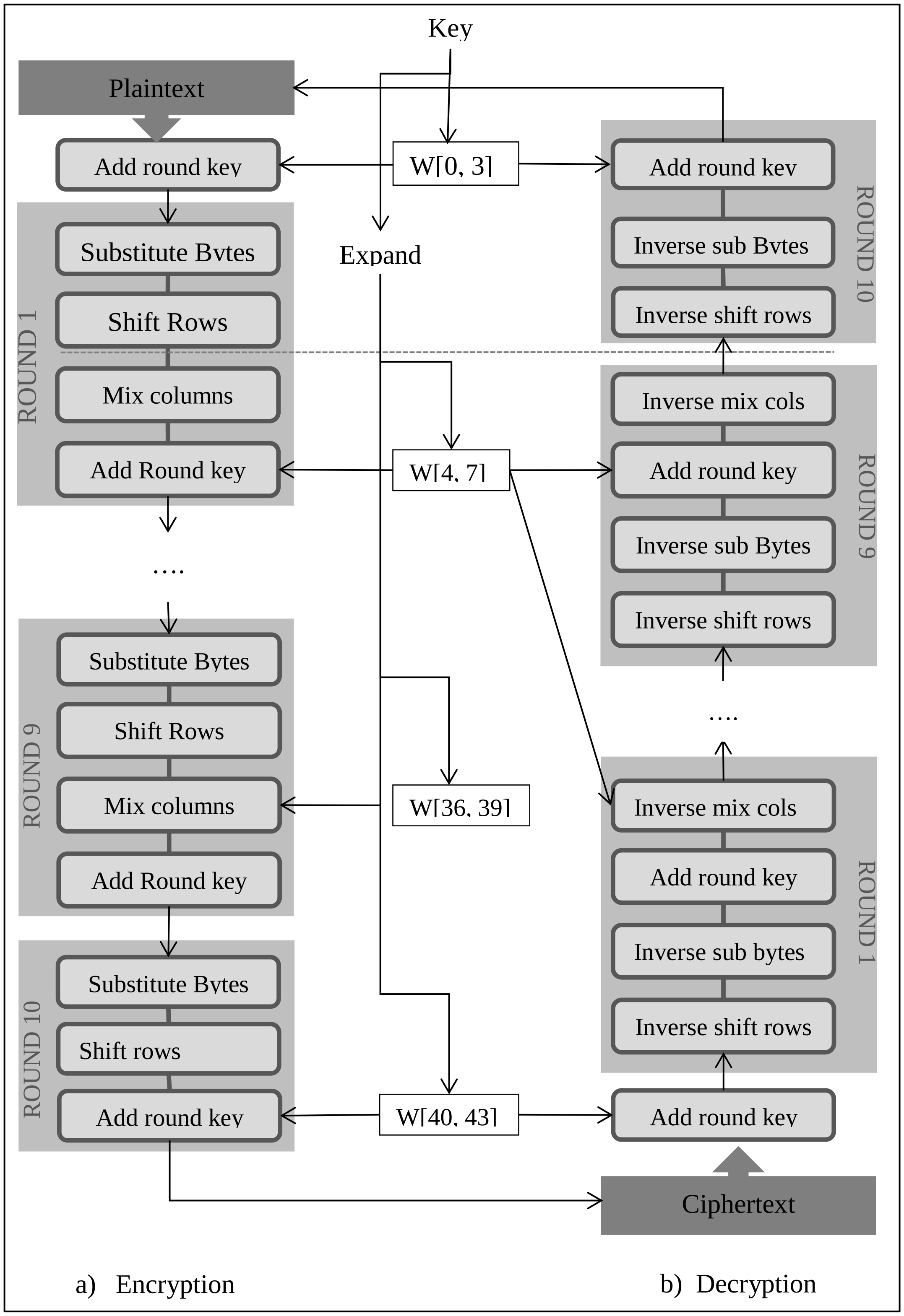


Figure 3: Advanced Encryption Standard (AES) [6]

2.1.4 Triple DES

In Triple DES algorithm [6] as its name suggests, the encryption algorithm is used multiple times, the structure of Triple DES Illustrated in Figure 4; thus, the plaintext is converted to the cipher text then the cipher text is used again as an input for the encryption algorithm. It is clear that the process of Triple DES continues in three steps. Actually, each step makes use of DES algorithm.

Triple DES needs two or three keys. Obviously, by using three different keys in three different step of encryption Triple DES counter to the “meet in the middle” attack.
One of the most important features of this algorithm is a mode of operation. There are five modes of operation that can be used with DES and AES, which are symmetric block ciphers: Electronic Code Book Mode, Cipher Block Chaining (CBC) mode, Cipher Feed Back (CFB) mode, Output Feed Back (OFB) mode and Counter mode.

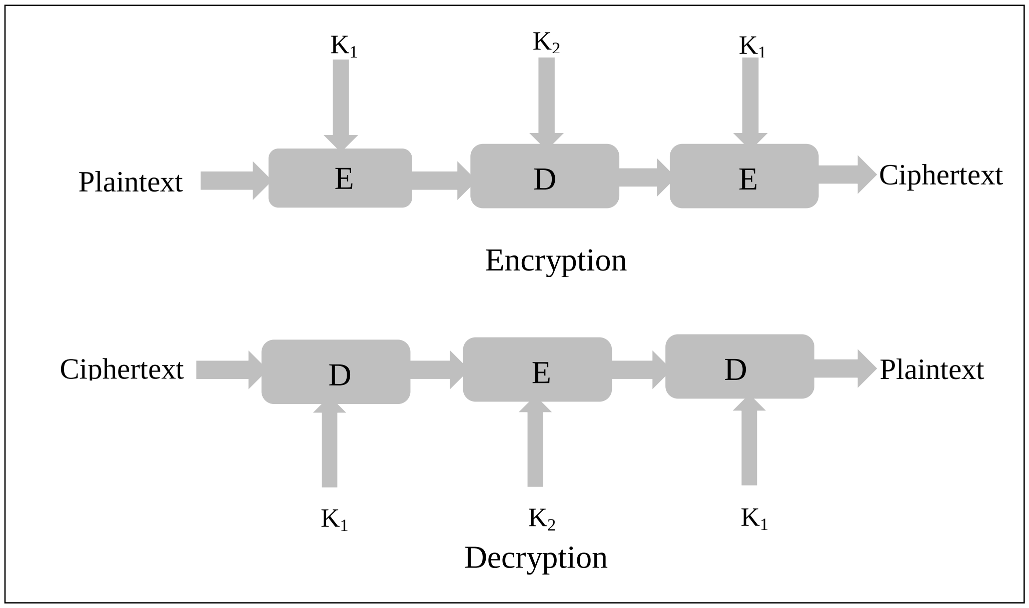


Figure 4: Illustration of Triple-DES [6]

2.1.5 SHA-1

SHA-1 is cryptographic hash function. SHA stands for secure hash algorithm have been most widely used during last years [6]. SHA-1 is called secure because it is computationally impossible to find two different messages which that generate similar message digest. Sha-1 accepts a message of any length less than 2^{64} bits as an input, then produces 160-bit output is called a message digest. Usually the message digest is used for signature verification, authentication.

In the study Sha-1 is applied in secure frameworks to assure that the receiving and sending frames are authenticated and the data has not changed, thus not only the security but also the integrity of data is provided by sha-1. In addition, it is used in process of generating true random number to make a secure fixed number of bits for buffer block will discuss later.

2.1.6 Message Authentication Code

Message authentication Code (MAC) is used to authenticate a message and providing data integrity [6]. In other words, it is used to verify the integrity of a message to assure that receiving data are not altered during transferring data and there is no modification or deletion and insertion so that it is exactly as the sending data. However, the sender should be authenticated and valid. MAC requires a secret key K and a variable length message M to generate a fixed length output T to detect both accidental and intentional modifications of the data.

$$\mathrm{T} = \mathrm{MAC}\,(\mathrm{K}, \mathrm{M})$$

An approach of forming a MAC to outputs a fixed length of data for any arbitrary length input data is using symmetric block cipher. Cipher-based Message Authentication Code (CMAC) is defined by NIST is keyed hash function duo to a symmetric key block cipher such as AES.

2.1.7 Salt

Salt is a sequence of zero and ones defined due to the time for performing a permutation during encryption process. The purpose of using salt in database is for distinguishing two similar passwords that set by two users. Actually, salts provide security as increase the length of passwords that make it so hard for attacker to guess [6].

2.1.8 Nonce

Nonce is an arbitrary number defined to prevent reply attack. During a communication between two computers, it increases one time for every transmitting message. Therefore, if a computer receives a message two times it will understand that the message is received two times and finally it will reject the second message to prevent replay attack [6].

2.1.9 True and Pseudo-Random Number Generators

In the view of random number generating, there are two approaches, computational and physical.

Pseudo Random Number Generators (PRNGs) treat as functions or formulas that produce random numbers based on the initial state, which is called the seed:

$$s_0 = seed, \;\; s_{i+1} = f(s_i), \qquad i = 0,1,2,3,\cdots$$

Moreover selecting a good seed for a given algorithm is often a matter of debate. The PRNG will repeat at some point based on the finite state, and the period of a random numbers is considerable in security algorithms. It is completely clear that starting a PRNG with the same seed allows repeatable random sequences. Apart from the negative impact that the periodicity of PRNG has on the cryptography, it is very useful for debugging among other things by following the generation pattern. While, pseudo random numbers are repeated after a period, they are guessable so that it is not reliable to use for cryptography applications [8].

True Random Number Generators (TRNG) measures some physical phenomenon to produce a value as random numbers. Moreover, it is not possible to reproduce the generated sequence of true random numbers. As they are aperiodic and it is difficult to guess the next generated number, they are good choices for cryptography usage. Some examples of TRNG are semiconductor noise, clock jitter in digital circuits [8].

However many aspects of cryptography require true random numbers. For instance in producing keys of cryptographic algorithms, or initial values that are used in many cipher modes such as Counter, Cipher Block Chaining (CBC), and Cipher Feed Back (CFB) mode. In addition, TRNG is used for nonce, which is an arbitrary large number use only ones in cryptographic communications for preventing replay attacks. As well as, TRNG takes in to account for the purpose of One-time pads and Salts.

2.2 Introduction to NIST Statistical Test

NIST tests includes fifteen different statistical tests [7] to determine whether a TRNG is appropriate for a particular cryptographic usage or not while in the view of cryptographic application we need to meet stronger requirements than for other usages. The outputs of TRNG need to be unpredictable and the generated random numbers should be in an acceptable quality level of randomness. According to NIST test, it is easily understood that the result of TRNG is reliable or not.

Nowadays demanding of Random numbers used in cryptographic algorithm is increased. Some crypto algorithms need a key, which must be generated as a random number. Suitable generators used in cryptographic algorithms should meet more insurance and requirement over time and their random numbers should be unpredictable. As well as many cryptographic protocols use true random numbers and pseudo random numbers as an input. This section is going to explain a set of statistical test checking the randomness.

It is proved by the National Institute of Standards and Technology (NIST) that these tests are useful enough for detecting the deviation and randomness of a bit sequence [7]. A NIST test also is able to recognize and attend that an obvious deviation from randomness caused by a week design generator or anomalies occurred in the tested binary sequence.

2.2.1 General Discussion

Actually, TRNG and PRNG are two basic generators. TRNG and PRNG are both generate a sequence of zeroes and ones for cryptographic algorithms. This sequence as a stream could be assessed into subsequence or blocks contain zero s and ones in random.

2.2.2 How to Apply the Tests

There are many statistical tests to apply to a sequence for comparing the evaluation of the sequence whether are generated truly random or not. It is clear that the randomness is a probabilistic feature; which means the attribute of a random sequence can be presented and specified in probability field. There are too many statistical tests, to assess the presentment and absence of a template, in that recognizing the randomness of the sequence. Existing too many tests to judge that is the sequence random or not, caused to there is not any particular complete set of tests. So that, the results of statistical tests should be presented by consider of some computational accuracy and error, to achieve a correct conclusion for a special generator. All statistical tests are defined to test a null hypothesis (H_0). In content of NIST test, the sequence, which has been tested, called as the tested null hypothesis. Related to the null hypothesis, when a sequence is not random, hypothesis will be alternative hypothesis (H_a). Therefor in each tests decision about to accept or reject null hypothesis makes based on the produced sequence.

The randomness statistic should be defined and applied to specify the acceptance or rejection of the null hypothesis. A test statistic value is derived from a computation on the tested sequence, during a statistic test. If the tested statistic value is not greater (in the other hand, equal or less) than the critical value, the null hypothesis for randomness will not be rejected (in some situation will be accepted). Otherwise, if the critical value becomes less than the test statistic value the null (the randomness) hypothesis for randomness will be rejected. Statistical hypothesis testing works because the critical value and the reference distribution are related. If the computed test statistic value becomes greater than the critical value, then according to the statistical hypothesis testing, inherently, the low probability event should not happen. Therefore, in situation that the critical value is equal or less than the computed test statistic, the conclusion will be determined in terms of the assumption of randomness is doubtful or beaten. Statistical hypothesis testing in this position yields the followed conclusions: rejection H_0 and concurrently acceptance of H_a.

Statistical hypothesis testing is a procedure of conclusion-generation procedure that has two possible consequences, acceptance of H_0 (the sequence is random) or acceptance of H_a (the sequence is non-random). Table1 is related to an unknown (true) position of the sequence to the conclusion inferred from the applied testing procedure.

Table 1: Conclusion Derived from a Usage of the Testing Procedure

Conclusion	True Situation	
	Data is random (H0 is true)	Data is not random (Ha is true)
To accept H_0 (rejection of H_a)	No error	Type II error
To accept H_a (rejection of H_0)	Type I error	No error

Firstly, the conclusion will be named a Type I error, when the sequence, in fact, be random and then a conclusion for rejection the null hypothesis occurs a small percentage of the time. Secondly, if the sequence is, non-random, then a conclusion deducted to reject the alternative hypothesis, it is named a Type II error. Thirdly, if the sequences are really non-random or random, then the conclusions reject or accept H_0, the inference will be correct.

The probability related to a Type I error is named the level of significance of the test. The probability is presented by α . α In the statistical test is a probability that the sequence is not random, although it is actually random. In other words, it produced by a good generator even though the sequence represented as non-random. In cryptography usage, the value of α often is around 0.01. In addition, it is clear that the Type II error has probability. The value β is the probability of a Type II error. In the statistical test β is the probability that the sequence is random, although it is non-random. On the other hand although the sequence represented as random sequence, it produced by a bad generator. Possible values for β are infinite, in that there are many ways that the sequence is non-random, and every way have a different β. While so many ways exists for producing non-random number, the conclusion of β is much more difficult than the conclusion of α . One of the basic aims of the statistical tests is reducing β value as much as possible.α and β are rely on each other. For a tested sequence with size of n in a way that each of α and β defined, the new value is specified. Scientists normally use a value for the level of significance α and a normal size n and an essential point for a given test, which will produce the minimum the probability of a Type II error (β).

Every statistical test depends on a statistic value. The critical value and the test statistic value is presented by t and S. Therefore the probability of Type I error is:

$$p(S > t \parallel H_0\, is\ true) = p(reject\ H_0 true)$$

Moreover, the probability of the Type II error is:

$$p(S \leq t \parallel H_0\, is\ false) = p(accept\ H_0 is\ false)$$

The other most important value is P-value. The P-value, which defines the resistance of the demonstration antagonistic the null hypothesis, is computed by the use of test statistic. If a P-value becomes equal to one, then the sequence will be completely random. Otherwise, if a P-value becomes zero, then the sequence will be perfectly non-random.

It is clear that the probability of the Type I error presented by α. In each test α can own different or arbitrary values. Normally the interval of α in the statistical test is considered as $[0.001,\ 0.01]$.

We can make a decision due to the relevance of P-value and α, which are, firstly the null hypothesis is rejected when P-value becomes less than α $(P - value < \alpha)$ in other words the sequence seems to be non-random and secondly the null hypothesis is accepted when P-value becomes equal or greater thanα $(P - value \geq \alpha)$, in other words the sequence seems to be random. By considering the previous definitions it is possible to inference the two followed conclusions.

- By assuming α as 0.001, it is expected that if a sequence is random, then the sequence should be rejected in 1000 sequences. If a P-value were equal or greater than 0.001, then the sequence with 99.9% confidentiality would be random. In addition, if a P-value were less than 0.001, then the sequence with 99.9% confidentiality would be non-random.

- By assuming α as 0.01, it is expected that if a sequence is random, then the sequence should be rejected in 1000 sequences. If a P-value is equal or greater than 0.001, then the sequence with 99% confidentiality would be random .also if a P-value were less than 0.001, then the sequence with 99% confidentiality would be non-random.

2.2.3 Randomness Measurements

a) Uniformity

Generation of a sequence of True random or pseudorandom bits, in every situation the probability to encounter a zero or one is equally likely and definitely is 1/2, i.e., the probability of each is exactly 1/2. In the other hand if the length of sequences were n, then it will anticipated the number of zeros or ones be n/2.

b) Scalability

If a test is be implemented for a sequence, also it is applicable for the subsequences, which are random. Therefore if there existed a random sequence, then every subsequences of the sequence should be random. For the results, every subsequent must pass total test for randomness.

c) Consistency

The generator must be hardly dependent on starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed, or an RNG because of an output produced from a single physical output.

Table 2: Some Neccessory Deffinition Related to NIST

Bernoulli Random Variable	A variable which randomly gives one with probability p or zero with probability 1-p
Entropy	A measure of the disorder or randomness in a closed system
Binary Sequence	A stream of zeros and ones
Bit String	A sequence of bits
Block	A subsequence of a bit stream A block has a predefined length
Cumulative Distribution Function (CDF) F(x)	A function giving the probability that the random variable X is less than or equal to x, for every value x That is, $F(x) = P(X \leq x)$
Kolmogorov-Smirnov Test	A kind of statistical test which is used recognize if a set of data comes from a particular probability distribution
Probability Density Function (PDF)	A function which produce the "local" probability distribution of a test statistic
Run	An uninterrupted sequence of like bits
Seed	A primary value of a pseudo random number generator By using different seeds with the same generator, it generates different pseudorandom sequences

2.2.4 Random Number Generation Tests

One type of statistical tests including 15 tests is NIST test. NIST implemented to test the randomness of binary sequences, which generated by hardware or software based random number generator. The tests derived from NIST are specified as follow.

a) The Frequency (Monobit) Test

This test works on compute the congruence of ones and zeros for an entire sequence. The purpose of the test is to specify whether the number of appeared zero and ones are as the same as expected. On the other hand, it tests randomness of the sequence by checking that the probability of zeros and ones is equal to $\frac{1}{2}$ or not. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

b) Frequency Test within a Block

This test computes the ratio of ones in M-bit block of a sequence. This test specifies whether the frequency of ones in M-bit block is almost M/2 or not. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

c) The Runs Test

This testis about total runs number in a sequence. A run is a continuous sequence of bits. It focus on the question whether the number of runs of zeros and ones is as the same as attended for a random sequence. In the other hand, the test specifies too slow or too fast switches between zeros and ones in the sequence. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

d) Tests for the Longest-Run-of-Ones in a Block

This test is about the longest runs of ones of M-bit blocks. In the test, the aim is about specifying none quality between the length of longest run of ones in a sequence and the expected length of runs of ones in a random sequence. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

e) The Binary Matrix Rank Test

This test concentrate in testing the rank of disjoint sub-matrices of a sequence. . It also wants to determine the linear dependency among fixed length subsequences of the original sequence. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

f) Discrete Fourier Transform (Specral) Test

This test belongs to spectral methods, which is a class of procedure. Actually, Discrete Fourier transform is the base of this test. Periodic features of the bit sequences are specified by the Fourier transform test. The features demonstrate a deviation derived from randomness assumption. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

g) The Non-overlapping Template Matching Test

The aim of the test is to decline sequences from a given periodic pattern, presenting much many or very few happening. This test also can be used for decline sequences from a given non-periodic pattern, presenting much many or very few happening. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

h) The Overlapping Template Matching Test

The purpose of the test not only is declining the bit streams presenting much many or very few happening of m-runs from ones, but also reclaiming for finding irregular happening of any alternative pattern B. By considering the sequence length at least 10^6 bits, results that are more reliable will be achieved.

i) Maurer's "Universal Statistical" Test

This test introduced in 1992 by Ueli Maurer. The aim of the test is the possibility of compressing the sequence without losing any information (i.e. the sequence which are compressible is non-random). It was suggested the tested sequence length to be chosen as a long bit sequence.

j) The Linear Complexity Test

In this test, leaner complexity is used for testing randomness. By considering the sequence length at least 10^6 bits, results that are more reliable will be achieved. For achieving reliable results the other variables such N and M in order should be in the ranges $N \geq 200$ and$500 \leq M \leq 5000$.

k) The Serial Test

According to the fact that a random series have the feature of uniformity, which means that the chance of happening for m-bit pattern is equal to the chance of any other m-bit pattern occurrence, the aim of the serial test is to understand if the number of accidental events is close to the expected random series. In other words, the test evaluate the period of all possible overlapping *m*-bit subsequences in comparison with the sequence. By considering variable m at less than $\lfloor \log_2 n \rfloor - 5$, results that are more reliable will be achieved.

l) The Approximate Entropy Test

This test conditionally related to the reiterated models is the bit sequence. By considering variable m at less than $\lfloor \log_2 n \rfloor - 5$, results that are more reliable will be achieved.

m) The Cumulative Sums (Cusums) Test

This test relates to the greatest amount derived from fragmentary sums of indicated from the stream defined in the mode of ± 1. Big statistic values represent existence of much many ones or zeros at the sequence primary steps. However, tiny values represent viewed combined zeros and ones accidentally. By considering the sequence length at least 100 bits, results that are more reliable will be achieved.

n) The Random Excursions Test

The test is based on the number of cycles which is visited K times in a cumulative sum random walk. It should be considered that the sequence of zero and one is mapped to the suitable minus one and one sequence due to partial sums. By considering the sequence length at least 10^6 bits, results that are more reliable will be achieved.

o) The Random Excursions Variant Test

This test is based on the number of a special case, which is frequently happened in a random visit cumulative sum. In other words the aim is computing deviation, by sing the ideated times of encountering different conditions from random visiting. By considering the sequence length at least 10^6 bits, results that are more reliable will be achieved.

In conclusion, to be sure that our random numbers are good enough, the P-value and ratio for all tests should be considered. In other words, P-value and ratio of different test should be compared with the expected P-value, expected ratio, and have to pass acceptable and reliable number of tests among the tests that have been running. If the tested P-value becomes less than 0.01, then conclude that the sequence is non-random. Otherwise, we can conclude that the sequence is random and the result is reliable.

Actually if the P-value amount is equal or more than 0.01 or 0.001 we can be sure that the generated random numbers pass the tests with the level confidence more than 0.99 or 0.001, in other words the result will be accurate with 99% or 99.9% level of confidence.
The randomness of random numbers produced in bit sequences, by Diffusion RNG $_{\text{Light}}$ technique, which is discussed in this study, can be assessed by NIST test.
In addition, it is noticeable that for getting reliable results from NIST, size of a block where n represents the length of the sequence must be equal or less than $\lfloor \log_2 n \rfloor$.
Another important measure is the ratio that represents the number of sequences with P-value, which is equal or greater than a certain measure θ. This value should be in range of $[0.001\,,0.01]$. The restriction for the least ratio is:

$$ratio > p - 3\sqrt{\frac{p(1-p)}{s}} \quad , \quad p = 1 - \theta$$

2.3 Related Work

Actually, many proposals have wide concentration about formation of TRNGs and they have short view about TRNG physical approaches.

In the view of sources of randomness some proposals focus on finding a source, which provides a variations, also it is important to be sure that generated random numbers of the specific source are not repeatable. For instance in [9] a Xilinx Digital Clock Manager's (DCM) jitter which is combined with a high frequency clock plays a fundamental role in producing a random sequence. While [9] have some problems in handling predictable results, the [10] makes some modification to create a replacement. Although the source of both [9] and [10] are based on two clocks the aim of [10] is a development on [9]. In some other cases, the generator measures the response time distance of the hard disk readings and the speed of disk drive as a true random number [11]. Although in [12] photons derived from Light rays supply a source of random numbers, [13] uses the Received Strength Signal Indicator (RSSI) that measures the radio waves power. In addition, environmental readings often consist of some noises that can be used in TRNG modules [14]. Also, thermal noise is considered as a source to obtain true random numbers, in fact while the electricity flow pass through a resistor, a random thermal movement of electrons happens which can be used as true random number [15].

It is clear that the role of physical measurements and sensory component is inevitable to provide a source of true random numbers [16, 17]. An approach to collect and produce a trail of true random numbers in WSN is by means of a sensory component as it is implemented in [1]. However based on the fact that each laptop in WLAN can be considered as a sensory node in WSN, the implementation of TRNGs in context of ad hoc WLAN should be considered.

However apart from different types of physical measuring for providing true random numbers source, it is necessary to protect the generation process to reduce the probability of manipulating the random numbers. Although the produced sequences are aperiodic, some mechanisms are needed to create a secure transmission [1].

2.4 Problem Definition

Therefore, this study is going to implement a modification of Scatter RNG_{Light} implemented in Wireless Sensor Network (WSN) also is described in [1]. The modification includes some revision on the formation of the frames. In addition, some cryptographic equipment is considered to achieve security to produce true random numbers. For instance, usage of AES, Triple DES and SHA-1 for encryption and decryption of the transmitted frames, applying CMAC combined with the mentioned cryptographic algorithms in TRNG module are considered instead of DES and RC2 represented in [18]. However, these modifications, which are implemented in Ad hoc WLAN, called Diffusion RNG_{Light} explained and evaluated in Chapter 3 in details. Furthermore, randomness of the true random numbers produced by Diffusion RNG_{Light} will be evaluated in comparison with Scatter RNG_{Light} by NIST tests [7] in Chapter 4.

Chapter 3

DIFFUSION RNG$_{LIGHT}$ IMPLEMENTATION IN WLAN

The purpose of this chapter is the implementation of Diffusion RNG$_{light}$ technique. It is performed in context of Ad-hoc WLAN and it insists on generation of true random numbers in a secure way. Although this technique caused time consumption, the technique provides security. Each participant node can multicast a request to others in a form of encrypted frame for collecting some true random numbers. All nodes, which receive the request, after decryption and authentication, generate a random number by measuring some physical parameters. Then they make encrypted reply message and send it back to the demandant node. In the end requester, use the collection of all received random number to generate the true random number with a scrambling function.

Subsequently the details of these concepts are discussed in three sections. The first section addresses the structure of Diffused authentication Reading Collection (Diffused ARC$_{Light}$) message. The second section is about the organization of multicasting frames for requests and collecting frames for replies. The last section is related to generate true random numbers by using enhanced RNG protocol, which depends on SHA-1 (one-way) function and Cypher-based Message Authentication Code (CMAC). In this study, Visual C#.Net (which is the subset of Visual Studio.net) has been applied as the programming languages.

In the view of implementation, the modular interface is required. Figure 5, as a modular representation, illustrate the process of generating a true random number. In the figure laptop A and B that are available in context of an Ad hoc WLAN called Diffusion are assumed. Laptop A, playing demandant role, wants to multicast its request to available laptops in Diffusion network. Laptop B, which is available in the network, received the request as an encrypted frame named Request frame, then Discriminator function according to the type of the frame (which is equal to one) recognizes that the frame is Request frame, thus inserts it in Request queue. Respondent thread in B-side, after extracting the frame from Request queue, firstly, authenticates it and reads some physical measurement (i.e. CPU temperature). Then the thread inserts a produced random number derived from the measurements in to the frame. Finally, the thread after changing type of the frame (to zero); sends it back to laptop A as Reply frame. Discriminator in A-side after checking Type of the frame (which is equal to zero); insert it to Request queue. In addition, Authentication Collection function (A.C. function) extracts the frame and authenticates the frame then passes the extracted random number from the frame to Scramble function (which is sub function of A.C. function). Finally, Scramble function after getting all extracted random number from received frames, produced a true random number.

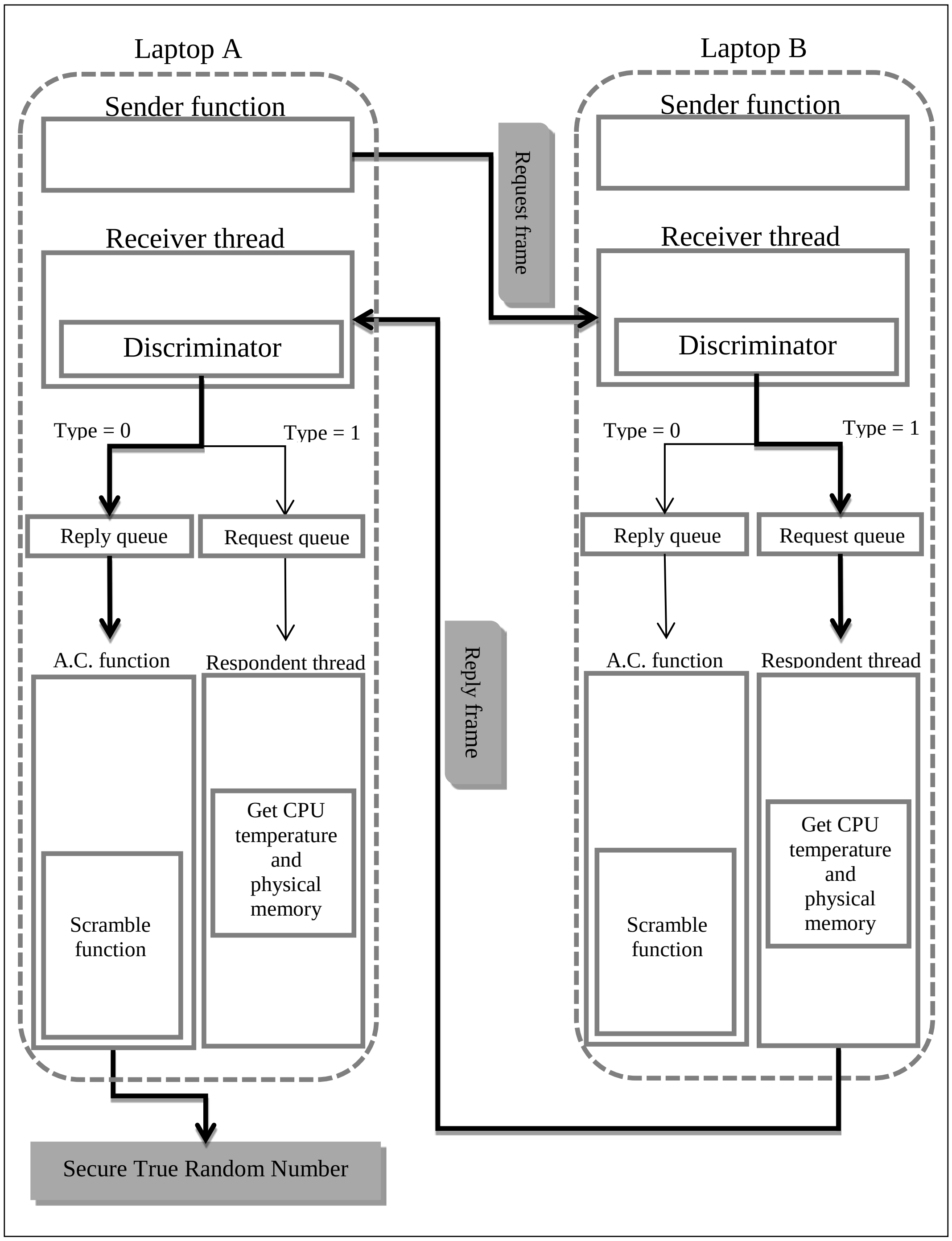


Figure 5: Communication Pattern Between Laptop “A” and “B”

3.1 Ad hoc WLAN Prefaces

According to the previous information, this study is going to implement of Diffusion RNG_{light}. Although, there exist many ways to establish a wireless networks but according to the similarity between characteristics of WSN, which is applied by Scatter RNG_{Light} [1] and Ad hoc WLAN, Diffusion RNG_{Light} is implemented in the context of Ad hoc WLAN. In this type one Laptop (node) prepares a context of WLAN used Mesh topology such a full graph. Therefore, the other nodes will be able to connect to this network called in this study Diffusion (the codes are represented in A.1).

Diffusion WLAN is a dynamic network and has some benefits. Firstly all nodes are able to establish Diffusion WLAN on their own, secondly, As soon as the Diffusion WLAN is interrupted, every nodes has enough potential to establish the Diffusion WLAN, the main point for this operation is every node which is more quick will establish the Diffusion. The third advantage of the Diffusion comes from the characteristic of Ad hoc every node can prepare a request for others in purpose of random numbers and able to answer to these requests as sending a random number to the demanding node.

3.2 Diffused ARC_{Light} Message Structure

By the use of symmetric concept, these benefits could be prepared in this implementation. Diffusion RNG_{light} technique is implemented symmetrically and simultaneously, so that nodes prepare their requests and answers named Diffused ARC_{Light} message (Diffused frame) illustrating in figure 6.

Diffused frames, which contains seven fields called; Encryption Type, Type, Payload length, IP address, Payload, Encrypted part discussed below on detail.

Encryption Type 4bit	**Type** 1bit	**Payload length** 3bit	**IP address** 32bit	**Payload** 48bit	**Encrypted part** 64 or 128bit

Figure 6: Fields of the Diffused $\text{ARC}_{\text{Light}}$ Message (Diffused Frame)

3.2.1 Encryption Type

The first field reserved by Encryption Type declared the algorithm used for digital signature or encrypting /decrypting the amount of Nonce and Hash code together called Encrypted part in the Last field. Actually, these fields reserved for another purpose to alter the size of Diffused frame to a sequence of Octets. Therefore, 4-bit space is necessary to consider this field.

3.2.2 Type and Payload Length

The second and the third fields contain Type (1 bit) and Payload length (3 bit). Type determines the type of diffused $\text{ARC}_{\text{Light}}$ message; when the frame is a request, the value of Type is zero otherwise, for reply this value is set to one.

Payload length represented the length of Payload derived from the third field, also the size of the field is 3 bit (because for introducing the size of a field in octet by the size of $48 = 6 * 8$ we need to reserve at least 3 bit). Payload length is used because the length of the payload is changeable and flexible. Payload length value shows how many octets the Payload contain.

3.2.3 IP Address

The fourth field is related to IP address for the demandant or the node replies to the requests. This field is necessary because when nodes multicast their requests, or their replies, the nodes should recognize who send the messages. In [1] the author used 12 bit for the field but in this study for showing IP address in an Ad hoc WLAN (which structured like the example 254.255.255.255) every nodes are inevitable to reserve leastwise 32 bit in the frame to recognize IP address.

3.2.4 Payload

The fifth field includes Payload containing 48 bit. It is the random number generated as measured with some physical phenomenon by the use of two numbers, CPU temperature [19] and capacity of physical memory [20]. In other words, the current amount of CPU temperature multiplied by the amount of free space in RAM at that moment is injected to the payload as a bit sequence. To avoid repetition of the measured value of CPU temperature, least significant bits of the multiplication are being fed in to the payload.

3.2.5 Encrypted Part

Finally, the Encrypted part is derived from the last field of Diffused $\text{ARC}_{\text{Light}}$ message. Actually, this frame divided in two parts each of which are used to increase the security of the frame. In other words, these two categories are used for Authentication of the demanding and guaranteeing the Integrity of the frame. The amounts of Nonce and Hash code are encrypted. This study used AES and Triple DES algorithms in mode of Cypher Block Chaining (CBC) to encrypt/decrypt the pair of Nonce and Hash code in terms of

Encrypted part. The total size of this field by using AES encryption/decryption set to 128 bits and by using Triple-DES encryption/decryption set to 64 bits (figure 7).

Encrypted part 64 or 128bit	
Nonce 8bit	**Hash code** 56 or 120bit

Figure 7: Pair of Nonce and Hash Code Derived from Encrypted Part

a) Nonce

Nonce field considerate to threaten and prevent replay attacks. The size of the Nonce containing a constant will be increase by the destination node (who will send a random number to the demandant) is 8 bit.

b) Hash Code

By considering the size of Nonce and the Input (Plaintext) used for two encryption algorithms (AES and Triple DES) Hash code size to have a reliable plaintext for AES or Triple DES in order is needed to be leastwise 120 or 56 bit. On the other hand, this part contains, a transform of the 120 or 56 least significant bits of computed digest derived from concatenated of the other field with Nonce together using SHA-1.

According to the previous information about structure of frames and the fields it is appeared that the field relating to the payload can be eliminated in the request frames.

3.3 Diffusion RNG$_{\text{Light}}$ Organization

This study categorizes Diffusion RNG$_{\text{light}}$ in three phases:

- Sending request
- Receiving
- True Random Number Generation (TRNG)

Note that the third phase is a subset of the second phase but in this study because of the importance of the Generating Random number, it determines as a separate phase. By the way, these phases can be presented in two categories, multicasting and collecting illustrating in figure 8. In fact, the composition of these phases is Diffused ARCLight protocol.

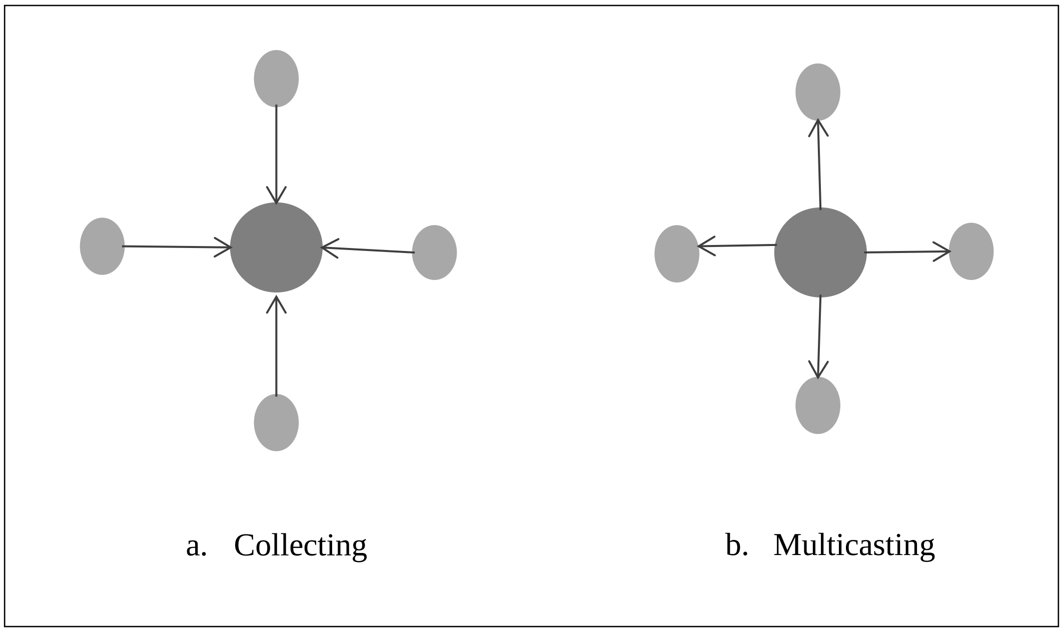


Figure 8: Categories of the Implementation

After establishing the WLAN and connecting the nodes are accessible by the WLAN nodes can freely ask some requests and reply them, so this implementation is Symmetric and synchronous for the sake of symmetric and synchronous it is unavoidable to send request and receive reply in parallel in programing. The structure of implemented Diffusion RNG_{Light} is illustrated in Figure 9.

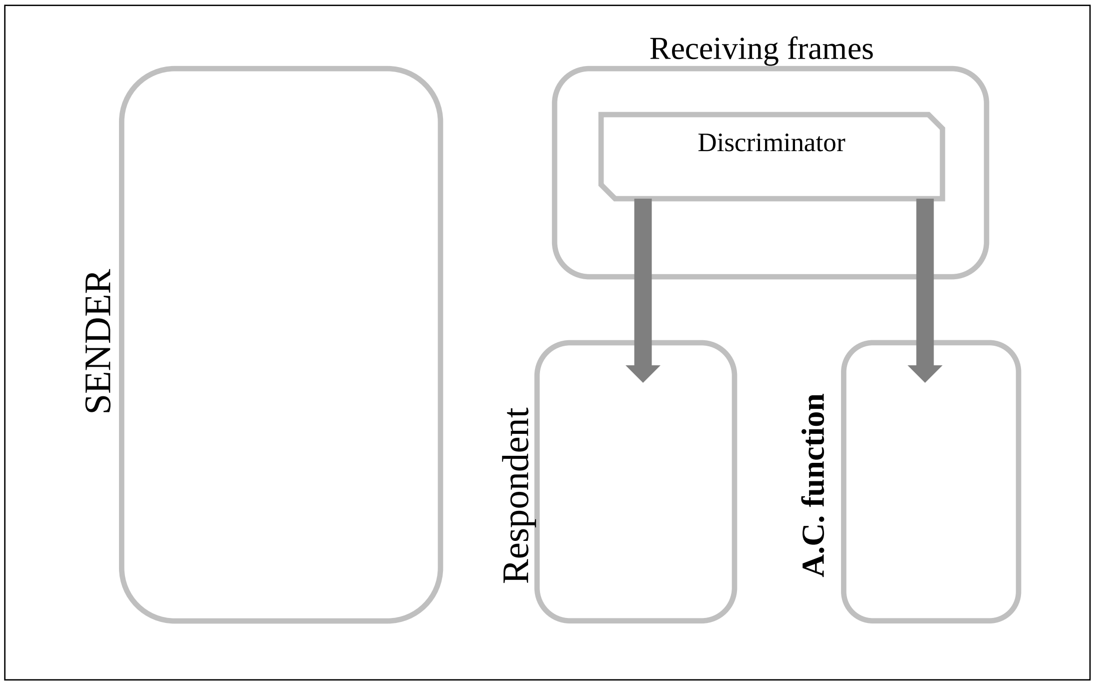


Figure 9: Structure of Diffusion RNG_{Light} Implementation

3.3.1 Multicasting Request

Particularly the first part of the study is about sending request. This part also is defined and programed as a function. The function is responsible for the role of multicasting requests named Sender. Sender function prepares Diffused ARC_{Light} message (Diffused frame) of requests to notify the other available nodes that it needs some Random Numbers.

These requests based on the previous discussions includes; Encryption Type, Type, Payload length, IP address, Encrypted part (contains encryption of the pair of Nonce and Hash code).it is obvious that, the Payload field is eliminated, because it is unnecessary and makes redundancy.

Sender after filling the first four Fields and generating a Nonce it starts to compute a message digest (Hash code) from the concatenation of the first 4 fields and 8bit Nonce (Figure 10). Now the input of the Encrypted part field containing the pair Nonce and Hash code are ready. After encrypting, the pair it will inserted to the field. Finally, the frame is ready to multicast (the related code is presented in A.2).

Encryption Type 4bit	**Type** 1bit	**Payload Length** 3bit	**IP address** 32bit	**Encrypted part** 64 or 128bit	
	For request frame is set 1			**Nonce** 8bit	**Hash code** 56 or 120 bit

Figure 10: Illustration of the Diffused Frame for Request that the Payload Field was Eliminated

3.3.2 Receiving

This phase is created to receive the frames and preparing reply fames. Receiving phase can be detached in some categories.

a) Received Frames

In this part, there exists a thread called Receiving frame. This thread will receive the frames by listening to the other nodes. The technique was applied for listening is the UDP socket programming.

b) Detachment of Requested and Replied Frames

As it was clear in figure 9, there is a function named Discriminator inside Receiver thread. Receiver thread always received two kind of frame, which are deferred by the Type field, thus requiring a function to divide these two kinds of frame is felt. Actually, this function works until the Receiving frame thread works. Discriminator function has duty to recognize the received frames are a request frames or is reply frames. According to this recognition, Discriminator inserts the request frames in a queue named Request queue and reply frames in the other queue named Reply queue.

c) Responding to the Requests

After inserting frames in a request queue, another thread should be defined for executing in parallel with the Receiving thread. The other reason is about lifetime. Every node after ordering a request set a time as life time to increase the security, it leads to create a thread being in parallel with Receiving frames thread. This thread named respondent.

For responding the frames, firstly, Receiving frame thread should authenticate the requesters and its frame. For authentication, Respondent frame should decrypt the last field of a frame using AES or Triple DES algorithms in Cypher Block Channing mode. The four first fields, Nonce and Hashcode using decryption should be retrieved, therefore by computing a message digest of concatenation of the first four fields with

Nonce field together by the using of SHA-1, then compare with the Hash code the integrity and authentication of the frame can be assessed. Secondly, the reply frames will be constructed follow this order; the Type and IP address fields should be replaced by zero and IP address of itself respectively. The Payload filled by generating random number, which was discussed before. Nonce field replaced by increasing its previous value one time. After preparing these parts Hash code will computed from them, then the Nonce and the transformed output of the Hash code will encrypted by the use of AES and Triple DES in Cypher Block Channing (CBC) mode (figure 11). After constructing the reply frame, finally, the frame will send back to the demandant IP. In addition, UDP socket programming applied for sending the reply, by knowing the destination IP address retrieved from IP addresses field of the request frame.

Notice that for sending the reply to the demandant IP address, applying the UDP multicast in Socket programming we should insert the demandant IP address instead of 169.255.255.255 (UDP codes are defined in A.2).

Encryption Type 4bit	**Type** 1bit	**Payload Length** 3bit	**IP address** 32bit	**Payload** 48bit	**Encrypted part** 64 or 128bit	
	For Reply-frame is set 0				**Nonce** 8bit	**Hash code** 56 or 120bit

Figure 11: Reply Frame, Which Obviously Include Payload Field

3.3.3 True Random Number Generation (TRNG)

This phase want to determine after receiving reply frames what would be happens in the programming. In this study, a function named Authentication Collection (A.C.) is defined to collect, authenticate and check the integrity of reply frames. Actually, it is declared that all these duties lead to generate a true random number. To this aim, this phase is categorized in the two followed parts.

a) Collecting the Frames

As it was mentioned in section 3.3.2, Reply queue, which was filled by Discriminator function after a specified lifetime, will be released by the function for A.C. function to collect the frames. While extracting frames from Reply queue, their integrity and authenticity should be investigated. In fact, the way that discussed in section 3.3.2 and these procedures are totally related to Diffused ARC_{Light} protocol.

b) Generating a True Random Number

After collecting Authenticated frames and extracting the Payload fields of the frames. Payload values will be fed as inputs in a function, named Scramble function. The output of the function will be the generated true random number.

As it was mentioned at the beginning of this chapter also in section 3.3, Diffused ARC_{Light} protocol is the core of the Diffusion RNG_{Light}. It is clear that the basic RNG module has been enhanced in Diffusion RNG_{ligh}t, which is added Local Clock (LC) and Digest Block, as represented in Figure 12.

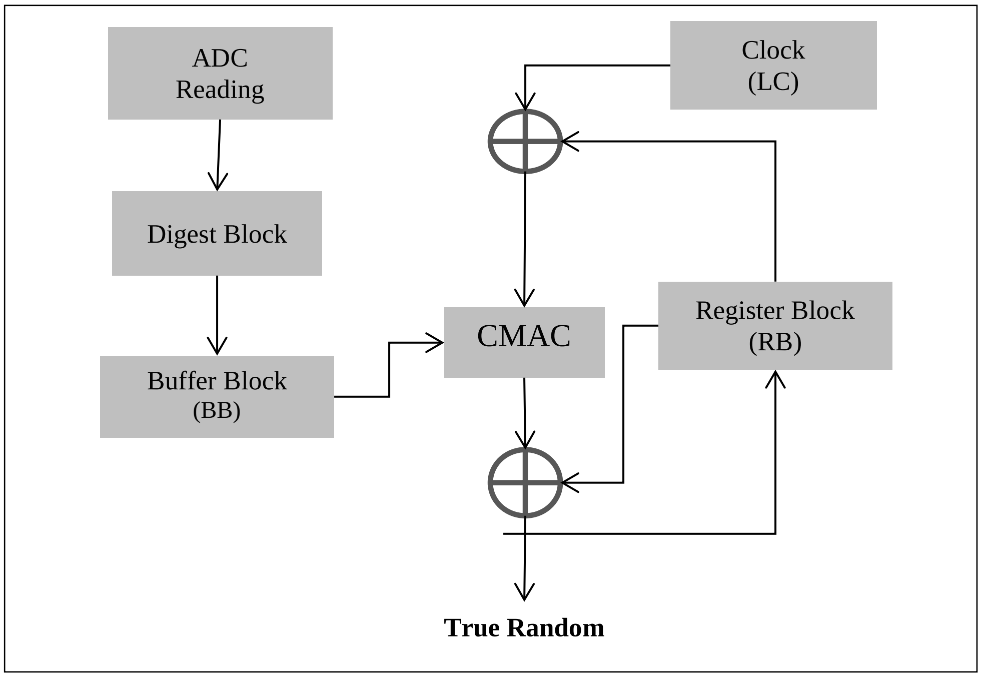


Figure 12: Illustration of Enhanced RNG Module

In this section, First of all value of the Payload field is extracted in size of the value of the Payload length field. Scramble function accepts as input the concatenation of the Payload value authenticating in section 3.4.1and are binary. Actually, Scramble function is based on the enhanced RNG.

Mixer after receiving its input computes160 bit digest of the concatenation of reading sets used SHA-1.

$$Y = SHA - 1(X_1 \parallel X_2 \parallel \cdots \parallel X_n)$$

The equation below depicted the Hash code (Y) will be added to the 160 bits non-circular left shifted value of the Buffer Block (BB). Therefore when the value of BB including 256-bit is shifted is shifted 160-bits, caused that, 96 least significant bits of BB to be replaced by Y (the output of the hash code). Actually, (≪) is an operator illustrating the bitwise left-shift.

$$BB = (BB \ll 160) + Y$$

The next step is fed output of Buffer Block in to Cipher-Based Message Authentication Code (CMAC) which is discussed in [6]; XOR of the 16 least significant bits of Local Clock (LC) and 64-bit vale of Register Block (RB) is applied for the Key of CMAC. It is noticeable the primary RB and BB includes set of ones.

According to CMAC algorithm Illustrated in Figure 12, the outcomes of this step will be XOR by the 64-bit Register Block Value. Finally, the result of the operation is True Random Number (TRN). The next formulas depict the sequence of TRNG in summarize.

$$TRN = CMAC(BB, Key) \oplus RB$$

$$Key = RB \oplus LC$$

Chapter 4

RANDOMNESS QUALITY OF GENERATED RANDOM NUMBERS BY DIFFUSION RNG$_{LIGHT}$

After implementation and execution of Diffusion RNG$_{Light}$ in C#.Net, some bit sequences as random numbers are achieved. These sequences which are out comes of the protocol are applied in NIST test to assess the quality of randomness [7]. First of all after obtaining the random numbers and saving them immediately one after one in a file, a long bit sequences stored in a file is gained. It is clear that the file containing long bit sequences are applied in NIST test. As it was mentioned in Chapter 3 to gain qualified results from NIST test, the sequence length should be at least 10^6.

In this study after executing the program on five laptops containing Intel Core i5 CPU and 2GB RAM and Windows 7 as Operating System, which were in distance of around 3 meters far from each other, consequence of results in length of $4 \times 10^7$was gained in two days. However, each day every laptop produced 4×10^6binary bits which means that 2×10^7binary bits are achieved from the five laptops. Totally, in two days about 40 MB random bits saved in a file. Therefore, in the test the length of sequence and the number of tested sequences (runs) are assumed in order 10^6 and 40. Secondly, after preparing the file (long length sequence) and arranging the inputs, NIST tests were applied in the long length sequences produced as results by Diffusion RNG$_{Light}$. Table 3 and Table 4 illustrate the inputs and outcomes of the NIST test.

Table 3: The Input Values of the Statistical Tests

Parameter	Value
Length of each sequences(L)	10^6
Number of tested sequences(s)	40
Threshold for P-value	0.01
Ratio value	0.942

Table 4: Comparison of the Expected and Observed Results

Test name	Scatter RNG In WSN [1]		Diffusion TNG In WLAN Triple DES used		Diffusion TNG In WLAN AES used	
	P-value	Ratio	P-value	Ratio	P-value	Ratio
Frequency	0.1223	0.9834	0.4465	0.989	0.5786	0.995
Block Frequency	0.3508	0.9910	0.6931	0.988	0.3565	0.943
Runs	0.1223	0.9811	0.2927	0.987	0.2565	0.978
Longest Run	0.5341	0.9916	0.5120	0.989	0.8932	0.997
Binary Matrix Rank	0.7351	0.9853	0.7727	0.992	0.7823	0.997
Discrete Fourier Transform	0.2135	0.9910	0.4812	0.991	0.5253	0.994
Non Overlapping template matching	0.4602	0.9893	0.5241	0.990	0.723	0.991
Overlapping template matching	0.3509	0.9831	0.2733	0.991	0.0751	0.980
Maurer’s Universal Statistical	0.8065	0.9996	0.7834	0.950	0.1296	0.945
Linier Complexity	0.8965	0.9923	0.9013	0.96	0.1973	0.954
Serial	0.5348	0.9974	0.7702	0.998	0.7943	0.999
Approximate entropy	0.7451	0.9959	0.7435	0.988	0.9863	0.997
Cumulative sums	0.7392	0.9882	0.8662	0.989	0.8798	0.99
Random excursion	0.6402	0.9811	0.4719	0.984	0.6714	0.987
Random excursion variants	0.7502	0.9941	0.6246	0.985	0.0356	0.961

Because, NIST test program are implemented in GCC and according that the LINUX operating systems containing GCC compiler, it was unavoidable to use Linux to execute NIST tests.

Finally, earned results from NIST test indicate that P-value of every test (such as Frequency test) are greater than 0.01 (value of α called null-hypothesis) assumed as default. Consider the Table 4 two inferences are achievable. Comparisons of the three techniques are discussed in three parts.

According to the p-values of diffusion RNGLight using Triple DES it can be easily understood that all these values are greater than 0.01 and they passed all fifteen tests. However by comparison of diffusion RNG_{Light} using Triple DES with Scatter RNG_{Light} in case of p-values, the produced random number of diffusion RNG_{Light} using Triple DES are more random than Scatter RNG_{Light}.

Obviously, the results show that for Frequency, Block frequency, discrete furrier transform and serial tests the p-values in Triple DES have the growth rate more than 0.2 in comparison with Scatter. While p-values for Runs, Binary matrix rank, Non-overlapping template matching, Linier complexity and Cumulative sums are not increasing considerably. In other words, Triple DES in 60% (nine tests out of fifteen tests) achieved more randomness than Scatter. In the view of ratio, approximate entropy and Random excursion variants tests have a significant decrement while totally in eight out of fifteen test (53%) Triple DES appears more successful than Scatter.

Considerably, the p-values of diffusion RNG_{Light} using AES are greater than 0.01, thus they passed all the fifteen tests.

As it represented in Table 4 the p-values and ratios in all tests for diffusion RNG_{Light} using AES is growing in comparison with Scatter except Block frequency, Runs, Overlapping template matching, Maurar ‘s universal statistical, Linier complexity and random excursion variants tests which means that entirely 60% improving.

Table 5: The Improvement P-values of the Different Usage of Diffusion RNG

	Diffusion RNG in WLAN Triple DES used	Diffusion RNG in WLAN AES used
Minimum improved values	0.0048	0.0057
Maximum improved values	0.3423	0.4563
Mean of improved values	0.1747	0.2045
(number of improved values > Mean) %	44%	54%

Therefore, the Table 5 shows 44% of the improved P-values of Diffusion RNG using Triple DES become more than mean value, also among the improved P-values of Diffusion RNG using AES, 54% of them are more than mean value. In conclusion, the improvement percentage of Diffusion RNG using AES is 10% more than Diffusion RNG using Triple DES.

Chapter 5

CONCLUSION

According to the previous discussion, WLAN security will not be achievable, unless by preventing attacks and threats of WLAN. Accordingly, it is unavoidable to use cryptographic algorithms and for additional assurance, hash codes to improve the security. Respectively random numbers used by some cryptographic algorithms as session key, nonce or salt play a critical role in security of WLAN.

In this study, Diffusion RNG_{Ligth} technique, which is modification of Scatter RNG_{Ligth} technique, is implemented. Although Scatter RNG_{Ligth} technique is performed on sensor nodes in a Wireless Sensor Network (WSN) to generate true random numbers, Diffusion RNG_{Ligth} technique is implemented on distributed laptops in context of Ad hoc WLAN. Moreover, the secure framework is remodeled in some parts also DES is replaced with AES and Triple-DES for encryption/decryption and CMAC application. Furthermore, the random numbers produced by Diffusion RNG_{Light} have been qualified by some statistical tests named NIST such as experimenting in [1]. Accordingly, the results show that the number of P-values in Diffusion RNG_{Light} using Triple-DES is 60% greater than the P-values of Scatter RNG_{Light}, while Diffusion RNG_{Light} technique using AES achieved 60% greater P-values than Diffusion RNG_{Light} using Triple-DES.

REFERENCES

[1] L. R. Giuseppe, M. Fabrizio and O. Marco, "Secure Random Number Generation in Wireless Sensor Networks," *SIN '11 Proceedings of the 4th international conference on Security of information and networks,* p. 175-182, Sydney, NSW, Australia, November 14 - 19, 2011.

[2] W. Terrill, "WLAN Security Today:Wireless more Secure than Wired," Siemens Enterprise Communications, München, Germany, July 2008, p. 4-9.

[3] S. A. Camtepe and B. Yener, "Key Distribution Mechanisms for Wireless Sensor Networks: a Survey," Rensselaer Polytechnic Institute , Computer Science Department, 2005, p.1-5.

[4] G. Anastasi, G. Lo Re and . M. Ortolani, "WSNs for Structural Health Monitoring of Historical Buildings," *IEEE 2nd confrence on Human System Interactions (HSI),* p. 574-576, Catania, Italy, May 21-23, 2009.

[5] Z. Benenson, N. Gedicke and O. Raivio, "Realizing Robust User Authentication in Sensor Networks," *ACM, Workshop on Real-World Wireless Sensor Networks (REALWSN05),* p. 65-72, Stockholm, Sweden, June 20-21, 2005.

[6] W. Stallings, "Cryptography and network security," Prentice Hall, 5th edition, 900 p.

[7] A. Rukhin, J. Soto, J. Nechvata and M. Smid, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Application," *Information Technology Laboratory (ITL)*, no. National Institute of Standards and Technology (NIST), p. 1-1 to 5-8, April, 2010.

[8] C. Paar and J. Pelzl, "Understanding Cryptography," Springer, 2010, 372 p.

[9] S. M. Kwok and E. Y. Lam, "FPGA-based High-spead True Rando Number Generator for Cryptographic Aplications," *TENCON 2006. 2006 IEEE Region 10 Conference*, p. 1-4, Hong Kong, November 14-17, 2006.

[10] K. Tsoi, K. Leung and P. W. Leong, "High performance physical random number generator," *IET Computer & Digital Techniques*, vol. 1, no. 4, p. 349-352, July, 2007.

[11] M. Jakobsson, E. Shriver, B. Hillyer and A. Juels, "A Practical Secure Physical Random Bit Generator," *5th, ACM on Computer and Communications Security (CCS)*, p. 103-111, San Francisco, CA, USA, November 02 - 05, 1998.

[12] A. Stefanov, N. Gisin, O. Guinnard, L. Guinnard and . H. Zbinden, "Optical Quantum Random Number Generator," *Journal of Modern Optics,* vol. 47, no. 4, p. 595–598, 2000.

[13] R. Latif and M. Hussain, "Hardware-based Random Number Generation in Wireless Sensor Networks (WSNs)," *Advances in Information Security and Assurance,Third International Conference and Workshops,* vol. 5576, p. 732-740, Seoul, Korea, June 25-27, 2009.

[14] V. Gaglio, A. De Paola, M. Ortolani and G. Lo Re, "A TRNG Exploiting Multi-Source Physical Data," *the 6th ACM workshop on QoS and security for wireless and mobile networks,* p. 82-89, Bodrum, Turkey, October 17 - 21, 2010.

[15] B. Jun and P. Kocher, "The INTEL Random Number Generator," Cryptography Research Inc. white paper, San Francisco, California, 1999. [Accessed 4 July 2013].

[16] S. Callegari, R. Rovatti and G. Setti, "Embeddable ADC-based True Random Number Generator for Cryptographic Applications Exploiting Nonlinear Signal Processing and Chaos," *IEEE Transactions on Signal Processing,* vol. 53, no. 2, p. 793-805, 2005.

[17] I. Vasyltsov, E. Hambardzumyan, Y. S. Kim and B. Karpinskyy, "Fast Digital TRNG Based on Metastable Ring Oscillator," *Cryptographic Hardware and*

Embedded Systems–CHES, vol. 5154, no. Samsung Electronics, SoC R&D Center, System LSI, Korea, p. 164-180, 2008.

[18] H. Alavizadeh, "Distributed Random Number Generator," *Eastern Mediterranean University, Master Thesis*, 1 July 2013.

[19] "ACPI Thermal Zone," Microsoft Developer Network (MSDN), 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa939962(v=WinEmbedded.5).aspx. [Accessed 25 05 2013].

[20] "PhysicalMemory Class," Microsoft Developer Network (MSDN), [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa394347(v=vs.85).aspx. [Accessed 25 05 2013].

APPENDICES

Appendix A: Programming Part

The table below is a short view on the programing formation of the study:

Component	Subject	Description
A.1 Initialization	Ad-hoc WLAN	• An ad-hoc wireless network is made automatically that each laptop can join or leave the network.
A.2 UDP	Sending	• UDP protocol is used to multicast a request for random numbers in a network • For replying to demanding IP
	Receiving	• For receiving a frame from a specific IP
A.3 Threads	Receiver Thread	• The thread always listening to the port for received frames • Divides the receiving frames due to their Type (request=1 / reply=0) • Puts them in a related queue
	Responder Thread	• The thread always checking the queue if any request frame is received;

		• Authenticate the frame; • Measure some specific parameter to produce random number; • make a frame to reply; • Encrypt(AES/Triple DES) and send it to the requester
Queues	Request Queue	The queue contains of the received request frames.
	Reply Queue	The queue contains of the received reply frames.
A.4 Timer	Life Time	• It shows the expected time for collecting random numbers • It starts to work when a request is sent • When it is expired then the Reply Queue is locked and no reply message is accepted and the collecting random numbers are prepared
	Network Timer	• The timer is defined to refresh the list of all available laptops which joined or leaved the network

A.5 Authentication Reading Collection	Authentication collection	• Before the expiration of the timer collects all reply frames then decrypt(AES/Triple DES) and authenticate them
	Scramble function	• Use achieved random numbers as an input, using Hash function, CMAC and a combination methods to generate a secure true random number

A.1 Initialization

First and the most an ad-hoc wireless network should be established that the nodes or laptops could communicate with each other. The ad-hoc network is named “Diffusion”. In this purpose, some initialization should be done. The most important part of initialization code is related to set the “Diffusion” profile and connect to it. In addition, the threads, P1 and P3, are defined as a receiver and a responder respectively:

```
private void Form1_Load(object sender, EventArgs e)
{
count = 0;
avrage = 0;
File = new System.IO.StreamWriter("..//Results.txt");
File2 = new System.IO.StreamWriter("..//Results2.txt");
//--------------------------------------------------
sending_end_point = new IPEndPoint(send_to_address, 11000);
//--------------------------------------------------
IPHostEntry myHostInfo = Dns.Resolve(Dns.GetHostName());
IP = myHostInfo.AddressList[0].ToString();
lblIP.Text = IP;
lblName.Text = Dns.GetHostName();
//--------------------------------------------------
foreach (WlanClient.WlanInterface wlanIface in client.Interfaces)
{
string xml_Diffusion = "<?xml version=\"1.0\"?>\r\n<WLANProfile
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v1\">\r\n\t<name>Diffusi
on</name>\r\n\t<SSIDConfig>\r\n\t\t<SSID>\r\n\t\t\t<hex>446966667573696F6E</hex>\
r\n\t\t\t<name>Diffusion</name>\r\n\t\t</SSID>\r\n\t\t<nonBroadcast>false</nonBro
adcast>\r\n\t</SSIDConfig>\r\n\t<connectionType>IBSS</connectionType>\r\n\t<conne
ctionMode>manual</connectionMode>\r\n\t<MSM>\r\n\t\t<security>\r\n\t\t\t<authEncr
yption>\r\n\t\t\t\t<authentication>open</authentication>\r\n\t\t\t\t<encryption>n
one</encryption>\r\n\t\t\t\t<useOneX>false</useOneX>\r\n\t\t\t</authEncryption>\r
\n\t\t</security>\r\n\t</MSM>\r\n</WLANProfile>\r\n";
wlanIface.SetProfile(Wlan.WlanProfileFlags.AllUser, xml_Diffusion, true);

wlanIface.Connect(Wlan.WlanConnectionMode.Profile, Wlan.Dot11BssType.Any,
"Diffusion");
}

//-------------------cpu information------------//
m_CPUCounter = new System.Diagnostics.PerformanceCounter();
m_CPUCounter.CategoryName = "Processor";
m_CPUCounter.CounterName = "% Processor Time";
m_CPUCounter.InstanceName = "_Total";
float cpu = m_CPUCounter.NextValue();
//----------------------------------------------//

temp.Text = GetTemperature (1);
p1 = new Thread(new ThreadStart(receiver));
p3 = new Thread(new ThreadStart(responder));
p1.Start();
p3.Start();

//---------------------------------------

}
```

In addition, a list of all available laptops in the network is provided and presents in the list box:

```
void net_view()
{
nb = new NetworkBrowser();
lstNetworks.Items.Clear();
string nbh = "";
foreach (string pc in nb.getNetworkComputers())
{
try
{
addresslist = Dns.GetHostAddresses(pc);
foreach (IPAddress address in addresslist)
{
nbh = address.ToString();
}
}
catch (Exception)
{
nbh = "No direct neighbour";
}

ListViewItem item = new ListViewItem(pc);
item.SubItems.Add(nbh);
lstNetworks.Items.Add(item);
nbh = "";
}
}
```

When a laptop is joined to the network, is added to the available laptop list. Actually, it is possible to multicast a request to gather random numbers. These codes show how a laptop encrypts a frame as a request and sends it to all others, by using UDP protocol:

```
private void btn_cnt_Click(object sender, EventArgs e)
{

collect_flag = false;
listBox1.Items.Add("-----------------------------------------");
string Sframe = "";
if ((Convert.ToByte(txt_pl.Text) > 6)||(Convert.ToByte(txt_pl.Text) == 0))
txt_pl.Text = "6";
byte pl =Convert.ToByte(txt_pl.Text);
byte[] Type_payload = new byte[1];
byte []Sender=new byte[4];
byte[] Nonce = new byte[1];
byte[] Hash = new byte[7];
Nonce[0] = 1;
Type_payload[0] = 0x08;

Type_payload[0] = (byte)(Type_payload[0] | pl);
string crypto_type = "";
crypto_type = cmb_Crypto_type.Text;
switch (crypto_type)
{
case "AES":
Type_payload[0] = (byte)(Type_payload[0] | (0x10));
break;
case "Triple DES":
Type_payload[0] = (byte)(Type_payload[0] | (0x30));
break;
default:
MessageBox.Show("Choose crypto type!");
break;
}
for (int i = 0; i < 4; i++)
Sender[i] = Convert.ToByte(IP.Split('.')[i]);
//------------------  Hash Code And Encryption is provided:

string test = Encoding.ASCII.GetString(Sender);
string Hashcode = h(1, Encoding.ASCII.GetString(Type_payload) +
Encoding.ASCII.GetString(Sender) + Encoding.ASCII.GetString(Nonce));

byte[] transform = Encoding.ASCII.GetBytes(Hashcode);
for (int i = 0; i < 7; i++)
Hash[i]=transform[i];
string plaintext =
Encoding.ASCII.GetString(Nonce)+Encoding.ASCII.GetString(Hash);
byte[] encrypted = EncryptStringToBytes(plaintext,crypto_type);

//-------------------- Preparing a frame to send:
byte[] send_buffer = new byte[5+encrypted.Length];
send_buffer[0] = Type_payload[0];
for (int i = 1; i < 5; i++)
send_buffer[i] = Sender[i-1];

for (int i = 5; i < 5+encrypted.Length; i++)
send_buffer[i] = encrypted[i-5];
lifetime.Interval = Convert.ToInt32(textBox1.Text);

//-------------------- multicasting:
listBox1.Items.Add("Multicasting");
multi_send(send_buffer);
listBox1.Items.Add("DONE");
listBox1.Items.Add("LifeTime Timer is started");
listBox1.Items.Add("DONE");

}
```

The function bellow gets a frame as an input and encrypts it due to encryption types, AES or Triple DES, then make an output as a string of bytes:

```
static byte[] EncryptStringToBytes(string plainText,string Encr_type)
{
byte[] encrypted;
switch (Encr_type)
{
//------------------------------------------------//
case "AES":
using (AesCryptoServiceProvider tdsAlg = new
       AesCryptoServiceProvider())
{
tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess_the_Key!?");
tdsAlg.IV = Encoding.ASCII.GetBytes("towish_is_toable");
tdsAlg.Mode = CipherMode.CBC;
// Create a decrytor to perform the stream transform.
ICryptoTransform encryptor = tdsAlg.CreateEncryptor(tdsAlg.Key, tdsAlg.IV);
// Create the streams used for encryption.
using (MemoryStream msEncrypt = new MemoryStream())
{
using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
{
using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
{
//Write all data to the stream.
swEncrypt.Write(plainText);
}
encrypted = msEncrypt.ToArray();
}
}
}
return encrypted;
//----------------------------------------------------//
case "Triple DES":
using (TripleDESCryptoServiceProvider tdsAlg = new
TripleDESCryptoServiceProvider())
{
tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess_the_Key!?");
tdsAlg.IV = Encoding.ASCII.GetBytes("big_bang");
tdsAlg.Mode = CipherMode.CBC;
tdsAlg.BlockSize = 8;
ICryptoTransform encryptor = tdsAlg.CreateEncryptor(tdsAlg.Key, tdsAlg.IV);
using (MemoryStream msEncrypt = new MemoryStream())
{
using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
{
using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
{
swEncrypt.Write(plainText);
}
encrypted = msEncrypt.ToArray();
}}}
return encrypted;
default:
return encrypted=Encoding.ASCII.GetBytes("Default Error");
}}
```

A.2 UDP Programming

Moreover, Lifetime timer starts counting a particular amount that is set to this timer, when the request frame is multicast in the network. The codes related to the UDP for multicasting a message to all reachable IP are as followed:

```
// to define a socket object which is the fundamental device used to network
```

```
Communications

Socket sending_socket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram,ProtocolType.Udp);

// to define an address object and populate it with the IP address.
// the particular address which ends in 255 means for sending to all devices
whose address begins with 169.254.

IPAddress send_to_address = IPAddress.Parse("169.254.255.255");
IPEndPoint sending_end_point;


private void multi_send(byte[] send_buffer)
{
try
{

if (btn_cnt.Enabled == true)
{
lifetime.Enabled = true;
sending_socket.SendTo(send_buffer, sending_end_point);
if (checkBox1.Checked)
{
                            sending_end_point = new
                            IPEndPoint(IPAddress.Parse(broadcast.Text), 11000);
                             sending_socket.SendTo(send_buffer,
                            sending_end_point);
}
btn_cnt.Enabled = false;
}
}
}
catch (Exception send_exception)
{
Console.WriteLine(" Exception {0}", send_exception.Message);
}

try
{
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}}
```

A.3 Thread

Immediately after connecting to the network, Receiver Thread and Responder Thread start working. The receiver thread receives the frames and calls Discriminator function:

```
void receiver()
        {
            request_rear = 0;
            reply_rear = 0;
            request_front = 0;
            reply_front = 0;

            while (true)
            {
               receive_byte_array = listener.Receive(ref groupEP);
               received_data = Encoding.ASCII.GetString(receive_byte_array,0,
              receive_byte_array.Length);

                Discriminator(receive_byte_array);

            }
        }
```

The Discriminator function checks the Type of the frames and categorizes them in two queues. Actually Discriminator puts the reply frames in Reply Queue and the request frames in Request Queue if the Lifetime is not expired. The codes related to this part are below:

```
void Discriminator(byte[] frame)
{

if (((byte)(frame[0] & (0x08)) / 8) == 1)
{
request_queue[(request_rear++) % 10000] = frame;
if (p3.ThreadState == ThreadState.Suspended)
p3.Resume();
}
else if (((byte)(frame[0] & (0x08)) / 8) == 0)
{
if (lifetime.Enabled == true)
reply_queue[(reply_rear++) % 10000] = frame;
}
else
{
MessageBox.Show("Incorrect Received!");
}
}
```

Note: if the request queue is empty, responder will be suspended until any new request frame is added to the queue.

The Responder Thread checking the Request Queue whether the requests frame is available or not. If any frame is available then take the frame from the queue. After authentication, measure the specific parameters to create a Payload. Then make a reply frame and set it with appropriate values, send it back to the demandant IP.

```
void responder()
{
//responder_flag = true;

IPEndPoint sending_end_point;
double random_value=0;
//------------------------//
while (true)
{
if (request_front != request_rear)
{
byte[] Current_frame = new byte[request_queue[(request_front) % 10000].Length];
Current_frame = request_queue[(request_front) % 10000];
request_front++;
string hashcode = "", nonce_hashcode = frame_read(Current_frame,
"nonce_and_hash");
for (int i = 0; i < 7; i++)
hashcode += nonce_hashcode[i + 1];
nonce[0] = Convert.ToByte(nonce_hashcode[0]);
type_payload[0] = Current_frame[0];
for (int i = 1; i < 5; i++)
sender[i - 1] = Current_frame[i];
//----------------------------------------------------//
byte[] hash = new byte[7];
string H = h(1, Encoding.ASCII.GetString(type_payload) +
Encoding.ASCII.GetString(sender) + Encoding.ASCII.GetString(nonce));
byte[] transform = Encoding.ASCII.GetBytes(H);
for (int i = 0; i < 7; i++)
hash[i] = transform[i];
if (hashcode == Encoding.ASCII.GetString(hash))//frame authenticated!
{
byte pl = Convert.ToByte(frame_read(Current_frame, "payload_length"));

//----------------------------------------------------//
double t1 = Convert.ToDouble(temperatue(0));
double m1 = Convert.ToDouble(GetPshysicalMemory());
random_value = ((double)(t1 * m1));
if (random_value == 0)
{
```

```
int i = 4;
}
byte[] Type_payload = new byte[1];//Convert.ToByte(0x08);
byte[] Sender = new byte[4];
byte[] Payload = new byte[pl];
byte[] Nonce = new byte[1];
byte[] Hash = new byte[7];
Nonce[0] = 1;
Type_payload[0] = (byte)(Current_frame[0] & (0xf7));//set type as '0' means reply
for (int i = 0; i < 4; i++)
Sender[i] = Convert.ToByte(IP.Split('.')[i]);
Nonce[0] = ++nonce[0];
//------------------------------------------//
double random = (random_value % (Math.Pow(2,pl*8)));
string true_int_number = random.ToString().Split('.')[0];
random=Convert.ToDouble(true_int_number);
Payload = NumberToOctet(random, pl);
//Payload = Encoding.ASCII.GetBytes("100"); //It is not a good way to use ASCII
string Hashcode = h(1, Encoding.ASCII.GetString(Type_payload) +
Encoding.ASCII.GetString(Sender) + Encoding.ASCII.GetString(Payload) +
Encoding.ASCII.GetString(Nonce));
byte[] Transform = Encoding.ASCII.GetBytes(Hashcode);
for (int i = 0; i < 7; i++)
Hash[i] = Transform[i];
//------------------------------------------------//
string plaintext = Encoding.ASCII.GetString(Nonce) +
Encoding.ASCII.GetString(Hash);
byte[] encrypted = EncryptStringToBytes(plaintext,
frame_read(Current_frame,"Crypto_type"));
//-----------------------------------------------//
byte[] send_buffer = new byte[5 + pl + encrypted.Length];
send_buffer[0] = Type_payload[0];
for (int i = 1; i < 5; i++)
send_buffer[i] = Sender[i - 1];
for (int i = 5; i < 5+pl; i++)
send_buffer[i] = Payload[i - 5];
for (int i = 5+pl; i < 5+pl+ encrypted.Length; i++)
send_buffer[i] = encrypted[i - (5 + pl)];
//----------------------------sending-------------------------------//
sending_end_point = new IPEndPoint(IPAddress.Parse(frame_read(Current_frame,
"sender")), 11000);
try
{
sending_socket.SendTo(send_buffer, sending_end_point);
}
catch (Exception send_exception)
{
Console.WriteLine(" Exception {0}", send_exception.Message);
}
}
}
else
{
p3.Suspend();
}}}
```

The Random Value, which is produced in Responder Thread, relies on the CPU temperature and the available physical memory at that moment. The codes related to finding these parameters are as follows:

```
string GetTemperature(byte t)
{
Double temper = 0;
decimal t1 = 0, t2 = 0, t3 = 0, t4 = 0;
//byte[] result = new byte[8];
decimal f_result = 0;
ManagementObjectSearcher searcher = new ManagementObjectSearcher(@"root\WMI",
"SELECT * FROM MSAcpi_ThermalZoneTemperature");
foreach (ManagementObject obj in searcher.Get())
{
temper = Convert.ToDouble(obj["CurrentTemperature"].ToString());
if (t != 0) temper = (temper - 2732) / 10.0;
else
{
temper *= m_CPUCounter.NextValue();
t1 = m_CPUCounter.NextSample().RawValue;
//t2 = m_CPUCounter.NextSample().SystemFrequency;
t3 = m_CPUCounter.NextSample().TimeStamp;
t4 = m_CPUCounter.NextSample().TimeStamp100nSec;
t1 = Convert.ToDecimal(Inverse(t1.ToString()));
t3 = Convert.ToDecimal(Inverse(t2.ToString()));
t4 = Convert.ToDecimal(Inverse(t3.ToString()));

f_result = t1 + t3 + t4 + (decimal)temper + ((decimal)temper * t1 * t2 * t3);

File2.Write(f_result + "\r\n");
}
}
return (f_result.ToString());
}



string GetPshysicalMemory()
{
//----------------memory available--------------//
System.Diagnostics.PerformanceCounter m_memoryCounter;

m_memoryCounter = new System.Diagnostics.PerformanceCounter();
m_memoryCounter.CategoryName = "Memory";
m_memoryCounter.CounterName = "Available MBytes";
//----------------------------------------------//
double totalMB = 0;
foreach (ManagementObject mObj in new ManagementObjectSearcher("select * from
Win32_PhysicalMemory").Get())
totalMB += Convert.ToDouble(mObj["Capacity"]);

return ((totalMB / 1024 / 1024) + (m_memoryCounter.NextValue())).ToString();
}
```

A.4 Timer

When the lifetime timer is expired then it is time to evaluate the collecting random numbers:

```
private void lifetime_Tick(object sender, EventArgs e)
        {
            lifetime.Enabled = false;
            Authentication_Collection();
            btn_cnt.Enabled = true;
        }
```

In addition, the list of available laptops in a Diffusion Network is updated based on a timer:

```
private void timer1_Tick(object sender, EventArgs e)
{
IPHostEntry myHostInfo = Dns.Resolve(Dns.GetHostName());
IP = myHostInfo.AddressList[0].ToString();
lblIP.Text = IP;
temp.Text = GetTemperature(1);
//----------------------------------------------------//
if (checkBox2.Checked == true)

              net_view();

}
```

A.5 Authentication Reading Collection

In Authentication Collection function, after decrypting and checking the authentication of the reply frames the archived random numbers used as an input for Scramble function:

```
void Authentication_Collection()
```

```
{
listBox1.Items.Add("LifeTime Timer is expired");
listBox1.Items.Add("Collecting Random numbers");
int reply_number = 0;
byte[] ADCs = new byte[Math.Abs(reply_rear - reply_front) *
Convert.ToByte(txt_pl.Text)];
while (reply_front != reply_rear)
{
byte[] Current_frame = new byte[reply_queue[(reply_front) % 10000].Length];
Current_frame = reply_queue[(reply_front) % 10000];
reply_front++;
byte pl = (byte)(Current_frame[0] & (0x07));
byte[] type_payload = new byte[1];//Convert.ToByte(0x08);
byte[] sender = new byte[4];
byte[] payload = new byte[pl];
byte[] nonce = new byte[1];
byte[] hash = new byte[7];
//----------------------------------------------------------------------
string hashcode = "", nonce_hashcode = ""; //= frame_read(Current_frame,
"nonce_and_hash");
string[] C_type = new string[5];
C_type[0] = "AES";
C_type[2] = "Triple DES";
//------------------------------Decryption based on AES or Triple DES

byte[] decrypt = new byte[Current_frame.Length - (5 + pl)];
for (int i = 0; i < 16; i++)
decrypt[i] = Current_frame[5 + pl + i];
nonce_hashcode = DecryptStringFromBytes(decrypt, C_type[((Current_frame[0] &
(0xf0)) >> 4) - 1]);
//------------------------------------------------------Authentication
for (int i = 0; i < 7; i++)
hashcode += nonce_hashcode[i + 1];
nonce[0] = Convert.ToByte(nonce_hashcode[0]);
type_payload[0] = Current_frame[0];
for (int i = 1; i < 5; i++)
sender[i - 1] = Current_frame[i];
for (int i = 5; i < 5 + pl; i++)
payload[i - 5] = Current_frame[i];
//byte[] hash = new byte[7];
string H = h(1, Encoding.ASCII.GetString(type_payload) +
Encoding.ASCII.GetString(sender) + Encoding.ASCII.GetString(payload) +
Encoding.ASCII.GetString(nonce));
byte[] transform = Encoding.ASCII.GetBytes(H);
for (int i = 0; i < 7; i++)
hash[i] = transform[i];
if (hashcode == Encoding.ASCII.GetString(hash))//frame authenticated!
{
listBox1.Items.Add(sender[0] + "." + sender[1] + "." + sender[2] + "." +
sender[3] + "    " + OctetToNumber(payload));
if (OctetToNumber(payload) == 0)
{
MessageBox.Show("");
}
//-----------------------------------------------collecting numbers//
for (int i = 0; i < pl; i++)
{
ADCs[(reply_number * pl) + i] = payload[i];
}
reply_number++;
replyN.Text = reply_number.ToString();
}
}

//----------------------------------Scramble Function //
decimal TrueRND=OctetToDecimal(scramble(ADCs));
string StringRND=TrueRND.ToString();

string InvRND = TrueRND.ToString();

decimal T_RND = Convert.ToDecimal(InvRND);
T_RND = (T_RND / MAX);
TRND.Text = T_RND.ToString();
listBox1.Items.Add("Generated True Random Number:");
listBox1.Items.Add(T_RND.ToString());

hs.Text = InvRND;
hs2.Text = StringRND;

try
{
count++;
}
catch (Exception e)
{
}
avrage += (ulong)reply_number;
lcount.Text = count.ToString();
listBox1.Items.Add("RESULTS SAVED IN A FILE");
}

private string Inverse(string InvRND)
{
string st = "";
for (byte i = 0; i <InvRND.Length; i++) // Inverse
st += InvRND[InvRND.Length - i-1];
return st;
}
```

For decryption a frame, the first four bit should be achieved to determine the type of decryption, AES or Triple DES. Decryption function code is explained bellow:

```
static string DecryptStringFromBytes(byte[] cipherText,string Decr_type)
{
string plaintext = null;
// Create an TripleDESCryptoServiceProvider object
// with the specified key and IV.
switch (Decr_type)
{
//--------------------------------------------//
case "AES":
```

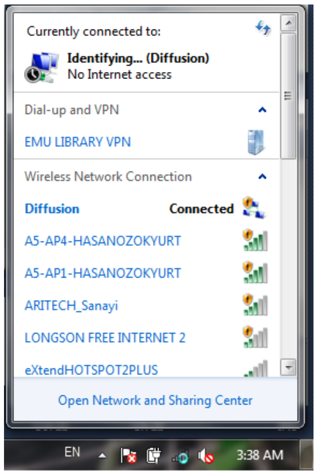
```
using (AesCryptoServiceProvider tdsAlg = new AesCryptoServiceProvider())
{
tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess_the_Key!?");
tdsAlg.IV = Encoding.ASCII.GetBytes("towish_is_toable");
tdsAlg.Mode = CipherMode.CBC;
// Create a decrytor to perform the stream transform.
ICryptoTransform decryptor = tdsAlg.CreateDecryptor(tdsAlg.Key, tdsAlg.IV);
// Create the streams used for decryption.
using (MemoryStream msDecrypt = new MemoryStream(cipherText))
{
using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
{
using (StreamReader srDecrypt = new StreamReader(csDecrypt))
{
// Read the decrypted bytes from the decrypting stream
// and place them in a string.
plaintext = srDecrypt.ReadToEnd();
}
}
}
}
return plaintext;
//-------------------------------------------------//
case "Triple DES":
using (TripleDESCryptoServiceProvider tdsAlg = new
TripleDESCryptoServiceProvider())
{
tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess_the_Key!?");
tdsAlg.IV = Encoding.ASCII.GetBytes("big_bang");
tdsAlg.Mode = CipherMode.CBC;
ICryptoTransform decryptor = tdsAlg.CreateDecryptor(tdsAlg.Key, tdsAlg.IV);
using (MemoryStream msDecrypt = new MemoryStream(cipherText))
{
using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
{
using (StreamReader srDecrypt = new StreamReader(csDecrypt))
{
plaintext = srDecrypt.ReadToEnd();
}
}
}
}
return plaintext;
//-----------------------//
default:
return plaintext = "Decryption Error";
}
}
```

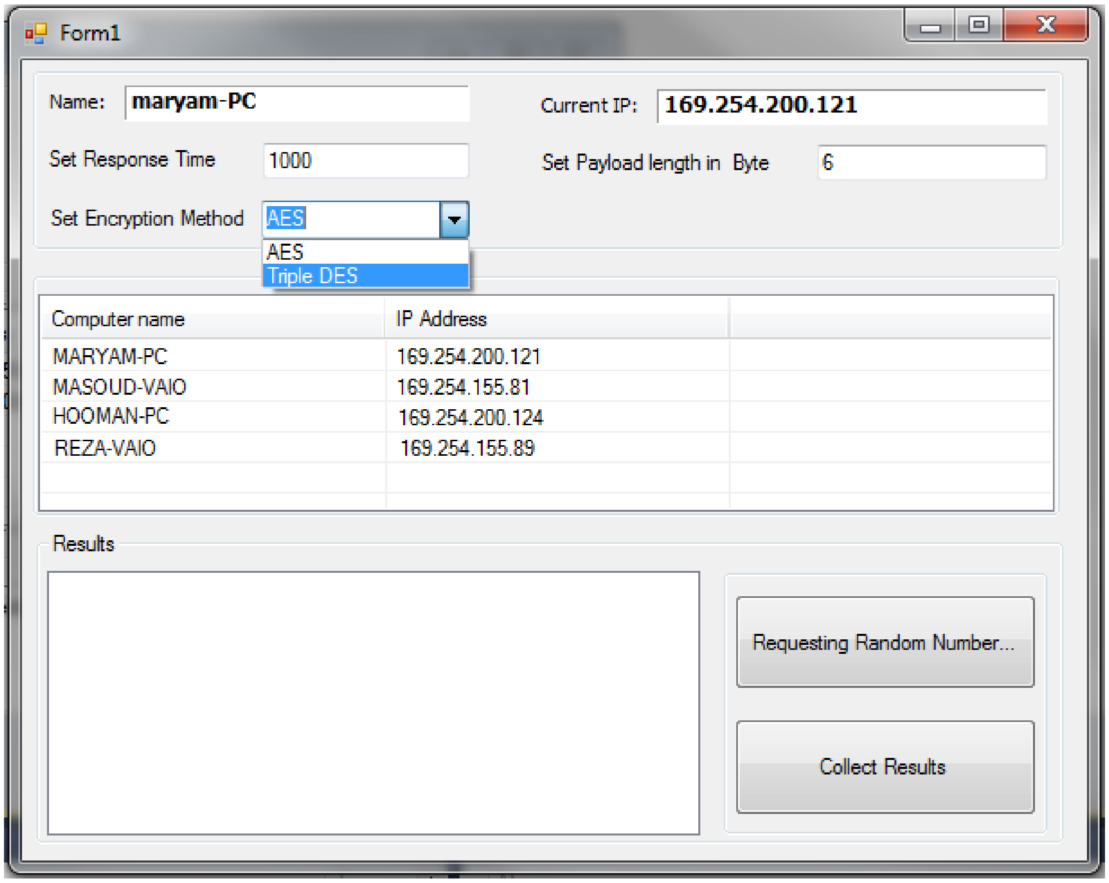
The codes related to the scramble function that is used hash function and CMAC are as follows:

```
private byte[] scramble(byte[] ADCs)
{
byte[] key = new byte[8];
byte[] LC = new byte[2];
byte[] TrueRND = new byte[8];
byte[] TempRND = new byte[8];
byte[] hash = new byte[16];
string sh = h(1, Encoding.ASCII.GetString(ADCs));
hs.Text = sh;
hash = FromHexToOctet(sh);
for (byte i = 0; i < 16; i++)
{
BB[i] = BB[i + 16];
BB[i + 16] = hash[i];
}

string temp = DateTime.Now.ToString("ffffff");
LC = NumberToOctet(Convert.ToDouble(temp), 2);
key[7] = (byte)(RB[7] ^ LC[1]);
key[6] = (byte)(RB[6] ^ LC[0]);

TempRND = CMAC(BB, key);
for (byte i = 0; i < 8; i++)
{
TrueRND[i] = (byte)(TempRND[i] ^ RB[i]);
File.Write(NumberToBinary(TrueRND[i]));
RB[i] = TrueRND[i];
}
return TrueRND;
}

private byte[] CMAC(byte[] BB, byte[] key)
{
// byte[] keyBytes = new byte[16];
double k2 = OctetToNumber(RB);
//string temp=k2.ToString()
var keyBytes= NumberToOctet(Convert.ToDouble(OctetToNumber(key).ToString() +
(k2.ToString())),16);
var mac = new MACTripleDES(keyBytes);
var macResult = mac.ComputeHash(BB);
return macResult;
}
```

Appendix B: User Guide

Upon the application is run, Diffusion Ad-hoc WLAN is established and added to the list of available wireless networks. Moreover, main window is opened and the computer Name and IP, which is set in a proper text box, is presented. As the figure bellow shows, it is possible to set some parameters such as the expected response time, payload length and the desired encryption method. Actually, a list of all reachable laptops that are connected to Diffusion network is represented in the window; these laptops are distinguished by their IP address.


Currently connected to:
Identifying... (Diffusion)
No Internet access
Dial-up and VPN
EMU LIBRARY VPN
Wireless Network Connection
Diffusion Connected
A5-AP4-HASANOZOKYURT
A5-AP1-HASANOZOKYURT
ARITECH_Sanayi
LONGSON FREE INTERNET 2
eXtendHOTSPOT2PLUS
Open Network and Sharing Center
EN 3:38 AM

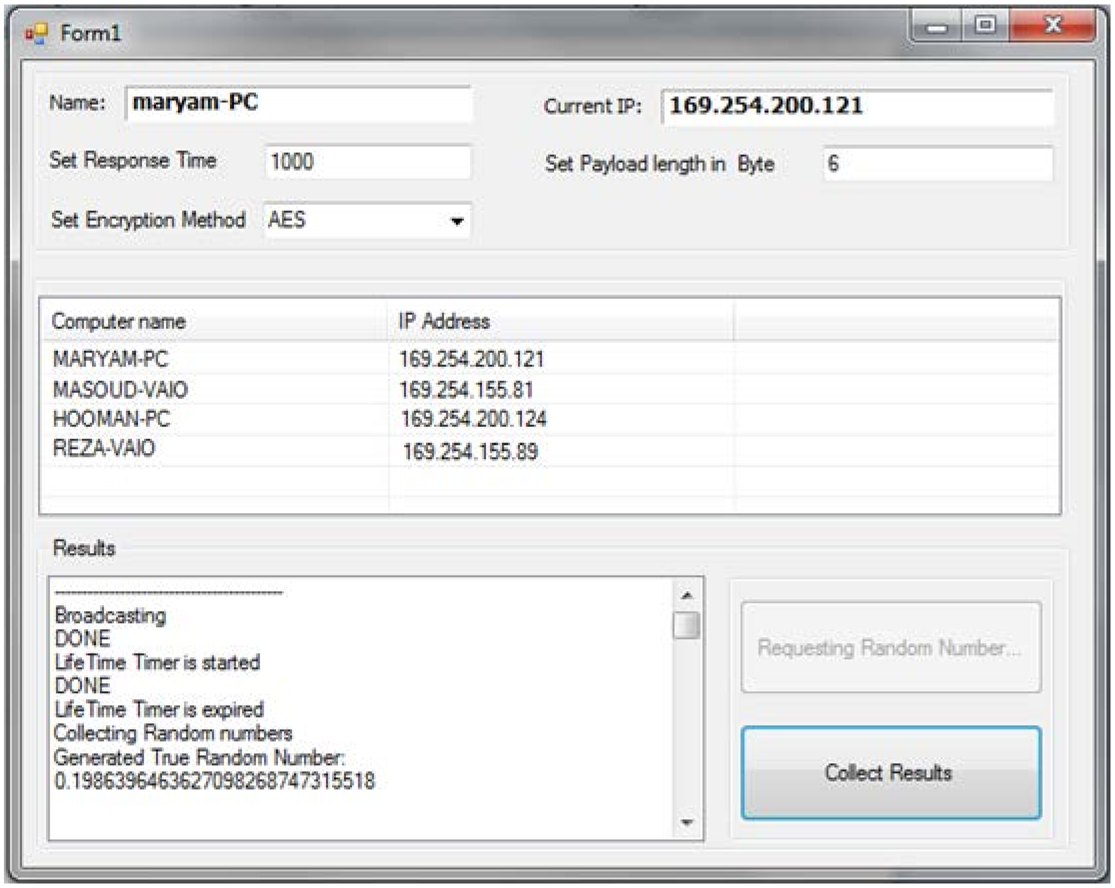

Form1

Name: maryam-PC
Current IP: 169.254.200.121
Set Response Time 1000
Set Payload length in Byte 6
Set Encryption Method AES
AES
Triple DES

Computer name	IP Address
MARYAM-PC	169.254.200.121
MASOUD-VAIO	169.254.155.81
HOOMAN-PC	169.254.200.124
REZA-VAIO	169.254.155.89

Results
Requesting Random Number...
Collect Results

After clicking on Requesting Random Number Button,The request sends to all laptops in a list and the process of collecting and generating true random number is shown in Result box:

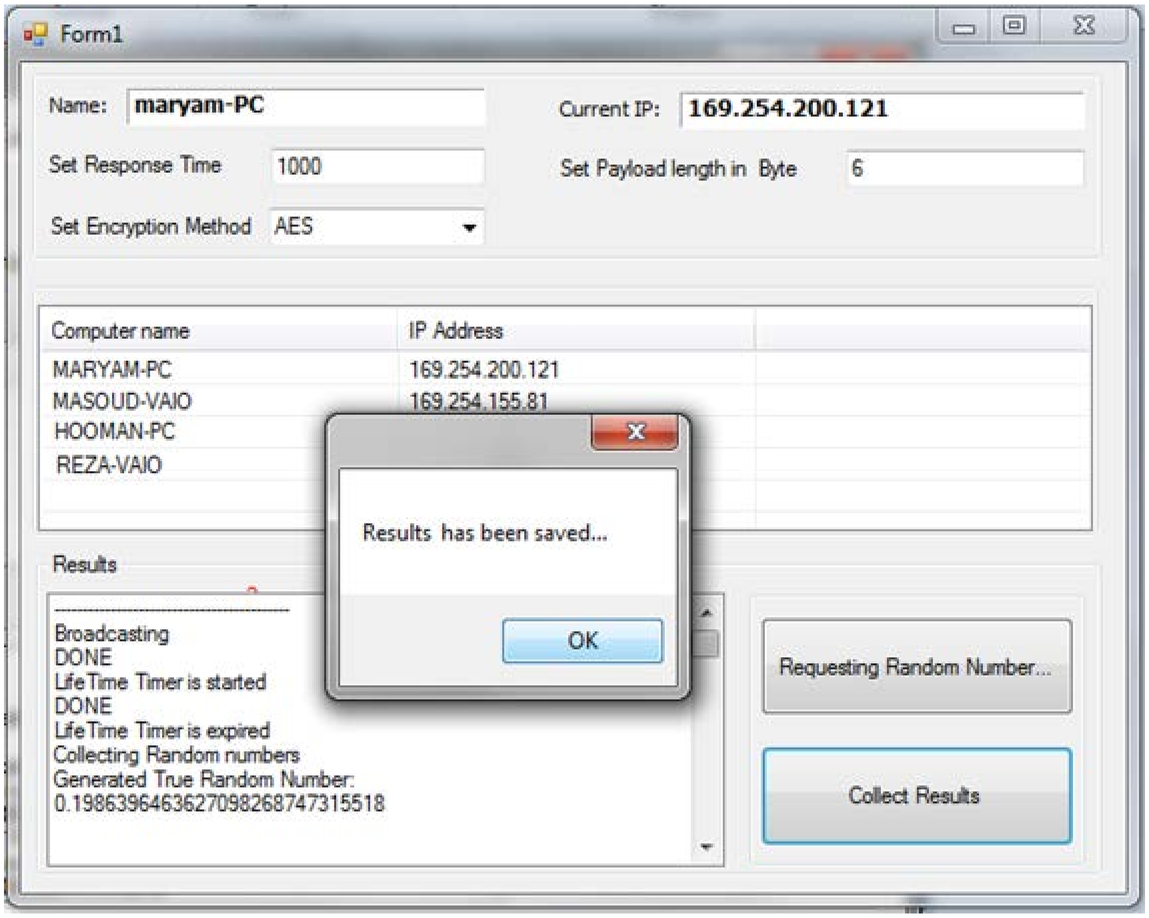

Form1
Name: maryam-PC
Current IP: 169.254.200.121
Set Response Time 1000
Set Payload length in Byte 6
Set Encryption Method AES
Results
Requesting Random Number...
Collect Results

Computer name	IP Address
MARYAM-PC	169.254.200.121
MASOUD-VAIO	169.254.155.81
HOOMAN-PC	169.254.200.124
REZA-VAIO	169.254.155.89

```
----------------------------------------
Broadcasting
DONE
LifeTime Timer is started
DONE
LifeTime Timer is expired
Collecting Random numbers
Generated True Random Number:
0.1986396463627098268747315518
```

As the results in the box shows, the process is done successfully. However, if any error in encryption or decryption and authentication happens, it will illustrate in the box.

In the End, by clicking on Collect Results Button all generated true random numbers save in a file named Results in Byte:


Form1

Name: maryam-PC

Current IP: 169.254.200.121

Set Response Time 1000

Set Payload length in Byte 6

Set Encryption Method AES

Computer name	IP Address
MARYAM-PC	169.254.200.121
MASOUD-VAIO	169.254.155.81
HOOMAN-PC	
REZA-VAIO	

Results

```
-------------------------------------------
Broadcasting
DONE
LifeTime Timer is started
DONE
LifeTime Timer is expired
Collecting Random numbers
Generated True Random Number:
0.19863964636270982687473155 18
```

Results has been saved...