Secure True Random Number Generator in a Distributed System via Wireless LAN (Distributed-RNG protocol)

Hooman Alavizadeh

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
June 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 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 Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov

Examining Committee

1. Assoc. Prof. Dr. Alexander Chefranov
2. Asst. Prof. Dr. Gürcü Öz
3. Asst. Prof. Dr. Önsen Toygar

ABSTRACT

In computer network and cryptography era, the necessity of generating true random numbers (TRNG), in the security directions and cryptography algorithms which require random numbers as nonce, public keys, private keys, session keys, secret keys, seeds, salts and etc. is inevitable. Cryptography algorithms are widely used in various networks; among them, Wireless networks need more security in comparison to the other networks due to their intrinsic vulnerabilities against possible attacks. In the most of recently studies, it is proven that an acceptable random number will not be generated unless using a distributed method for random number generator. This thesis firstly analyses a protocol for generation of secure true random number (Scatter) which used distributed method via a wireless network. Secondly, after making some changes on the mentioned protocol in regard to protocol structure and entropy of randomness source, the enhanced Distributed-RNG protocol is introduced; it provides more security and flexibility. Then, the quality of randomness of obtained random numbers using National Institute of Standards and Technology (NIST) tests is evaluated. Finally, after analyzing the results of both studies, it can be conducted that 60 percent of obtained results in this thesis are better than the existing protocol (Scatter).

Keywords: Cryptography, True Random Numbers Generator (TRNG), Network Security, Nonce, Key Generations

ÖZ

Bilgisayar ağları ve kriptografi çağında, güvenlik konusunda ve kriptografi algoritmalarında gerçek rasgele numaralar yaratmak kaçınılmazdır. Kriptografi algoritmaları değişik ağlarda kullanılmaktadır. Bu ağlar arasında bulunan kablosuz ağlar, kablolu ağlara kıyasla daha fazla güvenlik gerektirmektedirler.

Bu tezde, önce gerçek rasgele numaralar yaratma protokolu (Scatter protokolu) incelenmiştir. Daha sonra, yapısında ve rasgele numaralar kaynağı entropisinde bazı değişiklikler yapılmıştır. Geliştirilmiş olan protokol (Distributed-RNG) güvenlik ve esneklik sağlamaktadır.

Rasgele numaraların kalitesi Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) testleri tarafından değerlendirilmiştir. En sonunda iki algoritma karşılaştırıldığında, bu tezde önerilen algoritmanın sonuçları scatter algoritmalarından %60 daha iyi olduğu gözlemlenmiştir.

Anahtar Kelimeler: Kriptografi, Gerçek Rasgele Numaralar Yaratma (TRNG), Ağ güvenliği, NIST Testi

Dedicated to my family with love

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and appreciation to my supervisor, Dr. Alexander Chefranov, for his patient, guidance and encouragement throughout this study. His knowledge and experience have been prominent help for my work. I wish to express my thanks to all members of the department.

Last, but not least, I am greatly indebted to my mother for her invaluable supports and encouragement. I also owe special gratitude to my wife for her constant encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RNG	: Random Number Generator
TRNG	: True Random Number Generator
CPRNG	: Cryptographic Pseudo Random Number Generator
NIST	: Introduction of National Institute of Standards and Technology
RB	: Register Block
BB	: Buffer Block
RBS	: Random Bit Sequence
CBC	: Cipher Block Chaining
OFB	: Output Feedback
CFB	: Cipher Feedback
DES	: Data Encryption Standard
WLAN	: Wireless Local Area Network
MAC	: Message Authentication Mack
CMAC	: Cipher Based Mack

Chapter 1

INTRODUCTION

Nowadays, the world is becoming more interconnected with the advent of the Internet and new networking technology. There is a huge amount of personal, commercial, military, and government formation on networking infrastructures in all over the world. Among them, the role of network and data security is becoming more prominent because of the existence of the variety of attacks through the networks and the internet violating confidentiality, integrity, availability, authenticity and finally leads to disastrous. Cryptography is known as a powerful tool for avoiding the mentioned challenges; it uses mathematics to encrypt and decrypt sensitive data and they can be securely transmitted across insecure network (like the LAN, WAN and specially WLAN). On the other hand, cryptanalysis is the science of attempting for breaking secure communication. However, cryptography methods must continue to be developed because of the development of attackers since "Strong cryptography in the past can be easily broken today" [1, 2].

This study modifies a protocol in a WSN to generate secure TRNG and introduces an enhanced protocol which is more flexible and secure than the proposed one.

A wireless local area network (WLAN) is a network which links their devices using wireless distributions, such as spread-spectrum and OFDM radio. Despite its mobility in that users can move around under coverage area, it has inevitable disadvantage, wireless networks need more security than the other type of networks. Hackers and attackers can easily intercept wireless network traffic over air connection and eavesdropping can be done more convenient.

Consequently, the necessity of using powerful cryptography algorithm for such a network is considerable. Nevertheless, the imagination of having well-designed cryptography algorithm without using random number for security reasons of cryptography is not possible.

In Chapter 2, current approaches of random number generators (RNGs) are considered, then, necessary algorithms and also testing method for evaluating the quality of number randomness are surveyed. First of all, random numbers may be used in many different aspects like nonces, key generations (ex. Public keys, private keys, session keys, and secret keys), seeds, salts, padding and etc. There are two main kinds of generators applied to generate random numbers, the former is true random number generators and the latter is pseudorandom number generators; whereas, most of computers lack hardware based random number generators (RNG), software based technique can be used for generating random numbers known as pseudorandom number generators, they are very scarcely truly random, secondly, to generate true random data you need a source of entropy, or random input such as mouse movement, keyboard keys, hard drive activities, network activities, sensors, some CPU and memory criteria and etc. In the case of true random number there is an already study proposed in [3] as "Secure Random Number Generation in Wireless Sensor Networks" which introduces a protocol (Scatter protocol) using wireless sensor network for generating true random values, in which this protocol (scatter) is the base of this thesis. After some changes in scatter protocol the enhanced protocol will be introduced, for providing more flexibility and security. The general structure of scatter protocol is shown in Figure 1.

Type 1 bit	Payload length 3 bits	Sender 12 bits	Payload 48 bits	Nonce 8 bits	Hashcode 56 bits

Figure 2: Message Structure of Scatter Protocol

The node wishing random values initializes the frame and broadcast it through sensors nodes and receives a frame, which both nonce and hashcode fields are encrypted for providing integrity, however, the payload field which is responsible of taking random values are not encrypted and this may compromise the security. This problem is solved in enhanced protocol by encrypting this field. Next, in this structure there is no field that specifies the encryption method, thus, the receiver should know the encryption algorithm before, and this problem is solved in enhanced protocol by adding a new field for providing more flexibility of protocol. The details of both protocols will be explained later.

On the other hand, both of the mentioned generators produce a stream of zeros and ones knows as random bits sequence. The sequence of generated random numbers must be random enough to be applicable for cryptography, and it should satisfy two important criteria: Randomness and Unpredictability [4]. The definition of the former is that a random bit sequence can be supposed as the result of throwing an unbiased fair coin in which each side of the coin is labeled as zero or one, and the probability of coming each side after throwing is the same (½), moreover, the throwing processes are independent events, it means that the result of any previous throwing coin does not affect the next throwing. The unbiased coin is the appropriate random bit stream generator, because zeroes and ones in the sequence are uniformly distributed in [0, 1] [4]. The latter is also defined [4] as random number generators used in cryptographic applications that have to be completely unpredictable. For pseudo random number generators, by choosing an unknown seed, the next generated random sequence has to be unpredictable. In addition, determining the seed also should not be feasible from knowledge of any generated values. Moreover, there should not be any correlation between a seed and any generated value from that seed. Details and current approaches for testing the quality of randomness are described in Chapter 2.

In Chapter 3, the details of Distributed-RNG protocol is described. The protocol is a bit oriented protocol, the topology of organized network is mesh protocol, and it is supposed that each party can have direct or indirect neighbors. There are two main parts in protocol: requesting for a new random number (by requester node) – replying the requests (by receiver nodes). A requester sends a message to all reachable nodes in order to get a random number. The message consists of seven parts each of which has its own bit size. One of the differences of this protocol with the scatter protocol is in Crypto field added in the earlier bits of message, see Figure 2.

Crypto.	Type	Payload length	Sender IP	Payload	Nonce	Hashcode
4 bits	1 bit	3 bits	32 bits	48 bits	8 bits	56 bits

Figure 3: Message Structure of Distributed-RNG Protocol

Using MD5 hash function, the requester digests these fields and takes the least significant 56 bits of digest value as hashcode. Thus, it encrypts nonce and hashcode before sending for providing authentication and integrity. Two algorithms are provided for encryption DES and RC2. Next, the receiver applies necessary steps for authenticating the message, then; receiver decodes the fields of message and produces a true random sequence using CPU temperature and information of current active threads as entropy resource. Then, receiver fills the message again, random value as payload, and applies the security affair on them and then send it back to requester. Then, the requester receives sequences and joins them and using a scrambling function it finally produces 64 bits as final True random number, the final value is fed in payload. Finally payload, nonce and hashcode will be encrypted and return back to the requester. The encryption of payload together with the two last fields is the second difference of this protocol in comparison to scatter protocol which provides more security. More details will be explained in the fourth Chapter.

The achieved random sequence is tested in Chapter 4 by NIST tests [4] which apply various statistical tests on the sequences to evaluate the quality of number’s randomness. Then, the result of this protocol will be gathered and surveyed; finally, these results are compared with the result obtained by proposed protocol (“scatter protocol”).

In Chapter 5, the result is conducted and compared with the existing protocol. Both protocols first gather the same random bits sequence and perform NIST tests on their results. Then, the results obtained by these generators are tableted and completely discussed. Finally it is shown that better result is obtained in most of the tests of NIST in this study in comparison to the scatter protocol.

Chapter 2

RELATED WORK AND PROBLEM DEFINITION

The construction of RNGs has been widely discussed in literature; some of them work on pseudo-random number generators and the other group survey true-random number generators. This thesis analyses a known work discussed in the second group [3] and then introduces the enhanced protocol which is more reliable than the existing protocol. This chapter conveys the current approaches of true random number generators (TRNGs) and used materials and testing methods.

2.1 PRNGs Approaches

Pseudorandom number generators (PRNGs) are known as a deterministic random bit generators, they pursue an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The generated sequence is not truly random because it is absolutely obtained by a relatively small set of initial values, called PRNG's state, which include random seed. Sometimes they use procedural generation and linear congruential algorithms.

Nevertheless, cryptographic applications need unpredictable random sequences, and more elaborate designs which have not done according to linearity of simpler solutions. In spite of their randomness weakness, pseudorandom numbers are important practically for their speed in number generation and their reproducibility [5]. Other instances of PRNGs are Blum Blum Shub [6], Fortuna [7], and Mersenne Twister [8] algorithms.

However, generated sequences by pseudo-random number generator are not random enough and they are somewhat deterministic, so they are not definitely random, they have some guessable structure. Strong generators vary from bad generators in regard to having better hidden structure; i.e. in [9] the author defines a structure that is too difficult to detect by statistical tests and it seems to be true.

As a result, a good pseudo-random number generator should be difficult to analyze theoretically. Therefore, although, lots of efforts have been done by mathematicians to analyze complicated RNGs, fundamental issue between mathematical tractability and randomness has not been solved yet. Thus, it is necessary to find better random number generators which can be beneficial for the use of cryptographic algorithms and applications.

2.2 TRNGs approaches

True random number generators (TRNGs) are known as non-deterministic generators they use entropy source and physical quantity, for instance, the generator proposed by [10] defines PRNG as “utilize a pseudo random generator (PRNG). PRNGs use deterministic processes to generate a series of outputs from an initial seed state. Since the output is purely a function of the seed data, the actual entropy of the output can never exceed the entropy of the seed. It can, however, be computationally infeasible to distinguish a good PRNG from a perfect RNG.” it also uses electronically measurable characteristic shot noise and noise which are present in all resistors. Figure 3 illustrates general diagram of Intel TRNG generator.

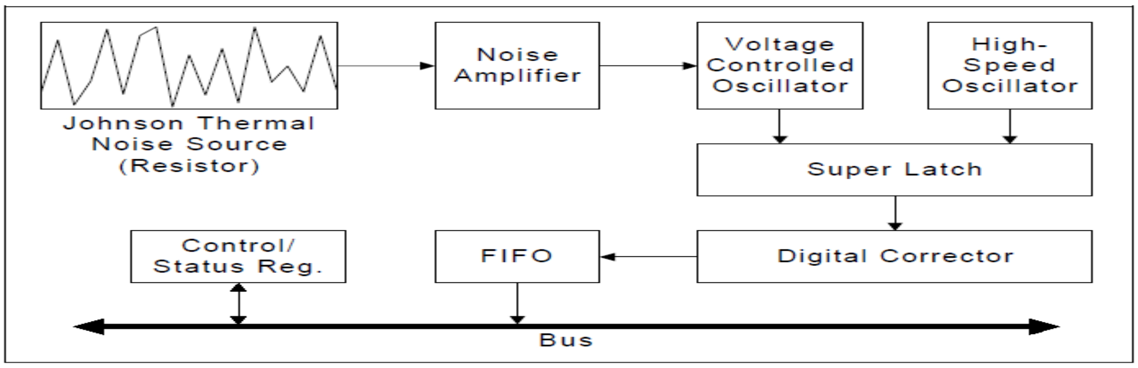


Figure 4: General Diagram of Intel TRNG Generator [10]

In [11] the authors use the rotation speed of hard disk sectors as unpredictable resource; it applies the response time of access and reading from hard disk sector as a measurement source. The study proposed by [11, 12] makes the sequence of random bits using the intrinsic features of quantum mechanics, it measures the arrival time of the photon emitted by light rays Figure 4.

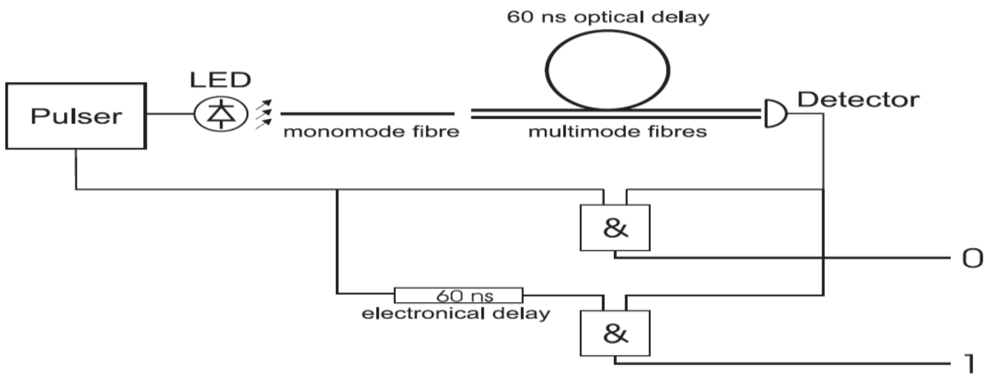


Figure 5: Schematic Diagram of the True Random Number Generator [11]

TRNGs significantly grow for Wireless Sensor Networks they apply the sensor component or the transceiver component (Strength Signal Indicator (RSSI) and transmission power function) as a source of sequence generations [13, 14]. It also conveyed in [15] as “A Cryptographic Random Number Generator for Wireless Sensors Network Nodes” they present tiny-RNG, Cryptographic Pseudo-Random Number Generator (CPRNG) for wireless sensor nodes, they assume that “transmission bit errors on a wireless sensor network are a very good source of randomness” and in their generation they use received bit errors for a source of randomness. Then, they illustrate that their result are randomly distributed and they are uncorrelated from one sensor to another and also they demonstrate that their result are secure and cannot be observed and manipulated by attackers. Figure 5 illustrates the general block diagram.

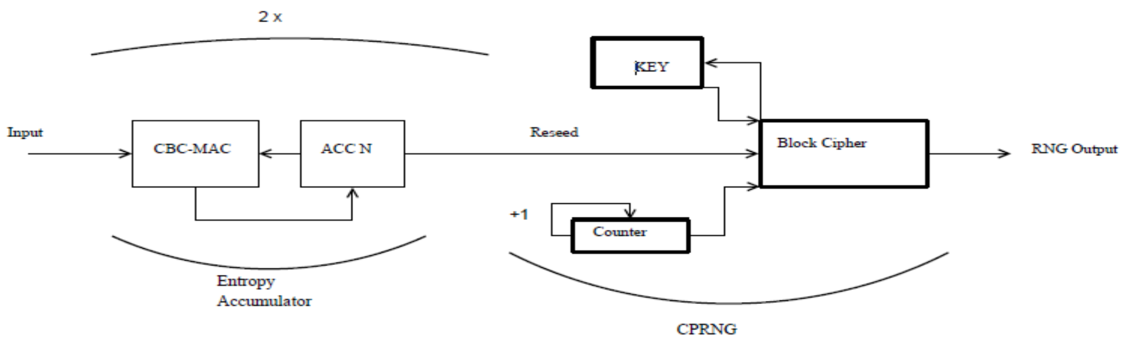


Figure 6: Tiny-RNG Block Diagram [15]

Its accumulator consisting of CBC-MAC function is proven in [5] as good random extractor and it minimizes the memory requirement by applying the same block cipher for CPRNG, after gathering enough entropy the accumulator reseeds the key of the CPRNG. The CPRNG block encrypts a counter by using a key and “The output of this

CPRNG (i.e. the encrypted-counter) is then available to applications through the standard Tiny-OS Random interface”.

In [16] the author introduces Tiny-Sec as “the first fully-implemented link layer security architecture for wireless sensor networks”, and it has three main goals: security, performance, and usability, finally, it concludes “Tiny-Sec addresses security in devices where energy and computation power present significant resource limitations. We have designed Tiny-Sec to address these deficiencies using the lessons we have learned from other security protocols. We have tried to highlight our design process from a cryptographic perspective that meets both the intended resource constraints and security requirements. Tiny-Sec relies on cryptographic primitives that have been vetted in the security community for many years.” [16]

Unlike LAN, wireless network can neglect the physical restriction of wired communication and it can provide flexibility, boost employee productivity and low cost. Figures 6 compares both traditional wired 10/100 base Ethernet LAN with a wireless LAN [16], the former has lots of installation cost and requires several days for install, and its arrangement incur additional cost and lots of challenge in physical area, while the latter, wireless LAN are cheaper and it can be implemented and maintained better and more convenient. In spite of the advantages of WLANs, these networks shows that they are more vulnerable than the LANs and it leads to attention of lots of security cryptography studies to WLAN [17,18,19,20,21].

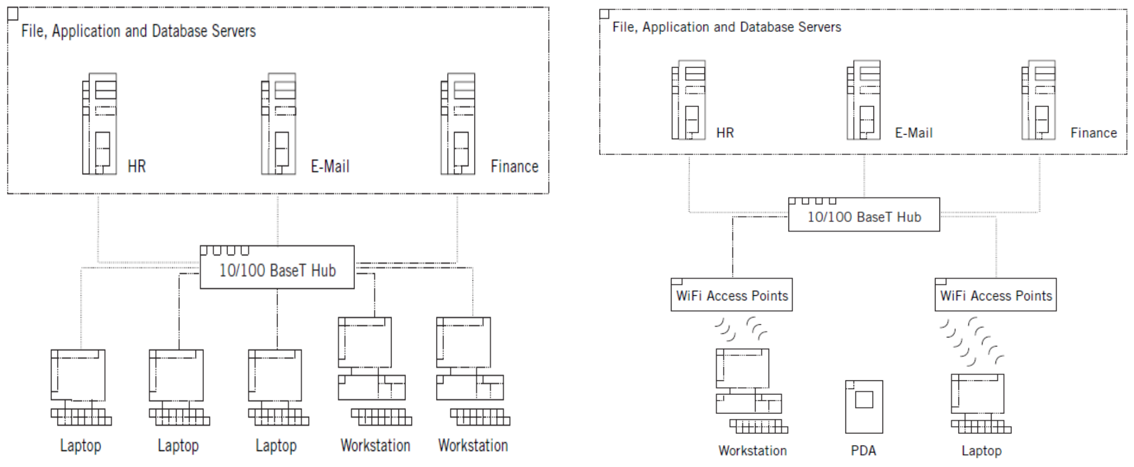


Figure 7: Traditional Wired 10/100 Base Ethernet LAN and Typical Wireless LAN [13]

2.2.1 Scatter Light TRNG Protocol

This section considers a currently proposed study [3] “Secure Random Number Generation in Wireless Sensor Networks”. Then, a protocol will be defined to enhance Scatter protocol in Chapter 4.

The goal of the mentioned literature is to collect sufficient values from environment and generate a high quality random number. It uses WSN (wireless sensor network) for implementation of protocol and also defines a framework providing security of data transmitted through the network. There are two kinds of nodes, requesters and repliers. The former nodes send a request to repliers in order to receive random values and latter nodes use entropy to return the desired values. The author is persuaded that a wireless sensor could be appropriate entropy for taking some natural values; thus, wireless nodes use sensors as a physical phenomenon, these values obtained by each nodes of network securely reach requester node. Then, collected values will be feed in a TRNG module and random number will be produced. For addressing these stages, a protocol is defined, the protocol constructs a message containing six fields in order to transmit the data, and these six fields are “type, payload length, sender, payload, nonce and hashcode”. Each of which has respectively 1 bit, 3 bits, 12 bits, 48 bits, 8 bit, 54 bits length.

The type of message is defined in type (request or reply), the desired length of random values is determined by payload length, and sender is the identifier of the nodes sending the message. The result of request returns back by payload, nonce and hash code are included in order to provide security. SHA-1 hash function is used as message digest of all fields for maintaining integrity and DES algorithm is used in order to encryption data (nonce and hashcode). In the constructed message nonce and hashcode had been encrypted before transmitting. As mentioned before, collected values are fed in a module and generated random number will be obtained as shown in Figure 7.

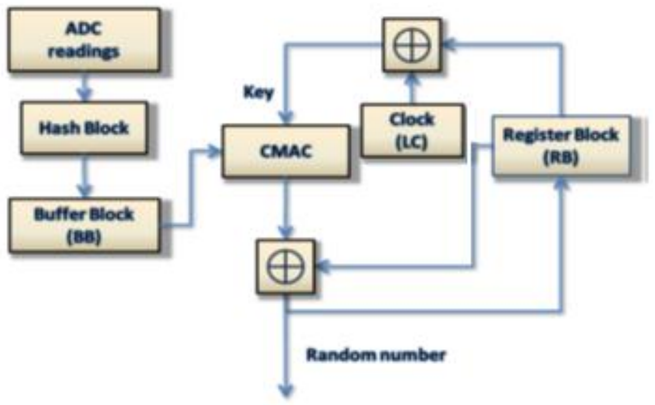


Figure 8: Block Diagram of TRNG Module [3]

Finally, the quality of obtained result is evaluated by NIST tests which will be discussed in the next section. Despite the advantages of the protocol, it has some unpleasant disadvantages. First of all, the encryption method is not specified in the message construction, it leads to lack of flexibility, second, the message containing results will not encrypt the payload which is the most important field of the message due to having valuable value, and it obviously violet the confidentiality (receiver doesn't know whether the received value is confident or not). Although the intruder can't modify the message, the Eavesdropper can obtain the payload value; as a result, the security may be compromised. In order to overcome these disadvantages, this study tries to enhance the capability of the protocol by making some changes in message structure and used algorithm. In the enhanced model, which will be introduced in Chapter 3, as Distributed –RNG protocol, firstly, the newly added field, Crypto, uses in message message which can show the receiver which kind of encryption algorithm is used to encrypt the message and also it indicates the mode of operation used for encryption, it certainly increases the flexibility of the protocol and its aftermath leads to more security. Secondly, as a return message, payload is encrypted together with nonce and hashcode to avoid Eavesdropping and its aftermath leads to message confidentiality. Then, MD5 hash function is chosen instead of SHA-1 for calculating hashcode and message digests mater because of cheaper computational complexity leading to consume lower battery. More details of explanation of the previous protocol and its changes in order to introduce enhanced one will be discussed in Chapter 3.

2.3 Survey of some Cryptography Algorithms

This study uses various known algorithms and functions having been recently developed and published. In this section, some of the existent and useful function and algorithms applied in development of this thesis have been briefly discussed, which are MD5 message digest function, DES encryption algorithm, RC2 algorithm and Triple-DES MAC function.

2.3.1 MD5 Hash Function

MD5 is known as one of the quickest message digest algorithm; this function takes a message with variable length and the 128 bits message digest is produced. This function is quite faster than the other message digest algorithms, because of lack of very large substitution tables. However, it is shown that MD5 can be vulnerable and can be compromised [22]; thus, it should not use in widely protocols such as SSL, SET, ESET and etc. which needs high level of security. Nevertheless, for RNGs purpose it is an appropriate algorithm yet, due to being slightly cheaper to compute and especially in wireless networks which battery usage is one of the biggest concerns of them. Hence, in the Distributed-RNG protocol discussed in Chapter 3 MD5 algorithm is used instead of SHA-1 used in previous proposed one (Scatter).

2.3.2 DES Encryption Algorithm

DES algorithm [23] published in the early 1970s is an encryption algorithm which takes 64 bits block as input and using 64 bits key (actually 56 bits are used) generates 64 bits cipher text during 16 main rounds. Each round uses a Feistel (F) function consisting of four stages:

1. Expansion: the most significant32-bit is expanded to 48 bits using E-table.
2. Key mixing: the XOR of the result of previous stage and 48 bits sub key is calculated and then its 48 bits result is feed as input of next stage.

3. Substitution: the result of XOR operation, will be feed in substitution choice module (S-box) which known as the heart of DES security. This stage produce 32 bits output.
4. Permutation: finally, a permutation of the 32 bits output of previous stage will be arranged.

The overall structure of DES is depicted in Figure 8.

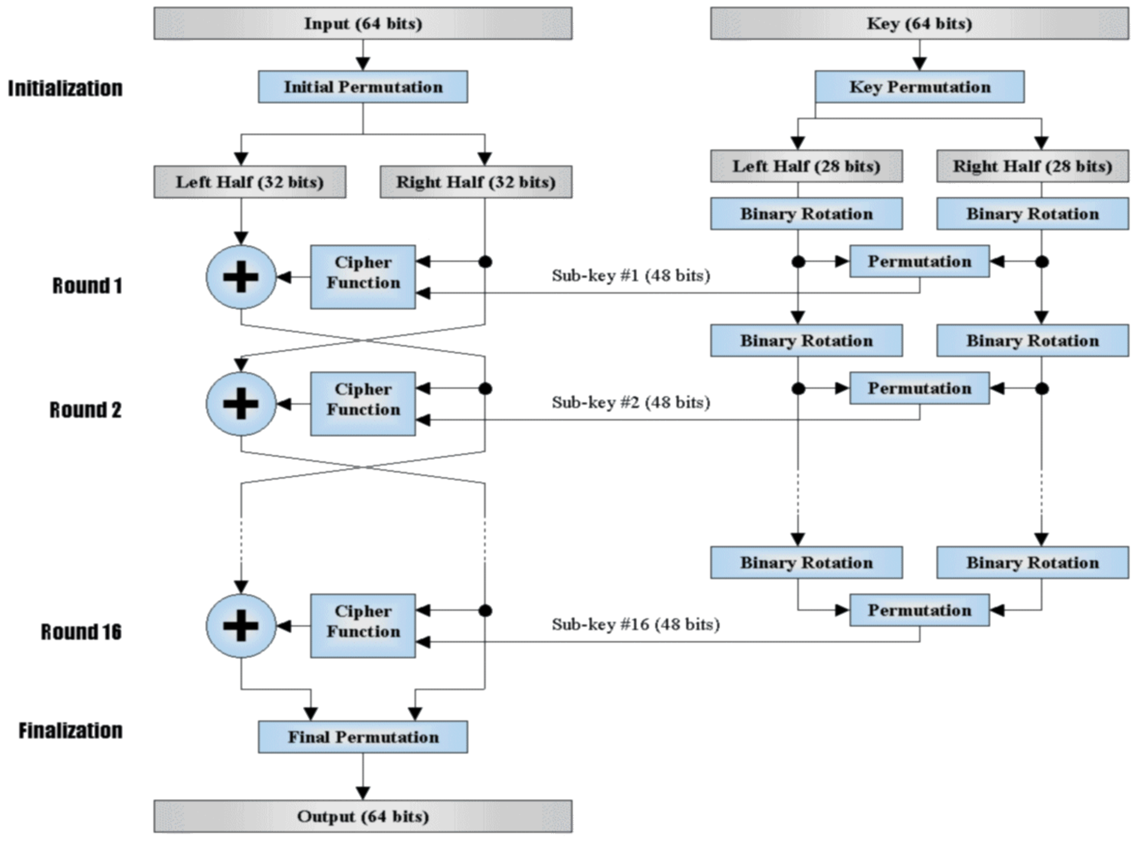


Figure 9: General Scheme of DES

DES algorithm is used in the Distributed-RNG protocol, and its related codes of implementation are provided in Appendix C.G.

2.3.3 RC2 Algorithm

RC2 algorithm firstly designed in 1987 is a data encryption algorithm. The size of input plaintext of this algorithm is the blocks with 64 bits length and the size ok key is variable. There are two main rounds in this algorithm: mixing and mashing rounds and a key expansion component. Today, RC2 encryption algorithm is vulnerable by key attack with the complexity of 2^{34}, but in this study this algorithm is used because of less complexity of algorithm which is beneficial for time complexity and battery consumption. This algorithm is used in this study, and its related code of implementation is provided in Appendix C.G.

2.3.4 DES MAC Function:

Mac functions are mostly used for two main reasons; authentication of message and integrity of message, a mac function is the same as hash function, but there is a basic difference between them, Mac functions uses a symmetric key while hash functions lack these key. The input of this function is a message with arbitrary length and a master key, and the result is known as tag. Mac functions use different cryptographic algorithms for their structure, for instance, AES, DES, Triple DES, Hash functions and etc. and most of them use cipher block chaining mode which are known as (CBC-MAC). In this study DES is used in order to encryption and constructing Mac function. Figure 9 shows a MAC design using DES in CBC mode of operation.

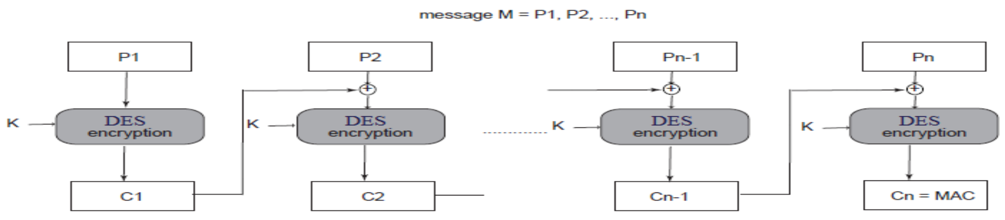


Figure 10: A MAC Design using DES in CBC Mode of Operation

2.4 NIST Tests

These tests consist of fifteen various tests each of which considers different aspects of entrance sequences. In this section, each of the NIST tests [4] will be briefly considered. Moreover, NIST tests represent how good are the random number generated by a RNG. The main criteria which are considered in NIST test on random bit sequences are:

a) Uniformity: This criteria indicates that the probability of occurrence one and zero in a truly random binary sequence is almost the same.
b) Scalability: A random sequence includes random subsequence; therefore, if the random sequence passes the tests, these tests should apply in all subsequences and all of the subsequences should also pass the tests.
c) Consistency: This creteria indicates the consistancy of starting values of generators a single seed or a single phsical output is not enough for a good random number generator.

There are some terms and definitions which should be defined before considering the fifteen NIST tests because these terms will be applied in the tests. At the end of this section, Table 1 briefly summarizes these definitions.

2.4.1 Frequency Test

The goal of this test is to compare the number of ones and zeros represented in a bit sequence, according to this test the probability of coming zero or one in a sequence should be the same (with probability ½). This situation should be applicable for subsequences. As it mentioned in NIST tests paper, the minimum bits sequence length should be at least 100 bits. The result of test is the p-value; if p-value ≥ 0.01, it means the sequence is random.

2.4.2 Block Frequency Test

This test is the same of the previous one, the proportion of zero and one will be evaluated in M-bit block. The number of ones and zeroes are expected to be M/2. This test for 1 bit block is frequency test. The reference of this test is " χ 2 distribution". The recommended sequence length for block frequency test is minimum 100 bits. The result of test is the p-value; if p-value ≥ 0.01, it means the sequence is random.

2.4.3 Runs Test

This test considers number of runs of 0 and 1, according to run test "the oscillation of zeros and ones will be represented which can be slow or fast". This test also follows the " χ 2 distribution". The input size of runs test is expected to be at least 100 bits.

2.4.4 Longest Run Test

In this test the longest run of ones in the M-bit block sequence will be compared with already tested one in regard to its consistency, " χ 2 distribution" is used as reference of the test. The minimum length of input sequence depends on its block size which is between 128 to 750000 bits.

2.4.5 Binary Matrix Rank Test

In this test, the sequence will be mapped to disjoined sub matrices and the rank of sub matrices will be obtained, it checks whether linear independency is exist in the sub string of sequence. This test also follows the " χ 2 distribution".

2.4.6 Discrete Fourier Transform Test

The periodic features of sequence such as, pattern repetition and pattern distance, is detected and it shows the deviation from the supposed randomness. The reference of this test is based on normal distribution. The length of input sequence should be at least 1000 bits.

2.4.7 Non-overlapping Template Matching Test

A specific pattern is selected from sequence, and then the number of its repetition will be counted. The search continues after finding funded patterns. This test also follows the "χ 2 distribution".

2.4.8 Overlapping Template Matching Test

This test is the same as the non-overlapping template test, the differences has been discussed in more detail in NIST paper. This test follows the "χ 2 distribution".

2.4.9 Universal Statistical Test

In this test, compressibility of the sequence will be considered; if the sequence is compressible it shows that the sequence is not random enough. The test follows normal distribution.

2.4.10 Linear Complexity Test

A length of a linear feedback shift register ("LFSR") is computed and it reflects the complexity of sequence. If the "LFSR" is long, the sequence is random, otherwise it is nonrandom. The statistic of this test follows "χ 2 distribution".

2.4.11 Serial Test

This test takes M-bit pattern and checks whether these patterns are similarly distributed over the random sequence. The test statistic in this test is "χ 2 distribution".

2.4.12 Approximate Entropy Test

This test is the same as previous one; the difference is comparison of pair of overlapping block of consecutive length (m and m+1). This test follows the statistic "χ 2 distribution".

2.4.13 Cumulative Sums Test

Cumulative sum of sub sequences which are in the test sequence is calculated and compared with the sum of random sequence. This statistic test is based on normal distribution. The minimum entrance sequence of this test is 100 bits.

2.4.14 Random Excursions Test

This test is define in NIST statistical tests as "computes the cumulative sums and checks if particular states (+4, +3,... , -3, -4) are visited with the distribution probability that should have a random sequence". The test statistic in this test is "χ 2 distribution".

2.4.15 Random Excursions Variant Test

This test is define in NIST statistical tests as "the same as the previous test, with the difference that the states that will be analyzed vary among a wider range (+9, +8,..,-8, -9)". This statistic test is based on normal distribution.

More information and details about the mentioned test has been widely conducted in NIST tests (“Introduction of National Institute of Standards and Technology“) in NIST tests paper [4].

Table 1: NIST Test Terms [4]

Term	Definition
Binary sequence	“A sequence of zeroes and ones.”
Bit string	“A sequence of bits “
Block	“A subset of bit string. A block has a determined length.”
Critical Value	“The value that is exceeded by the test statistic with a small probability (significance level).”
Entropy Source	“A physical source of information whose output rather appears to be random in itself or by applying some filtering process, which can input of a TRNG or PRNG.”
Kolmogorov-Smimov Test	“A statistical test which many be used to determine whether asset of data comes from a particular probability distribution.”
Level of Significance (alpha)	“the probability of falsely rejecting the null hypothesis, i.e., the probability of concluding that the null hypothesis is false when the hypothesis, in fact, true .the tester usually chooses this value; typical value are 0.05,0.01 or 0.001; occasionally, smaller values such as 0.0001 are used.”
p-value	“The probability that the chosen test statistic will assume value that are equal to or worse than the observed test statistic value when considering the null hypothesis. The P-value is Frequently called the “tail probability.”
Probability Distribution	“The assignment of a probability to the possible outcomes (realizations) of a random variable.”
Random Binary Sequence	“A sequence of bits for which the probability of each bit being a “0” or “1” is 1/2. The value of each bit is independent of any other bit in the sequence i.e. each bit is unpredictable.”
Random Variable	“Random variables differ from the usual deterministic variable in that random variable allow the systematic distributional assignment of probability values to each possible out com.”
Rank (of a matrix)	“Refers to the rank of a matrix in linear algebra over GF (2) having reduced a matrix into row-echelon from via elementary row operations, the number of nonzero rows. If any. are counted in order to determine the number of linearly indent rows or columns in the matrix. ”
Run	“An uninterrupted sequence of like bits. “

In summary, this chapter is surveyed due to being the base of this study, as it was mentioned. Therefore, the purpose of this thesis is to generate secure random number which needs to consider and survey related work and studies. In this chapter, firstly, the two basic generators have been considered: PRNGs and TRNGs, secondly, scatter protocol [3] has been discussed with more detail as the base of this thesis. Next, the materials, algorithms and functions that are appropriate for generators have been discussed. The previous proposed NIST tests have been almost considered in order to test the quality of obtained results. Finally, in Chapter 4, the results of NIST test are discussed.

Chapter 3

DISTRIBUTED-RNG PROTOCOL IMPLEMENTATION

In this study, a new protocol is introduced for constructing a message and distributes it through the network; the network can have different topology. However, the necessity of generating a true number or sequence in a wireless network is more prominent in comparison with the other types of the network because of the lack of high security of wireless networks. Thus, a mesh topology is supposed for implementation of the protocol. Figure 10 depicts some known network topology:

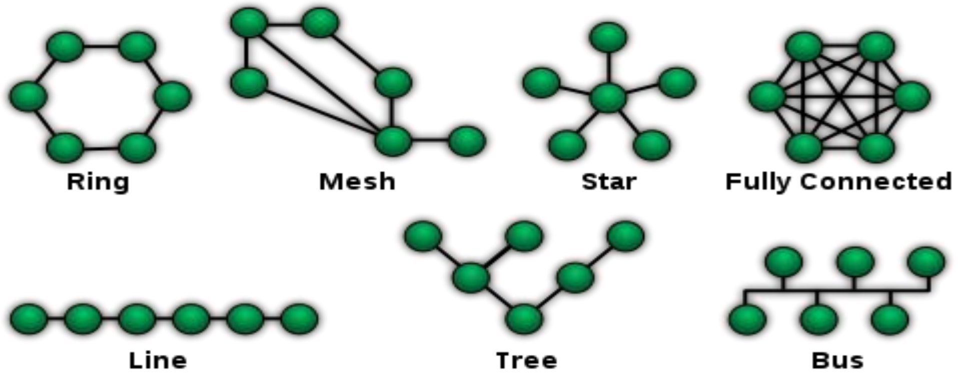


Figure 11: Some Commons Network Topologies

In a mesh topology constructed by wireless, each node has some direct or indirect neighbors, in a network consisting of n node there is at least one and at most n-1 neighbor. Figure 11 illustrates a mesh network for wireless which has both direct and indirect neighbor.

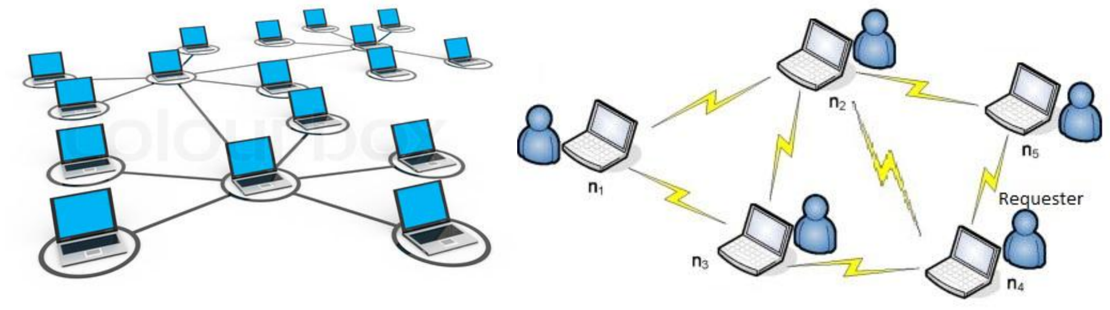


Figure 12: Scheme of Wireless Mesh Topology

There are two main roles for network nodes: requester- receiver; the former, requester, broadcast its request to all other nodes in order to receive a message containing desired random number or random sequenced and the latter, receiver, after receiving a message of requesting a random value, prepare a true random sequence and send it back securely to the requester. Each node in the network can be both requester and receiver concurrently. In this protocol, a requester can send its requests to its direct neighbor, indirect neighbors are not directly accessible by requester, for instance, if node n4 in Figure 6 plays the requester role, the nodes n2, n3 and n5 are direct neighbors of n4 and n1 is an indirect neighbor of n4. In the next session Distributed-RNG protocol will be defined and more details of protocol will be discussed.

3.1 Distributed-RNG Protocol Definitions

The distributed-RNG protocol includes two main phases: *request distribution* and *values collection* see Figure 12. The Implementation of the Distributed-RNG in C# is in Appendix section. The aim of request distribution phase is to distribute a request through the n-direct neighbors in a network in order to receive valid true random sequences. Collecting received values as true random sequence generated by other nodes is called values collection phase. As it was mentioned before, each node can play two requester and receiver roles, and each node contains both request distribution and values collection phases. In addition, whenever a node needs to have some random values, it broadcasts its request using the former phase, and after a definition delay (may be while the response is completed) it starts to latter phase and collect the reply messages. On the other hand, the number of collected values does not depend on the number of available network nodes; it means for a network consisting of n node, received messages may be lower than n and at most the received items reach n. Moreover, the messages passing through the nodes has a definition structure named frame, how to make the message and its related fields will be discussed later. The prepared messages is the most critical part of distributed-RNG protocol, because it holds valuable results which should be securely transmitted and all security requirement such as, integrity, confidentiality, and authenticity for avoiding security threads. Finally, collected results obtained from values collection apply in a TRNG module called MIXER. More detail about MIXER will discussed latter.

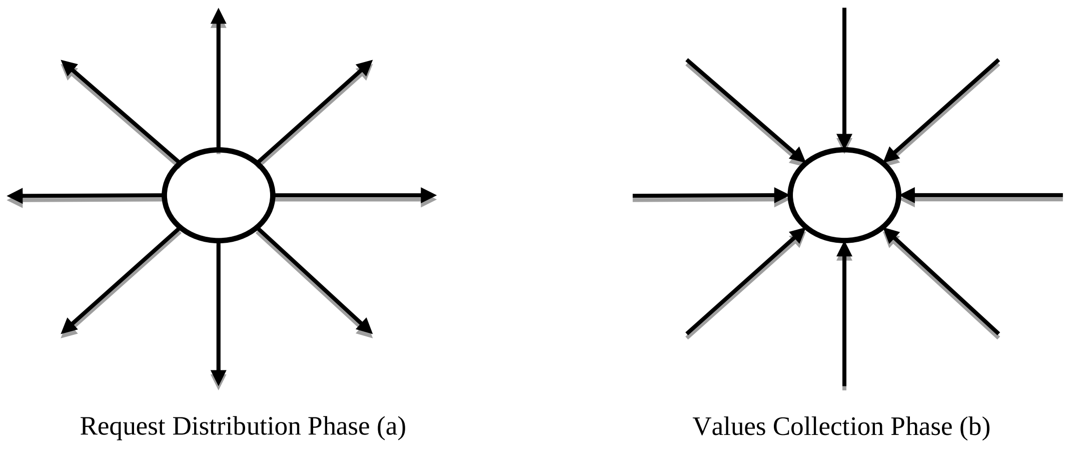


Figure 13: Both Request Distribution and Values Collection Phases

Figure 14 illustrates a general flowchart of Distributed-RNG protocol.

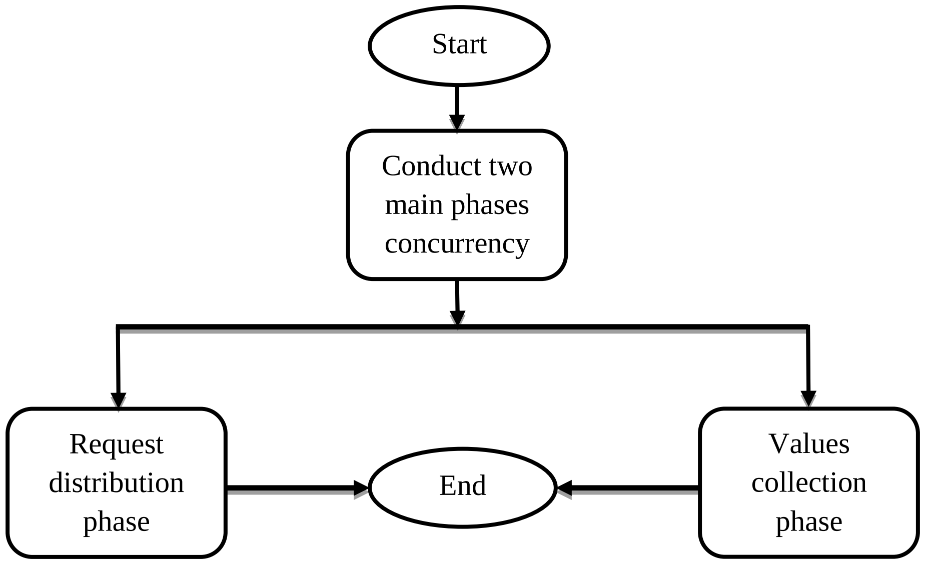

Start
Conduct two main phases concurrency
Request distribution phase
End
Values collection phase

Figure 14: Flowchart of General Flowchart Showing Concurrency of Request Distribution and Values Collection Phases

3.1.1 Message Definition

The message transmitted through nodes (requester and receivers) has to be defined by a structure, the message structure used for this purpose is called frame, thus, the network nodes, without considering their roles, should send each other a message packet as a request message or reply message. The general format of both phases (request distribution and values collection) of message is demonstrated below in Figure 14.

Crypto. 4 bits	Type 1 bit	Payload length 3 bits	Sender IP 32 bits	Nonce 8 bits	Hashcode 56 bits

(a) Request Message Structure

Crypto. 4 bits	Type 1 bit	Payload length 3 bits	Sender IP 32 bits	Payload 48 bits	Nonce 8 bits	Hashcode 56 bits

(b) Reply Message Structure

Figure15: Fields and Formats of RNG Protocol Message of Request Distribution Phase (a), Values Collection Phase (b)

As it was mentioned before, the new field crypto is added in the first part of both messages; this is one of changes which separate both Scatter and Distributed-RNG protocol. The indicated fields have the following meanings:

a) Crypto: This newly added field provides security type and defines kind of cryptography algorithm applied for encryption of necessary fields in order to transmitting message. As it illustrated in figure 11, four bits is considered for showing which algorithm for encryption and also which mode of operation is used to encrypt necessary message fields (nonce and hashcode will be encrypted as request message and payload, nonce and hashcode will be encrypted for reply message). The two least significant bits show encryption algorithm which are AC5 and DES and the two most significant bits demonstrate the mode of operation CBC (cipher block chaining), OFB (output feedback) or CFB (cipher feedback). This field provide security flexibility of Distributed-RNG protocol.
b) Type: As mentioned before, there are two type of messages request message and reply message. This field in a message shows the received message type; in request distribution phase this filed sets one and for the values collection phase it sets to zero. Hence, the receiver can recognize the type of message and will perform proper actions. the how message encryption, authentication and obtaining type of message will be later discussed.
c) Payload length: This field plays two role in messages and it related on payload field, as a request message this field indicates the number of bytes requester wishes to receive, whereas, the size of payload length is three bit; therefore, it can address maximum seven octet, but the protocol uses just six byte as payload field (this field

will be discussed later in this section). Nevertheless, as a reply message in the second phase, it indicates the exact number of bytes returned as payload. In RNG protocol the minimum sequence is 8 bits (payload length is one) and maximum is 48 bits (payload sets six).

d) Sender: It is the identifier of node making the message. This field provides IP address of both source of request node and reply node. Using this field the receiver can notice the message source and it knows where the result of generated random sequence should be sent. IP address consists of four parts separated with dot, each section take values between 1 and 255; it depends on the kind of network. Hence, for addressing these four sections 32 bit is necessary. And for requester it shows the received message and random sequence is related to which nodes in network.

e) Payload: As mentioned before, this field is just applicable for reply message and in request message (request distribution phase) it will be neglected. It contains the result of reading some entropy (it will be completely discussed later) as a true random sequence, and it will send back to requester. Payload has direct relation with payload length field, and it can be variable according to payload length field. The size of this field is maximum 48 bits, and payload length specifies how many random bit sequence should payload field have inside it. i.e.: if payload length sets on six (110), it means 48 bits random sequence is necessary. Furthermore, payload is the heart of message and due to it; this field should be securely encrypted .

f) Nonce: The role of nonce in cryptographic protocols is inevitable, the nonce is a arbitrary random number or sequence used just once in messages communication. İt is used in order to providing authentication and ensure security in cryptographic protocol, and it is significantly necessary to avoid reply attack. A nonce can be a random value or a pseudo-random value. Once it used in the first message of source node, it can not be repeated again, and receiver should provide reply message by increasing the value of nonce. İn this protocol one byte is considered in message in order to cover nonce.

g) Hashcode: The integrity of message will be provided by hashcode fields. In request distribution phase, the hash of all of the mentioned field will be computed and construcs hashcode field. Thus, both sender and receiver can ensure the integrity of received message by computing message digest of all other fields and comparing with the received hashcode. In Distributed-RNG protocol MD5 hash function is used to calculate message digest, it takes a message with arbitrary length and produce 128 bits digested value. This field has 56 bit length in message, therefore, only the 56 least significant bits of the result of MD5 is obted for hashcode field. Further detail of MD5 message digest will be considered later.

3.1.2 Request Distribution Phase

A requester node that wishes a random sequence prepares a message and broadcasts it over network. However, it is not guarantee that the number of reply messages is equal to all nodes in network. Because of using wireless, some nodes may not be available for sending back the reply message for different reasons such as, network issues, battery limitations, deadline of request timeout and etc. Implementation of request distribution phase is in Appendix C.E.

Moreover, this part constructs the main phase of the protocol. In this phase, a message in order requesting on appropriate random sequence is prepared and broadcasted through the network. On the other hand, after receiving a message, receiver provides proper information and desired random sequence and it sends the message back to requester. The message can be sent only to direct neighbor in the network, Figure 15 illustrates a wireless mesh topology, and direct neighbor of a node is shown, the black nodes are direct neighbor of node n1 and other nodes are indirect neighbor of node n1, thus the request of n1 can broadcast through the black nodes.

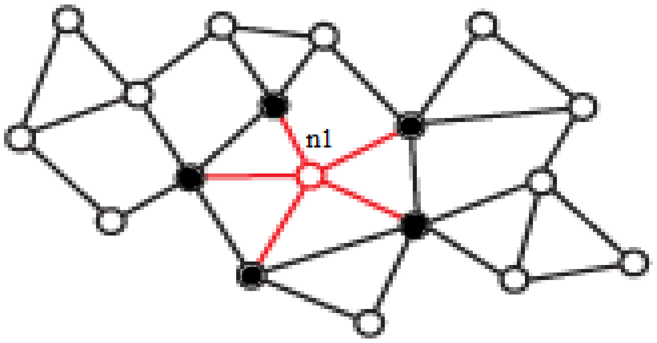


Figure 16: Request Distribution Phase through Direct Neighbors

Moreover, when requester decides to achieve a true random sequence, it should follow three main stages which construct the message, define a lifetime for collecting results and distribute it over network. These stages are defined as below:

a) Construct message: In this stage, all fields of message described in section 3-1-1 will be initialized and will be prepared, these are crypto, type, payload length, sender, payload, nonce and hashcode each of which should be exactly initialized, since that payload field is not used in request distribution phase, it will be considered in values collection section. The base of the protocol is rely on bit structure (bit oriented) , the Figure 16 illustrates these fields and their position and length in a message. The total size of request message is 13 byte.

Crypto. 4 bits b0,b1,b2,b3	Type 1 bit b5	Payload length 3 bits b6,b7,b8	Sender IP 32 bits b9-b40	Nonce 8 bits b41-b48	Hashcode 56 bits b49-b104

Figure 17: Bitwise Structure of Message

First of all, the encryption algorithm is defined, Distributed-RNG protocol supports RC2 and DES encryption algorithm, each of which will be discussed in detail later. Then, the mode of operation will be defined; it can be CBC (cipher block chaining), OFB (output feedback) or CFB (cipher feedback). As it mentioned, the Crypto field is used for this purpose, this field consists of four bits supposed as b0-b3. The bits b2 and b3 shows encryption algorithm used for encryption; and bits b0 and b1 define the mode of operation related to selected encryption algorithm. The former bits group b2 and b3 address four values, but only two values of them are used to address RC2 and DES, and the other two values are reserved in order to further modification. If these two least significant bits indicate one (01) then the selected algorithm is DES and if it shows (10) RC2 will be selected as encryption algorithm. This method is also used for the two most significant bits b0 and b1 for defining the mode of operation, zero (00), one (01) and two (10) respectively shows CBC, OFB and CFB. I.e. if four bits of crypto fields shows ten (1001) it means, encryption algorithm is DES with CBC mode of operation. Next, b5 bit which is the type of message have to set to 1 to notify receiver that this message is located on request category. Then, requester is able to define how many bit sequence it wishes to receive from other nodes, by filling payload length field (b6,b7,b8), as mentioned before, although these bits address at most seven (111) number, RNG protocol can take at most 48 bit random sequence (6 bytes) in its payload field; it means when payload length is six (110), six byte (octet) requester wishes to receive, and since that payload field is at most 48 bits, the number seven can’t be used for payload length. Next, IP address of sender consists of four part each of which are values between 1 to 255, by using 32 bit we can address these four numbers; each part can be shown with 8 bit, for instance if the first eight bit take values (11111111) it indicates the first part of IP address has value 255; hence all four separated parts can be shown by 32 bit, in this protocol bits 9 until 40 in the message is reserved for this purpose. The fifth field is nonce, it takes one byte random values from bit b41 till b48 which requester should prepare, it can be a random value or a pseudo-random value. Then, requester which broadcast the message by arbitrary value of nonce will be expected to receive in reply message its random value plus one. Nonce has to be encrypted because of providing confidentiality. During a communication the old values of nonce should not be repeated and the received messages with similar nonce should be ignored to avoiding reply attack, Finally, the last 56 bits of message is related on hashcode, from bits b49-b104, as mentioned before the message digest algorithm used for RNG protocol is MD5 which take an arbitrary length message and produce 128 bit digested value, the input of MD5 hash function of this protocol is the first 49 bits of message, it means, all fields before last field will be included as input of hash function, hashcode field provide integrity of message, both requester and receiver will ensure that the fields of message has not changed during transmission and communication line. The 56 most significant bits of output of digested value will be selected and added to hashcode field. Then, before transmitting the message, both nonce and hashcode will be encrypted by defined algorithm and mode of operation having been considered in crypto field. These fields are considered together as 64 bits plaintext which are input of selected algorithm, If the used algorithm is DES, it is supposed that the 64 bits master key is agreed in both requester and receiver nodes, the result of encryption is 64 bits of ciphertext and can be added to the last 64 bit of message. But if RC2 is opted as encryption method both requester and receiver side agree on a master key with variable length, and again the 64 bits result of encryption will add to the last 64 bits of message. As a result, the message is ready to broadcast with considering all of the security necessities. The messages below illustrates final construction of message.

Suppose that:

$$M1 = crypto + type + payload\_length + sender$$

$$M2 = Hash(M1 + nonce)$$

Then message $= M1 + E_{Kmaster}[nonce + M2]$

b) Defining collecting results lifetime: After preparing message, a given time respite should be defined for request distribution phase. Before requester broadcast its message through the network, a timer will be started and the process terminates when time respite meets its deadline or all nodes in network sends back their reply messages. In fact, this time respite is directly related to the second phase (values collection), it means after finishing this time, the requester can gather the data collected from values collection phase. By considering this conditions, if the time respite is not be considered, any failure in the network which prevent reaching reply message leads to failure in protocol; there is no termination condition.

c) Message distributing: In this stage, the message has been prepared to send and also time respite has been defined; consequently, the request distribution phase broadcast the message through network nodes. Once the phase starts, the value collection phase will concurrently start in order to gather the reply messages. This condition being continued until time respite finishes or values collection phase receive all reply messages, more details of termination condition will be considered in section 3-1-3.

The Flowchart shown in Figure 17 illustrates of general view of request distribution phase.

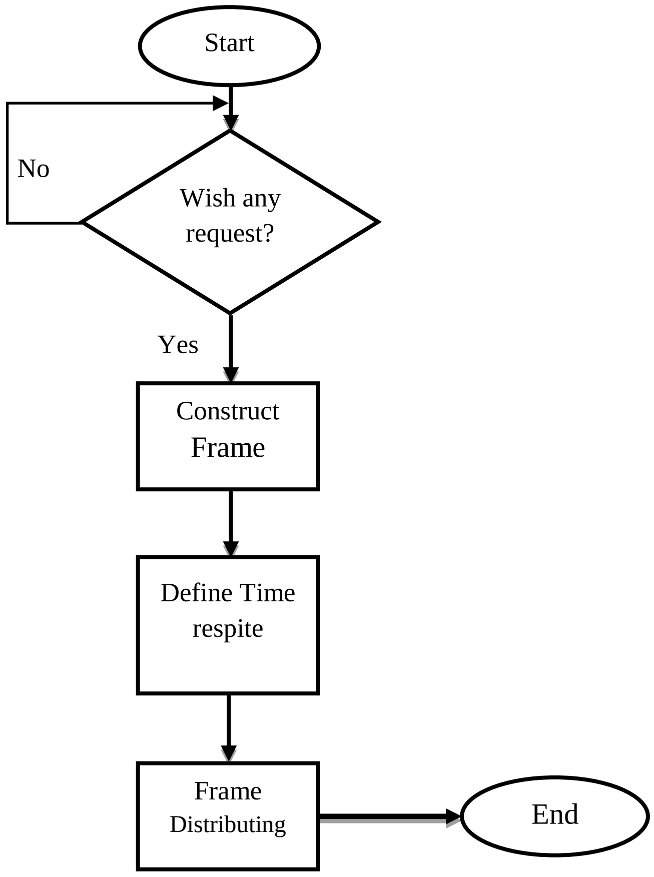


Figure 18: The General Flowchart of Request Distribution Phase

3.1.3 Values Collection Phase

This phase is a more considerable in that the arrival message on a node is divided to two main categories: incoming reply messages and incoming request message; and values collection phase take two arrival messages. First of all, the values collection phase should decide whether arrival message is the reply message or request message of another node. As mentioned before, a network node includes two phases and can be both requester and receiver. Suppose that a node broadcasts its request and wishes to receive some random sequences, so it expects to receive results of its request (replies), on the other hand, another node has the same situation and broadcast its own request, the messages received by the first requester is not only its expect about collecting results, and can be also request of another node. Consequently, these two different incoming messages and their respond routine will be considered separately. Moreover, the values collection phase should follow three main sub stages. These stages are defined as below and its implementation is provided in Appendix C.C:

a) Message decodes and authentication: First of all, the incoming message fields have to be retrieved. And, the receiver should be ensured that security necessities are provided; authenticity, integrity and confidentially. Before discovering the mentioned issues, the type of message should be determined. Total size of request message is 104 bits and this value for reply message is 152 bits.

   The fifth bit of received message determines the type of the message, if the type field has value one, its means this message is request message, otherwise, by having value zero it is a reply message. The next stage is authentication. The Distributed-RNG protocol supports two encryption algorithm and three mode of operation for encrypting message. By analyzing crypto field the algorithm and its mode of operation will be discovered. As mentioned in 3-1-2 section and part a, this field has four bits that the two least significant bits specify the encryption algorithm and the two most significant bits indicate the mode of operation, see Table 2.

Table 2: Structure of Crypto Field

Crypto field	Value	Descryption
The two least significant bits	01	DES
	10	RC2
The two most significant bits	00	CBC
	01	OFB
	10	CFB

Next, for the request messages, the last 64 bits of message, nonce and hashcode, will be taken as input of decryption algorithm, and will be decrypted using obtained encryption algorithm and mode of operation. Thus, the both fields, nonce and hashcode, will be discovered. Then, the 40 most significant bits of message feeds in MD5 hash function, and the message digest of crypto, type, payload length, sender, nonce will be recomputed and compared with the newly obtained hashcode for ensuring integrity, If they are same, it proves the validity of message and all remain fields can securely retrieved. On the other hand, for the reply messages, the last 104 bits of message will be encrypted and the result of encryption reveals these fields: payload, nonce, hashcode. The authentication of reply message is the same as request message. Finally , the reply messages will be considered in reply messages routine and request messages will be discussed in reply messages routine.

b) Reply messages routine: This section focus on reply messages. When a requester node broadcasts its request, it receives lots of reply messages, thus, only the requester node takes such a message, it means the nodes which don't have any request will not receive any reply message (the incoming message of these nodes is only request messages emitted by other requesters). In the previous section, the type of incoming messages has been determined and the integrity of message has been proved and also all of message fields have been retrieved. The desired field for values collecting phase is payload field which contain random bits sequence. The size of the random bits sequence is related to payload length field which requester determines before distributing of message, i.e. when requester construct a message and set its payload length to six (110) and broadcast it, requester receives the messages containing six bytes random sequence in its payload length field. As it mentioned in request distribution phase, a time respite T had been defined before the requester broadcast its request, hence, reply messages routine collect the incoming values during time T. Finally, after expiring the time T, values collection phase in the requester will not accept other incoming reply messages. It means the requester achieves its desired random bit sequenced and enough values are collected. Another condition for termination of collecting reply messages is when all of network nodes return their reply messages back; for instance, if requester broadcasts its request message to n node, exactly n reply message received by requester. As a result, the sequences collected from reply messages routine will be sends to a MIXER module in order to generate true random sequence. The section 3-1-4 describes the MIXER module.

c) Request messages routine: This section focuses on request messages. When a requester node broadcasts its request, the other nodes receive request message. After authentication and retrieving message fields having been done before, the value of payload length field is taken. Then according to desired random sequence indicated in payload field, which takes value from 1 to 6 (because it has three bits), one till six bytes true random value will be return by payload field. This random value will be obtained by some entropy.

The main purpose of this part is to generate a true random number using some entropy source. Two values are used for this purpose: CPU temperature, CPU elapsed time on a non-Idle thread.

CPU temperature is mostly used in lots of security algorithms (it is discussed in the related work, second Chapter) and it can be considered as good entropy for taking some random values, these values may be absolutely different in computer nodes in the network and if the number of network nodes increase, better result will obtained. Secondly, there is another aspect of CPU which provides more randomness value; it is the percentage of elapsed time that the processor spends to execute a non-Idle thread Appendix C.F. It is calculated by measuring the percentage of time that the processor spends executing the idle thread and then subtracting that value from 100%. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). Finally, the multiplication of the mentioned values makes a true random number; this value is called total value entropy. The payload field will be filled from total value entropy. Other fields which are necessary to obtain from request message are sender and type. Constructed message should be returns back only to the IP address indicated in sender field as a reply message. Thus, before constructing message, the type field of message should be set as zero (reply message) and after construction of message, message should not be broadcasted and must be sent to IP address of sender. The other stage of message encryption, message digest computation and constructing message is the same as request distribution phase. The Figure 18 illustrates general view of values collection phase.

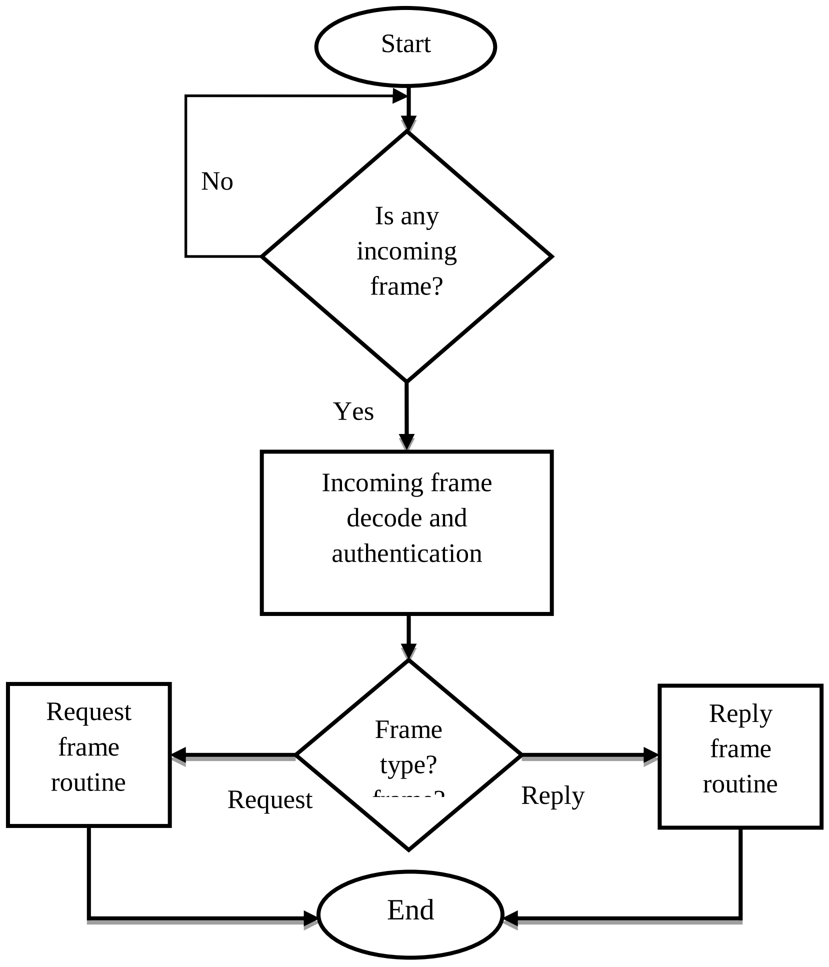


Figure 19: Flowchart of General View of Values Collection Phase

3.1.4 MIXER Module structure

The block diagram of MIXER module for Distributed-RNG protocol is illustrated in Figure 19. The code of implementation of this module is in Appendix C.D.

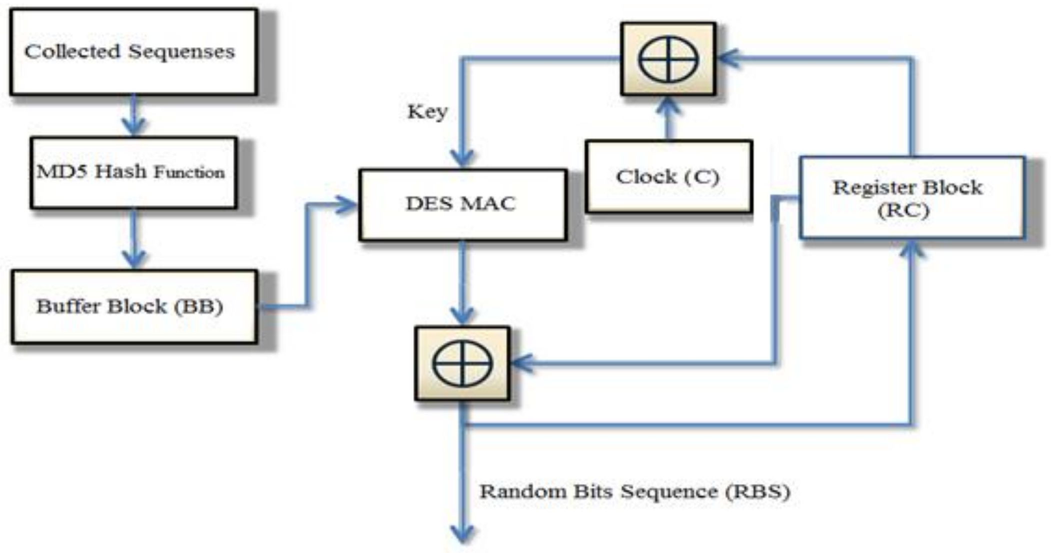


Figure 20: Block Diagram of MIXER Module

Distributed-RNG is completely a bit oriented protocol and it is based on bit sequences. The input of MIXER module is the results of values collection phase for requester and its output is 64 bits true random sequence. In this module, the random bit sequences collected from other nodes will be attached each other; thus, concatenation of these values will be feds to MIXER module input, the length of input sequence is variable and it depends on the number of received message. In the next stage, the digest of concatenated sequences will computed by MD5 hash function. The output of MD5 hash function is digested value with 128 bits length, lets the set of collected sequences obtained in values collection phase be set of X, then:

$$D = Hash(X_1 + X_2 + ... + X_n),$$

Then, this value (D) will be copied to the 128 least significant bits of a buffer block (BB) whose length is 256 bits, and old value of the 128 least significant bits will be copied to the first half bits; it means after 128 bits left shift of BB, the 128 least significant bits will updated by the result of MD5 function. These stages are shown as below:

$$BB = (BB << 128) + D ,$$

Next, the output of the buffer block will be the input of DES MAC function, the key of DES MAC is obtained by the XOR of a clock (C) and register block (RB), where C obtained from the 16 least significant bits of microsecond of local clock of computer, the content of RB is initialized by 64 bits zero, and then will be updated by 64 bits random sequence finally generated by MIXER module. Finally, the XOR of 64 bits output of DES MAC with RB content represents true random sequence. In summary, the random bits sequence (RBS) is generated as follows:

$$RBS = TripleDesMAC(BB, RB \oplus C) \oplus RB$$

3.1.5 General Scheme

As mentioned before, each node in the network consists of two main phase which concurrently run. The description of Distributed-RNG protocol and its two main phases: request distribution and values collection, having been discussed in foregoing section, and also their relations and orders is depicted by a two general Flowchart Figure 20.

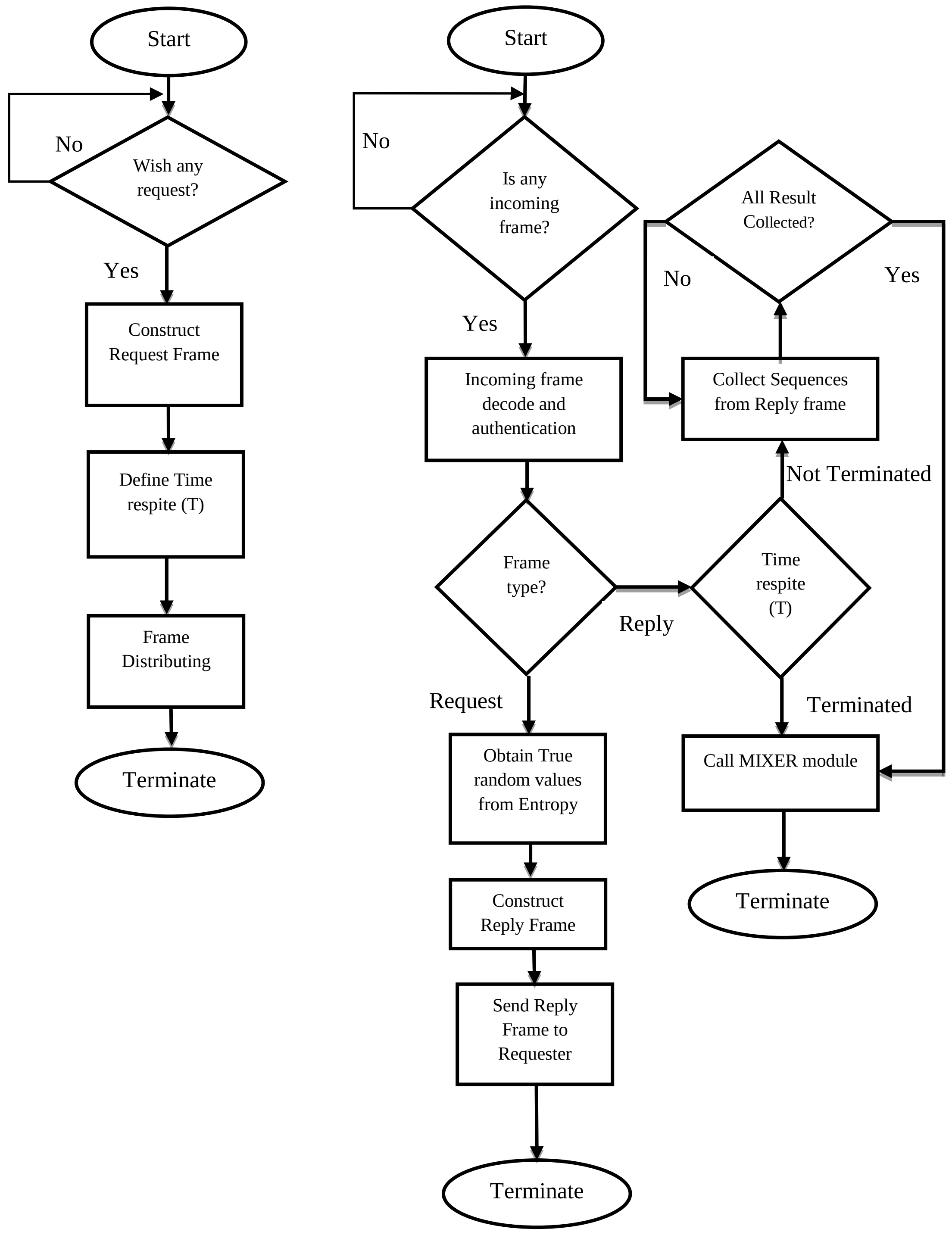


Figure 21: Flowchart of General View of Distributed-RNG Protocol Phases

In order to summarize, the Distributed-RNG protocol has been considered in detail in this chapter and its implementation using C# programing language is in the Appendix section. Then, the two main phases of protocol has been discussed: request distribution phase and values collection phase. The constructed message which is known as the heart of protocol has been defined. Furthermore, the differences between this protocol and Scatter protocol have been shown.

The necessity of having appropriate and secure protocol for especially vulnerable wireless network in regard to flexibility and security leads to making some essentially changes in Scatter protocol structure. A new field added to message (Crypto) for making the protocol more flexible than the Scatter protocol. By using this field, the encryption algorithm and its mode of operation is represented in the message; thus, receiver can notice how it can decrypt the encrypted parts of the message. Another change providing confidentiality of the message is encryption of payload field which handles the obtained random value of each node. Scatter protocol doesn’t encrypt payload and it compromises the confidentiality of message.

Chapter 4

EXPERIMENTAL SETUP AND RESULTS OF NIST STATISTICAL TESTS

Cryptographic algorithms need random numbers for lots of reasons, like nonce, public keys, private keys, session keys, secret keys, seeds, salts, padding and etc. Randomness and unpredictability are the property of a good random number. Thus, weak random numbers make the security algorithms more vulnerable. In the Chapter 3, Distributed-RNG protocol was described, the result of this generator is 64 bits random sequence. The quality of generated bit sequences will be evaluated under NIST statistical tests having been discussed in section 2.4. For conducting the result NIST statistical test suite is used [24] which is the application of NIST tests. In this Chapter, the logically and physically connectivity between network nodes will be considered in experimental setup section 4.1 and conducting and comparison of the results will be considered in 4.2, 4.3.

4.1 Experimental Setup

4.1.1 Physical Setup

For setting up the laptops and taking the results four laptops had been located in EMU library in the ground floor and they had constructed a wireless network consisting of four nodes with full mesh topology. The distance of the requester laptop with modes n2, n3 and n4 is respectively 8 meter, 6 meter and 5 meter. For obtaining the results, the requester laptop broadcasts its request every 60 millisecond, it means that the duration of broadcasting the request and collection the results for each random bit sequence is 60 milliseconds. Consequently, 156250 random bit sequences was collected that the length of each random bit sequence is 64 bits. As a result, 10000000 random bits was collected in 2 hours and 36 minutes. Figure 21 illustrates physically location of laptops and their distances.

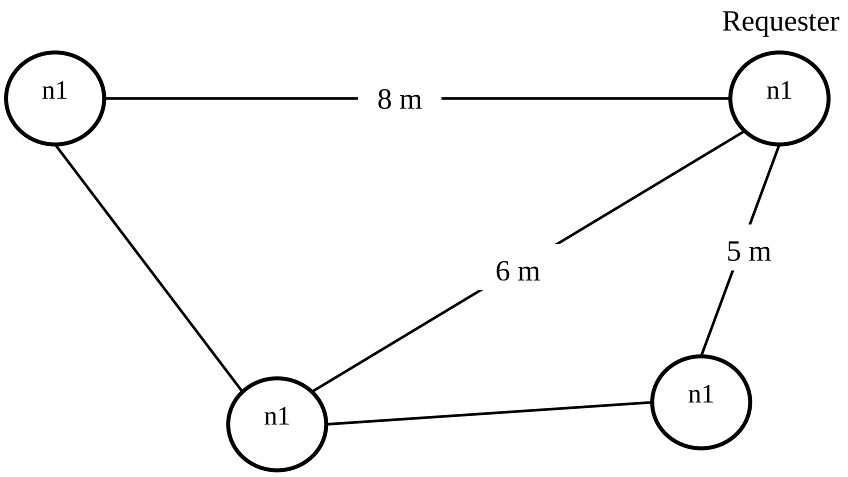
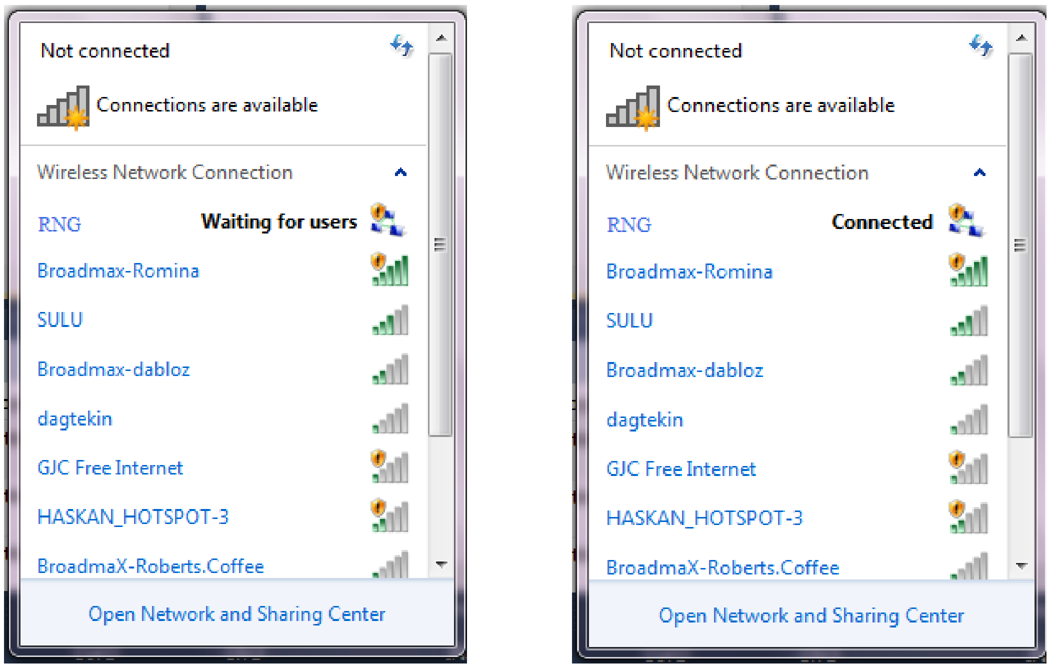


Figure 22: Physically Setup Laptops Constructing Wireless Full Mesh Topology

4.1.2 Connection Setup

For providing a wireless network each laptop needs to organize dynamic connection, consequently, the Distributed-RNG application which is started in each laptop (node) should dynamically set up a wireless ad-hoc (computer to computer) network. First of all, a wireless profile should be defined and made; it consists of a profile name, SSID configuration, authentication and encryption. Then, the laptop should be connected to the network which is specified by wireless profile, in this case there may be two conditions, if there is not any wireless network which makes wireless profile before, application of current laptop makes the wireless profile and waits for other network nodes in the protocol. On the other hand, if there is any laptop which already sets wireless profile for this protocol, the application of current laptop is connected to the previous wireless network, see figure 22. The related codes of this phase are provided in Appendix C.H.



Not connected
Connections are available
Wireless Network Connection
RNG Waiting for users
Broadmax-Romina
SULU
Broadmax-dabloz
dagtekin
GJC Free Internet
HASKAN_HOTSPOT-3
BroadmaX-Roberts.Coffee
Open Network and Sharing Center

Not connected
Connections are available
Wireless Network Connection
RNG Connected
Broadmax-Romina
SULU
Broadmax-dabloz
dagtekin
GJC Free Internet
HASKAN_HOTSPOT-3
BroadmaX-Roberts.Coffee
Open Network and Sharing Center

Figure 23: Wireless Network Statuses

4.2 Conduct the Results

For conducting results, the algorithm of Distributed-RNG protocol had been run in a wireless network consisting of four computers, and the results had been saved. These results which are about 10,000,000 random bits sequence will be fed as input of NIST test. To address the tests, the results are saved in a text file containing ten million bits represented as zero and one ASCII code. Then this file is fed to NIST tests suite for testing with separation of 10000 bits as length of each sequence and 1000 bits as the number of sequences to test Table 3. The results of these fifteen tests obtained according to p-values and proportion of passing (ratio) out of 1000 is tabulated and compared with the existing protocol (Scatter protocol). On the other hand, this test had been done before in the previous proposed study (Scatter) with 20 wireless sensor nodes and the result was gathered according NIST tests. The comparison results are tabulated in Table 4.

4.3 Discuss the Results

According to NIST tests, p-values are the main aspects of evaluation of the quality of sequence randomness. If the p-value obtained by each of the mentioned tests be equal or greater than 0.01, it means the sequence is random with confidence of 99 percent. Thus, the p-values are tabulated in Table 4 in order to consider their condition of randomness. Then, the ratios of passing p-values of these fifteen tests are also tabulated in Table 4. The most prominent part of conducting results is that comparison of Distributed-RNG protocol with already proposed Scatter-RNG [3] which is in fact the main source of this thesis.

Table 3: NIST Parameters

Parameter	Value
Length of each sequences(L)	10000
Number of tested sequences(s)	1000
Threshold for P-value	0.01

Furthermore, this test is applied for a wireless network consisting of four nodes; the requester makes its message (message) and broadcasts it through the other nodes, here in, the encryption algorithm sets to DES and CBC mode is opted as mode of operation.

Table 4: Comparison of Results of Scatter-RNG and Distributed-RNG Protocol

TEST	Scatter RNG In WSN		Distributed-RNG protocol	
	P-value	Ratio	P-value	Ratio
Frequency	0.1223	0.9834	0.737915	0.9901
Block Frequency	0.3508	0.9910	0.583145	0.9920
Runs	0.1223	0.9811	0.361938	0.9811
Longest Run	0.5341	0.9916	0.834308	0.9919
Binary Matrix Rank	0.7351	0.9853	0.834308	0.9903
Discrete Fourier Transform	0.2135	0.9910	0.496351	0.9871
Non Overlapping template matching	0.4602	0.9893	0.691081	0.9891
Overlapping template matching	0.3509	0.9831	0.35485	0.9893
Maurer’s Universal Statistical	0.8065	0.9996	0.275709	0.9948
Linier Complexity	0.8965	0.9923	0.7632	0.9976
Serial	0.5348	0.9974	0.173770	0.9941
Approximate entropy	0.7451	0.9959	0.162606	0.9897
Cumulative sums	0.7392	0.9882	0.980341	0.9882
Random excursion	0.6402	0.9811	0.437274	0.9941
Random excursion variants	0.7502	0.9941	0.209948	0.9881

As it can be concluded from Table 4, the average rate of passing p-values in scatter protocol is 0.9895 while this value for Distributed-RNG protocol increases to 0.9905. It shows that almost better result is obtained in regard to proportion of passing sequences. Then it can be seen that eight tests of fifteen tests obtained greater ratio in comparison with the proposed one. The p-value of frequency, longest run, binary matrix rank and cumulative sums are around 80 percent in Distributed-RNG protocol which are greater than the previous protocol and the other tests have gentle growth than the others and in some tests such as Maurer’s universal statistical, linear complexity, serial, approximate entropy, random excursion and random excursion variants the p-values have been decreased.

In summary, despite being more flexible and secure than Scatter, Distributed-RNG protocol shows that 60 percent of p-values and also 60 percent of ratio obtained from Table 4 is equal or greater than the Scatter protocol.

Chapter 5

CONCLUSION

The necessity of generating random and pseudorandom numbers increases in lots of cryptographic applications. The networks need cryptography to provide their security, these necessities are more sensitive in vulnerable networks such as wireless networks; thus, this study enhances a proposed protocol of RNG via wireless sensor network and introduces optimized Distribute-RNG protocol which is more flexible and secure. On the other hand, the generated random value must be random enough to be used in the cryptographic algorithms; indeed, their outputs must be unpredictable in the absence of knowledge of the inputs. Then, some recommended statistical tests are provided. A set of statistical tests for randomness is described in the national Institute of Standards and Technology (NIST). These tests may be beneficial at first to determine if a generator is appropriate for a particular cryptographic application or not. Nevertheless, no set of statistical tests can completely approve that the generator is suitable enough for applying in a particular application.

The results obtained from firstly prove that introduced Distributed-RNG protocol is a powerful generator and can be securely used in vulnerable wireless networks and can be reliable source for cryptographic algorithm which needs nonce, public keys, private keys, session keys, secret keys, seeds, salts and etc. Secondly, these results are compared with the existing study and show 60 percent of the test values obtained better results.

As further development, I intend to introduce such a protocol in LAN networks and Internet and also design a method for sharing securely master keys via networks and finally obtain more quality random numbers. I am trying to develop application in ad-hoc networks containing laptops and mobiles together, using C# and android programing.

REFERENCES

[1] D. Estrin, L. Girod, G. Pottie, M. Srivastava. Instrumenting the World with Wireless Sensor Networks. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'01), pp. 2033–2036, 2001.

[2] G. Anastasi, G. Lo Re, M. Ortolani. WSNs for Structural Health Monitoring of Historical Buildings. In 2nd Conference on Human System Interactions, pp. 574 –579, 2009.

[3] L. Giuseppe, M. Fabrizio, O. Marco. Secure Random Number Generation in Wireless Sensor Networks.

[4] Http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf.

[5] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, T. Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. 2004.

[6] L. Blum, M. Blum, M. Shub. A Simple Unpredictable Pseudo-Random Number Generator, SIAM Journal on Computing, pp. 364–383, 1986.

[7] N. Ferguson, B. Schneier. Practical Cryptography. published by Wiley. 2003.

[8] M. Matsumoto, T. Nishimura, M. Twister. A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation, pp. 3–30, 1998.

[9] http://www.iro.umontreal.ca/~lecuyer/papers.html

[10] B. Jun, P. Kocher. The INTEL Random Number Generator. Cryptography Research Inc. white paper, 1999.

[11] A. Stefanov, N. Gisin, O. Guinnard, L. Guinnard, H. Zbinden. Optical Quantum Random Number Generator. Journal of Modern Optics, pp. 595–598, 2000.

[12] I. Vasyltsov, E. Hambardzumyan, Y.S. Kim, B. Karpinskyy. Fast Digital TRNG Based on Metastable Ring Oscillator. Cryptographic Hardware and Embedded Systems–CHES 2008, pp. 164–180, 2008.

[13] R. Latif, M. Hussain. Hardware-based Random Number Generation in Wireless Sensor Networks (WSNs). Advances in Information Security and Assurance, pp. 732–740, 2009.

[14] V. Gaglio, A. De Paola, M. Ortolani, G. Lo Re. A TRNG Exploiting Multi-Source Physical Data. In Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks, pp. 82–89, 2010.

[15] A. Francillon, C. Castelluccia. TinyRNG: A Cryptographic Random Number Generator for Wireless Sensors Network Nodes. In 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, pp. 1–7, 2007.

[16] C. Karlof, N. Sastry, D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In SensSys, ACM Conference on Embedded Networked Sensor Systems, 2004.

[17] S.A. Camtepe, B. Yener. Key Distribution Mechanisms for Wireless Sensor Networks: a Survey. Technical report, 2005.

[18] T.H. Chen, W.K. Shih. A Robust Mutual Authentication Protocol forWireless Sensor Networks. ETRI Journal, pp. 704–712, 2010.

[19] Y. Yu, K. Li, W. Zhou, P. Li. Trust Mechanisms in Wireless Sensor Networks: Attack Analysis and Countermeasures. Journal of Network and Computer Applications, pp. 867 – 880, 2011.

[20] Z. Li, G. Gong. A Survey on Security inWireless Sensor Networks. Technical report, 2011.

[21] A. Perrig, J. Stankovic, D. Wagner. Security in Wireless Sensor Networks. Communications of the ACM, pp. 53–57, 2004.

[22] M.M.J. Stevens. On Collisions for MD5. MIT master thesis, 2007

[23] National Institute of Standards and Technology (NIST). Data Encryption Standard. FIPS Publication, 1993.

[24] http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.1.zip.

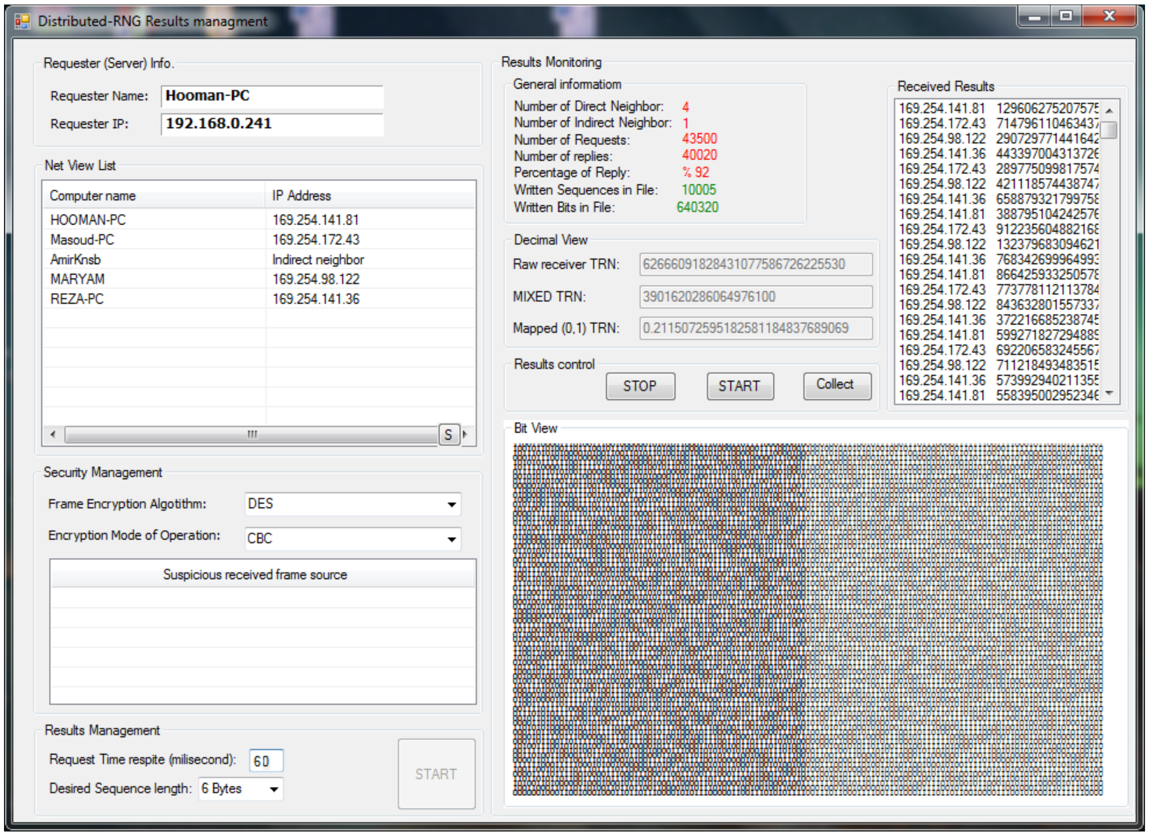
APPENDICES

Appendix A: Used Program Language (C#)

The programming language C# (Microsoft Visual Studio 2010) is used for implementation of Distributed-RNG application due to its simplicity, capability, compatibility, flexibility and lots of other positive aspects. C# provides a user friendly interface for programmers and offers variety of functions related to cryptography and security. C# is a network-based program; and it is comfortable to write distributed programs via network, there are lots of tools for socket programing and message passing interface. Finally, C# is not only simple and user friendly but also it provides pure power.

Appendix B: Application Screen Shots

To illustrate the implemented Distributed-RNG application, here in, the screen shot of application are provided and the stage of using application are followed.



Requester (Server) Info.

Requester Name: Hooman-PC

Requester IP: 192.168.0.241

Net View List

Computer name	IP Address
HOOMAN-PC	169.254.141.81
Masoud-PC	169.254.172.43
AmirKnsb	Indirect neighbor
MARYAM	169.254.98.122
REZA-PC	169.254.141.36

Security Management

Frame Encryption Algotithm: DES

Encryption Mode of Operation: CBC

Suspicious received frame source

Results Management

Request Time respite (milisecond): 60

Desired Sequence length: 6 Bytes

Results Monitoring

General informatiom

Number of Direct Neighbor:	4
Number of Indirect Neighbor:	1
Number of Requests:	43500
Number of replies:	40020
Percentage of Reply:	% 92
Written Sequences in File:	10005
Written Bits in File:	640320

Decimal View

Raw receiver TRN: 62666091828431077586726225530

MIXED TRN: 3901620286064976100

Mapped (0,1) TRN: 0.2115072595182581184837689069

Results control

Received Results

```
169.254.141.81   129606275207575
169.254.172.43   71479611046343
169.254.98.122   29072977144164
169.254.141.36   44339700431372
169.254.172.43   289775099817574
169.254.98.122   42111857443874
169.254.141.36   65887932179975
169.254.141.81   38879510424257
169.254.172.43   91223560488216
169.254.98.122   132379683094621
169.254.141.36   76834269996499
169.254.141.81   86642593325057
169.254.172.43   773778112113784
169.254.98.122   84363280155733
169.254.141.36   37221668523874
169.254.141.81   59927182729488
169.254.172.43   69220658324556
169.254.98.122   71121849348351
169.254.141.36   57399294021135
169.254.141.81   55839500295234
```

Figure Appendix 1: The Main Scheme of Application

Figure Appendix 1 is a screen shot of Distributed-RNG application running on a computer connected to five laptops via an ad-hoc wireless network. Requester info group box illustrates the name and IP of this computer. Then, net view list shows the computers joined in created ad-hoc network, in any times, computers can be added to the list and can leave the ad-hoc networks, a node in the network which can't be access able by this computer is known as indirect neighbor; it means no request is sent to that node. In this Figure AmirKnsb computer is connected to network via MARYAM computer, thus, it is indirect neighbor of this computer. Next, in security management group box we can select two encryption algorithms: DES and RC2 and also three modes of operations: CBC, OFB, and CFB using combo boxes. There is a window in security management group box which represent the list of suspicious computers during running protocol; for instance, if the nonce of received message differ from the expected nonce, this computer will add to this black list and will not contribute in the protocol in next message. After adjusting the security options, the result options should be adjusted, request time respite determine the deadline T, it means, in each request, just the results are acceptable receiving during this time, in this application, after finishing respite time another request will be broad cast. Another option of result management is desired sequence length which is payload length (maximum is six). By clicking on start button, the protocol is begun.

On the other hand, in result monitoring group box some utilities information is shown in separated parts. Firstly, general information group box displays some information about neighbors situations, the number and percentages of requests and replies messages and some information about the bits sequence written in a file and also the number of bits written in the file. Then, received result group box has a window which shows each obtained result and the source of received results in any time. In the decimal view group box there are three parts:

a) Row receidved TRN: The result of each request distribution phase which collected by values collection phase is shown in this text box in decimal.

b) MIXED TRN: This text box shows the final result of each sent request; it means, the result of previous text box which is feed in MIXER module and makes final result in decimal.

c) Mapped (0,1) TRN: It shows final result of one request distribution phase mapped in zero and one intervals.

Moreover, in the result control group box, you can control the obtained results, it means, you can stop the request distribution phase, resume request distribution phase and also using collect button, you can save the obtained results (bit sequences) in a file for further tests.

Finally, Bit View group box, the bit sequences written in the file can be seen in very small scale, the reason of using this window is that, it helps you to see the degree of results randomness at a glance, i.e. when the results are not random, you can see some shapes repeated in this windows while in the case of randomness the contents of the window are completely scrambled and seen thrushes.

Appendix C: Application Source codes

Appendix C.A:

The code bellow defines prototypes and variables used in the implementation of Distributed-RNG protocol.

```
//****************************************************************************
//                   Secure True Random Number Generator
//                 In a Distributed System via Wireless LAN
//                        (Distributed_RNG protocol)
//****************************************************************************
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using NativeWifi;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using ListNetworkComputers;
using System.Management;
using System.Security.Cryptography;
using System.IO;

        Thread p1,p2,p3;
        private delegate void setDisplay(string Text);
        //------------------------New Wireless Variables---------------------------------//
        IPAddress[] addresslist = null;
        NetworkBrowser nb;
        WlanClient client = new WlanClient();
        string IP = "";
        //----------------------  Send and Receive Socket -------------------------------//
        //------------RECIEVE---------------//
        private const int listenPort = 11000;
        UdpClient listener = new UdpClient(listenPort);
        IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, listenPort);
        string received_data;
        byte[] receive_byte_array;
        //-------------SEND-----------------//
        Socket sending_socket = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram,ProtocolType.Udp);
        IPAddress send_to_address = IPAddress.Parse("169.254.255.255");
        IPEndPoint sending_end_point;
        //------------------------- Distributed-RNG frame -------------------------------//
        string Rframe = "";
        string hash = "";
        //------------------------- Receiver fields of frame ----------------------------//
        public byte[] type_payload = new byte[1];
        public byte[] sender = new byte[4];
        public byte[] payload = new byte[6];
        public byte[] nonce = new byte[1];
        public byte[] hashcode = new byte[7];
        bool responder_flag = true;
        byte[][] request_queue = new byte[10000][];
        int request_front = 0, request_rear = 0;
        byte[][] reply_queue = new byte[10000][];
        int reply_front = 0, reply_rear = 0;
        //---------------------------- MIXER Variables ----------------------------------//
        byte[] BB = new byte[32];
```

```
byte[] RB = new byte[8];
//--------------------------------------------------------------------------------//
TripleDESCryptoServiceProvider myTripleDES = new TripleDESCryptoServiceProvider();
DESCryptoServiceProvider myDES = new DESCryptoServiceProvider();
RC2CryptoServiceProvider myRC2 = new RC2CryptoServiceProvider();
//------------------------------- Results variables --------------------------------//
System.IO.StreamWriter File = null;
System.IO.StreamWriter File2 = null;
bool collect_flag = false;
ulong count = 0,avrage=0;
decimal MAX = 18446744073709551615;

System.Diagnostics.PerformanceCounter m_CPUCounter;
System.Diagnostics.PerformanceCounterPermissionEntryCollection t1;
System.Diagnostics.ProcessModuleCollection t2;
System.Diagnostics.ProcessThread t3;
System.Diagnostics.StackTrace t4;

Bitmap Bmp,Bmp2;
Graphics g,g2;
Rectangle rect,rect2;
int cellWidth;
string bit_ascii = "";
//-----------------------------------------------------------------------------------//
```

Appendix C.B:

The function below illustrates the nodes that are available in the network and it determines whether the nodes are accessible (direct neighbor or indirect neighbor). This function also shows the IP address and name of computers nodes.

```
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 = "Indirect neighbour";
        }

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

Appendix C.C:

The code below and its related functions implement values collection phase (sec 3.1.3) where receiver thread of both requester and repliers nodes run this part concurrency and separate the received message to both request message and reply message and their related routines. Firstly the code or receiving and dividing message is shown:

```
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);
        divider(receive_byte_array);
    }
}
string frame_read(byte[] frame,string type)
{
    switch (type)
    {
        case "type":
            return ((frame[0] & (0x08)) / 8).ToString();
        case "payload_length":
            return (frame[0] & (0x07)).ToString();
        case "sender":
         return frame[1].ToString() + "." + frame[2].ToString() + "." +
         frame[3].ToString() + "." + frame[4].ToString();
        default:
            return "";
        case "payload":
            return " ";
        case "nonce_and_hash":       //////nonce_and_hash together
            {
                string[] C_type = new string[5];
                C_type[1] = "DES";
                C_type[3] = "RC2";

                byte[] decrypt = new byte[frame.Length - 5];
                for (int i = 0; i < 16; i++)
                    decrypt[i] = frame[5 + i];
                 string all_decrypt = DecryptStringFromBytes(decrypt, C_type[((frame[0] &
                 (0xf0)) >> 4) - 1]);
                return all_decrypt;
            }
        case "Crypto_type":       //////nonce_and_hash together
            {
                string[] C_type = new string[5];
                C_type[1] = "DES";
                C_type[3] = "RC2";
                return C_type[((frame[0] & (0xf0)) >> 4) - 1];
            }
    }
}
void divider(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!");
            }
        }
```

Second, the codes bellow shows requests message routine where a random value is fed for requested message, in this case, authentication of received message is done.

```
        void responder()
        {
            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));
                        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);
                         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 following code is related to value collection phase for reply message (reply messages routine). Where the result of requester return back by repliers and are fed in MIXER module.

```
        void Reply_frame_routine()
        {
            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 = "";
                string[] C_type = new string[5];
                C_type[1] = "DES";
                C_type[3] = "RC2";

                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]);
                //----------------------------------------------------------------------//
                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];
                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)
                        for (int i = 0; i < pl; i++)
                            ADCs[(reply_number * pl) + i] = payload[i];
                    reply_number++;
                    replyN.Text = reply_number.ToString();
                }
            }
            decimal RawRND = OctetToDecimal(ADCs);
            hs.Text = RawRND.ToString();
            decimal TrueRND = OctetToDecimal(mixer(ADCs));
            hs2.Text = TrueRND.ToString();
            decimal T_RND = (TrueRND / MAX);
            TRND.Text = T_RND.ToString();
            cellWidth = 4;
            for (int i = 0; i < 128; i++)
            g.DrawString(bit_ascii[i].ToString(), new Font("Tahoma", 5.0f), Brushes.Black, i *
            cellWidth + (cellWidth / 2 - 7), (int)(count % 32000) * cellWidth + (cellWidth / 2 -
            6));
            pictureBox1.Image = Bmp;
            pictureBox2.Image = Bmp2;
            bit_ascii = "";
            count++;
            avrage += (ulong)reply_number;
            lcount.Text = count.ToString();
        }

        void Rect(PictureBox pic,PictureBox pic2, int Cells)
        {
            Bmp = new Bitmap(pic.Width, pic.Height);
            g = Graphics.FromImage(Bmp);
            g.FillRectangle(Brushes.White, 0, 0, pic.Width, pic.Height);
            Bmp2 = new Bitmap(pic2.Width, pic2.Height);
            g2 = Graphics.FromImage(Bmp2);
            g2.FillRectangle(Brushes.Black, 0, 0, pic2.Width, pic2.Height);
            pic.Image = Bmp;
            pic2.Image=Bmp2;
        }
```

Appendix C.D:

The function below is known as MIXER module in 3.1.4, where the collected results are fed into mixer module for obtaining the final random values.

```
    private byte[] mixer(byte[] VAL)
    {
        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(VAL));
        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]));
            bit_ascii += NumberToBinary(TrueRND[i]);
            RB[i] = TrueRND[i];
        }
        return TrueRND;
    }

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

Appendix C.E:

The function below implements request-distribution phase where a requester wishes a true random value (sec 3.1.2).

```
    private void Request_distribution(object sender, EventArgs e)
    {
        collect_flag = false;
        //-----------------------//
        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 "DES":
                Type_payload[0] = (byte)(Type_payload[0] | (0x10));
                break;
            case "RC2":
                Type_payload[0] = (byte)(Type_payload[0] | (0x20));
                break;
            default:
                MessageBox.Show("Choose crypto type!");
                break;
        }
        for (int i = 0; i < 4; i++)
            Sender[i] = Convert.ToByte(IP.Split('.')[i]);
        //--------------------------------------------------//
        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);
        //-----------------------//
        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);
        Broadcast_frame(send_buffer);
    }

    private void Broadcast_frame(byte[] send_buffer)
    {
        try
        {
            lifetime.Enabled = true;
            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);
        }

    }
```

Appendix C.F:

This part shows the function used for providing values from CPU information entropy resource. These values are taken from CPU information such as temperature.

```
string CPU_AND_thread_information(byte t)
        {
            Double temper = 0;
            decimal t1 = 0, t2 = 0, t3 = 0, t4 = 0;
            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;
                    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());
        }
```

Appendix C.G:

Some used function for implementation of Distributed-RNG protocol are shown as code and functions below. These functions are some necessary convert function, decode function, encryption and decryption function which are about DES, RC2 and DES-MAC.

```
        public string NumberToBinary(double num)
        {
            Int64 div = (Int64)num;
            string rbinary = "", binary = "";
            while (div >= 2)
            {
                rbinary += (div % 2);
                div = (div / 2);
            }
            rbinary += div;
            for (int i = 0; i < 8 - rbinary.Length; i++)
                binary += '0';
                for (int i = rbinary.Length - 1; i >= 0; i--)
                {
                    binary += rbinary[i];
                }
```

```
    return binary;
}
public string DecimalToBinary(decimal num)
{
    decimal div = num;
    string rbinary = "", binary = "";
    while (div >= 2)
    {
        rbinary += (div % 2);
        div = (div / 2);
    }
    rbinary += div;
    for (int i = rbinary.Length - 1; i >= 0; i--)
    {
        binary += rbinary[i];
    }
    return binary;
}
public double BinaryToNumber(string bin)
{
    double num = 0;

    for (int i = bin.Length-1; i >= 0; i--)
        if (bin[bin.Length - 1-i]=='1')
        num += Math.Pow(2, i);
        return num;
}
public byte[] NumberToOctet(double num, byte pl)
{
    double div = num;
    byte count = 0;
    byte[] p = new byte[pl];
    while (div >= 256)
    {
        if (count == pl) return p;
        p[(pl - 1) - count] = (byte)(div % 256);
        div = Math.Floor((div / 256));
        count++;
    }
    if (pl!=count) p[(pl - 1) - count] = (byte)div;
    return p;
}
public double OctetToNumber(byte[] pl)
{
    double num = 0;
    for (int i = pl.Length - 1; i >= 0; i--)
        num += pl[i] * Math.Pow(256, (pl.Length - 1 - i));
    return num;
}
public decimal OctetToDecimal(byte[] pl)
{
    decimal num = 0;
    for (int i = pl.Length - 1; i >= 0; i--)
        num += pl[i] * (decimal)Math.Pow(256, (pl.Length - 1 - i));
    return num;
}
public byte[] FromHexToOctet(string hexstring)
{
    int digit = 0;
    byte[] octet= new byte[16];
    for (int i = 0; i < hexstring.Length; i += 2)
    {
         digit = (Convert.ToByte(hexstring[i].ToString(), 16)) * 16 +
         Convert.ToByte(hexstring[i + 1].ToString(), 16);
        octet[i/2] = (byte)digit;
    }
    return octet;

string frame_read(byte[] frame,string type)
```

```
{
    switch (type)
    {
        case "type":
            return ((frame[0] & (0x08)) / 8).ToString();
        case "payload_length":
            return (frame[0] & (0x07)).ToString();
        case "sender":
        return frame[1].ToString() + "." + frame[2].ToString() + "." +
        frame[3].ToString() + "." + frame[4].ToString();
        default:
            return "";
        case "payload":
            return " ";
        case "nonce_and_hash":       //////nonce_and_hash together
            {
                string[] C_type = new string[5];
                C_type[1] = "DES";
                C_type[3] = "RC2";

                byte[] decrypt = new byte[frame.Length - 5];
                for (int i = 0; i < 16; i++)
                    decrypt[i] = frame[5 + i];
                 string all_decrypt = DecryptStringFromBytes(decrypt, C_type[((frame[0] &
                 (0xf0)) >> 4) - 1]);
                return all_decrypt;
            }
        case "Crypto_type":       //////nonce_and_hash together
            {
                string[] C_type = new string[5];
                C_type[1] = "DES";
                C_type[3] = "RC2";
                return C_type[((frame[0] & (0xf0)) >> 4) - 1];
            }
    }
}
static byte[] EncryptStringToBytes(string plainText,string Encr_type)
{
    byte[] encrypted;
    switch (Encr_type)
    {
        //------------------------------------------------------------------------//
        case "DES":
            using (DESCryptoServiceProvider tdsAlg = new DESCryptoServiceProvider())
            {
                tdsAlg.Key = Encoding.ASCII.GetBytes("Guess_it");
                tdsAlg.IV = Encoding.ASCII.GetBytes("big_bang");
                 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;
        //----------------------------------------------------------------------//
        case "RC2":
            using (RC2CryptoServiceProvider tdsAlg = new RC2CryptoServiceProvider())
            {
                tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess!?");
                tdsAlg.Mode = CipherMode.CBC;
                tdsAlg.IV = Encoding.ASCII.GetBytes("big_bang");
                 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;
        //-----------------------------------------------------------------------//
    }
}
static string DecryptStringFromBytes(byte[] cipherText,string Decr_type)
{
    string plaintext = null;
    switch (Decr_type)
    {
        //-----------------------------------------------------------------------//
        case "DES":
            using (DESCryptoServiceProvider tdsAlg = new DESCryptoServiceProvider())
            {
                tdsAlg.Key = Encoding.ASCII.GetBytes("Guess_it");
                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;
        //------------------------------------------------------------------------//
        case "RC2":
            using (RC2CryptoServiceProvider tdsAlg = new RC2CryptoServiceProvider())
            {
                tdsAlg.Key = Encoding.ASCII.GetBytes("_Who_Can_guess!?");
                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();
                        }
                    }
                }
```

Appendix C.H:

In this appendix the code of making wireless network profile and establishing connection is provided.

```
IPHostEntry myHostInfo = Dns.Resolve(Dns.GetHostName());

lblName.Text = Dns.GetHostName();
//--------------------------------------------------
foreach (WlanClient.WlanInterface wlanIface in client.Interfaces)
{
    string xml_RNG = "<?xml version=\"1.0\"?>\r\n<WLANProfile
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v1\">\r\n\t<name>RNG</name>\r\n\t<SSIDCo
nfig>\r\n\t\t<SSID>\r\n\t\t\t<hex>736361746572</hex>\r\n\t\t\t<name>RNG</name>\r\n\t\t</SSID>\r\n
\t\t<nonBroadcast>false</nonBroadcast>\r\n\t</SSIDConfig>\r\n\t<connectionType>IBSS</connectionTy
pe>\r\n\t<connectionMode>manual</connectionMode>\r\n\t<MSM>\r\n\t\t<security>\r\n\t\t\t<authEncry
ption>\r\n\t\t\t\t<authentication>open</authentication>\r\n\t\t\t\t<encryption>none</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_RNG, true);  //736361746572
wlanIface.Connect(Wlan.WlanConnectionMode.Profile, Wlan.Dot11BssType.Any, "RNG");
}
```