Design and Implement Pseudo Random Number Generator for Block Cipher Encryption Algorithm

Maytham M. Hammood [1], Tayseer S. Atia [2] , Ahmed Y. Yousuf [2]

[1] *Department of Computer Science , University of Tikrit , Tikrit , Iraq*

[2] *Department of Computer Science and Information System , University of Technology , Baghdad , Iraq*

(Received 15 / 5 / 2008 , Accepted 5 / 6 / 2008)

Abstract

The generation of pseudo-random numbers is an important and common task in computer programming. While cryptography and certain numerical algorithms require a very high degree of *apparent* randomness, many other operations only need a modest amount of unpredictability

Random numbers play an important role in the use of encryption for various network security applications. This paper design and implement a proposed Pseudo Random Number Generator for use in block cipher algorithm as a session key with an initialization vector(IV) to use them in cryptographic applications. The generator structure's makes use of HMAC algorithms with preprocess and post process stage to generate pseudo random sequence with length of ٢٥٦-bit.

Keywords: HMAC,RNG,PRNG,IV,SHA,IDEA,CFB.

Introduction

A **random number generator** (often abbreviated as RNG) is a computational or physical device designed to generate a sequence of numbers or symbols that lack any pattern, i.e. appear random. Computer-based systems for random number generation are widely used, but often fall short of this goal, though they may meet some statistical tests for randomness intended to ensure that they do not have any easily discernible patterns. Methods for generating random results have existed since ancient times, including dice, coin flipping, the shuffling of playing cards, the use of yarrow stalks in the I Ching, and many other techniques.

The many applications of randomness have led to many different methods for generating random data. These methods may vary as to how unpredictable or statistically random they are, and how quickly they can generate random numbers [1].

Almost all cryptographic protocols require the generation and use of secret values that must be unknown to attackers. For example, random number generators are required to generate public/private keypairs for asymmetric (public key) algorithms including RSA, DSA, and Diffie-Hellman. Keys for symmetric and hybrid cryptosystems are also generated randomly. RNGs are also used to create challenges, nonces (salts), padding bytes, and blinding values. The one time pad – the only provably-secure encryption system – uses as much key material as ciphertext and requires that the keystream be generated from a truly random process. Because security protocols rely on the unpredictability of the keys they use, random number generators for cryptographic applications must meet stringent requirements.

The most important is that attackers, including those who know the RNG design, must not be able to make any useful predictions about the RNG outputs. In particular, the apparent entropy of the RNG output should be as close as possible to the bit length. According to Shannon1, the entropy *H* of any message or state is:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

where *pi* is the probability of state *i* out of *n* possible states and *K* is an optional constant to provide units (e.g $1/\log_{(2)}$ bit). In the case of a random number generator that produces a *k*-bit binary result, *p*i is the probability that an output will equal *i*, where $0 \leq i \leq 2^k$ Thus, for a *perfect* random number generator, $pi = 2^{-k}$ and the entropy of the output is equal to *k* bits. This means that all possible outcomes are equally (un)likely, and on average the information present in the output cannot be represented in a sequence shorter than *k* bits. An RNG for cryptographic applications should appear to computationally-bounded adversaries to be close as possible to a perfect RNG [2].

Pseudorandomness

Traditionally, the concern in the generation of a sequence of allegedly random numbers as been that the sequence of numbers is random in some well-defined statistical sense. Uniform distribution, Independence and Unpredictability criteria are used to validate that a sequence of numbers is random. The distribution of numbers in the sequence should be uniform: that is, the frequency of occurrence of each of the numbers should be approximately the same. Independence means that no one value in the sequence can be inferred from the others. In applications such as reciprocal authentication and session key generation, the requirement is not so much that the sequence of numbers be statistically random but that the successive members of the sequence are unpredictable. With "true" random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable [3].

Proposed Pseudo Random Number generator

Pseudo-random number generators (PRNGs) are algorithms that can automatically create long runs (for example, millions of numbers long) with good random properties. The proposed Pseudorandom number generator classify as computational Pseudo-random number generators. Figure (1) shows the structure of the proposed random number generator this generator makes use of the HMAC structure found in [4] that is used to compute message authentication code with some modification to the internal structure and buffer value to accommodate our purpose.

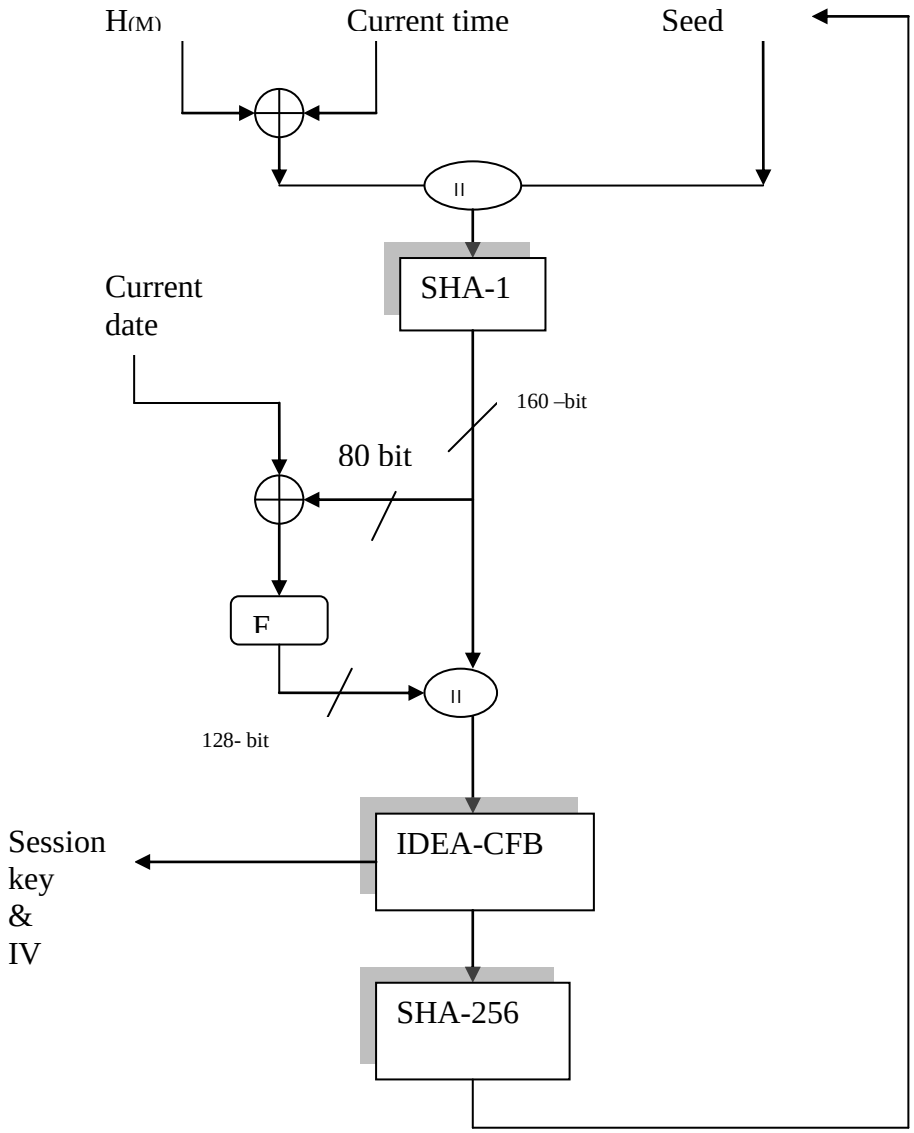


Figure (1) The Proposed Random Number Generator

This PRNG takes as input the following items:
H (m): 160-bit message digest computed using SHA-1.
Current time: 32-bit value repeated five times to be of length 160-bit.
Seed: 192-bit value stored in file called seed.bin.
Current date: 32-bit value repeated five times to be of length 160-bit.
And the following process:
SHA-1: Takes input from two sources the first one from the seed and the second one XOR operation between the current time and H (m).
And output 160-bit value.
F Function (Expansion function): This function takes input of length 80 bit and produces output of length 128-bit. It works as follows:

Divide the 80-bit value into four parts of 20-bit each. Each part is divided further divided into four parts of 5-bits each; these 5-bits are used as index to produce 8-bits from the Substitution Table shown below. As shown each index in this Table carries XOR operation between a constant value and the value of the 5-bits itself. These constant values are taken from the fractional part of the constant rate (22/7). These resulted each 8-bits from each 5-bits are processed further as shown in Figure (2) to produce an 8-bit value. Then for all the four parts the same processes are repeated. Finally the values of these parts are combined to produce the128-bit value

Table (1) Substitution Table

Index *value*	*Index* *value*	*Index* *value*
0 B0 ⊕ f9	11 B11 ⊕ 71	22 B22⊕ 0f
1 B1 ⊕ 7a	12 B12 ⊕ 12	23 B23 ⊕ 24
2 B2 ⊕ e4	13 B13 ⊕ fb	24 B24 ⊕ cd
3 B3 ⊕ 98	14 B14 ⊕ 1c	25 B25 ⊕ 3c
4 B4 ⊕ 7b	15 B15 ⊕ 2e	26 B26 ⊕ fc
5 B5 ⊕ fe	16 B16 ⊕ 4b	27 B27 ⊕ 70
6 B6 ⊕19	17 B17 ⊕ 1e	28 B28 ⊕ 48
7 B7 ⊕ 74	18 B18 ⊕ 78	٢٩ B29 ⊕ c8
8 B8 ⊕ 8f	19 B19 ⊕ f5	٣٠ B30 ⊕ ee
9 B9 ⊕ 82	20 B20 ⊕ ff	٣١ B31 ⊕ ad
10 B10 ⊕ 8d	21 B21 ⊕ 9e	

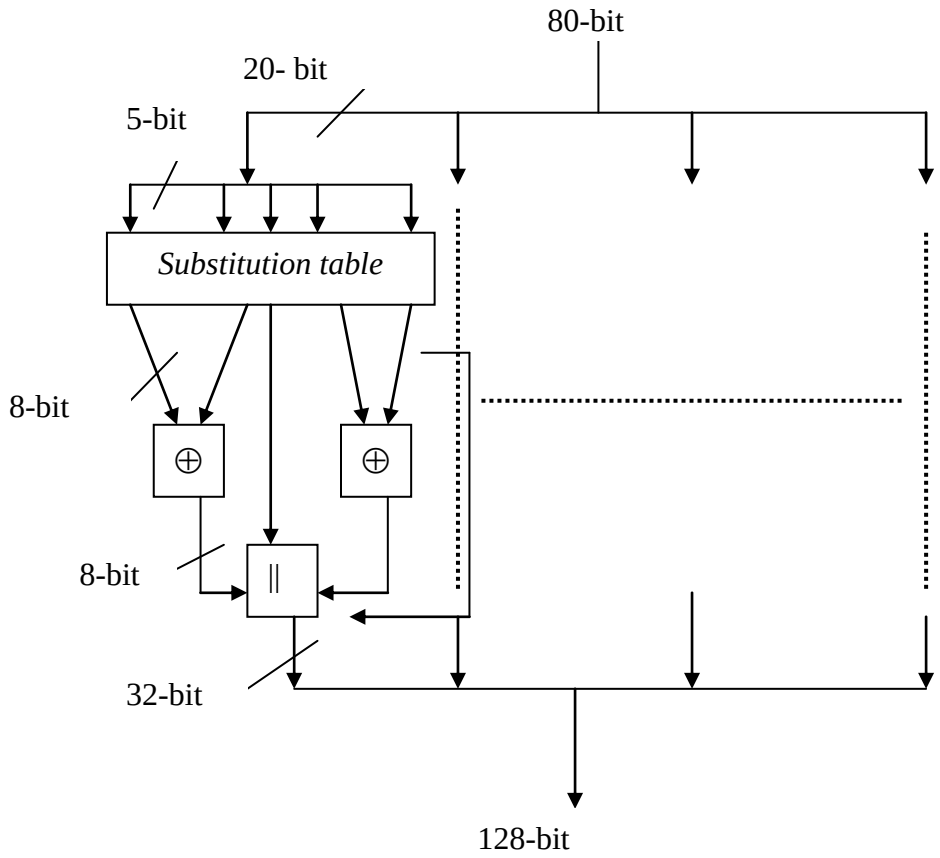


Figure (2) F Function details

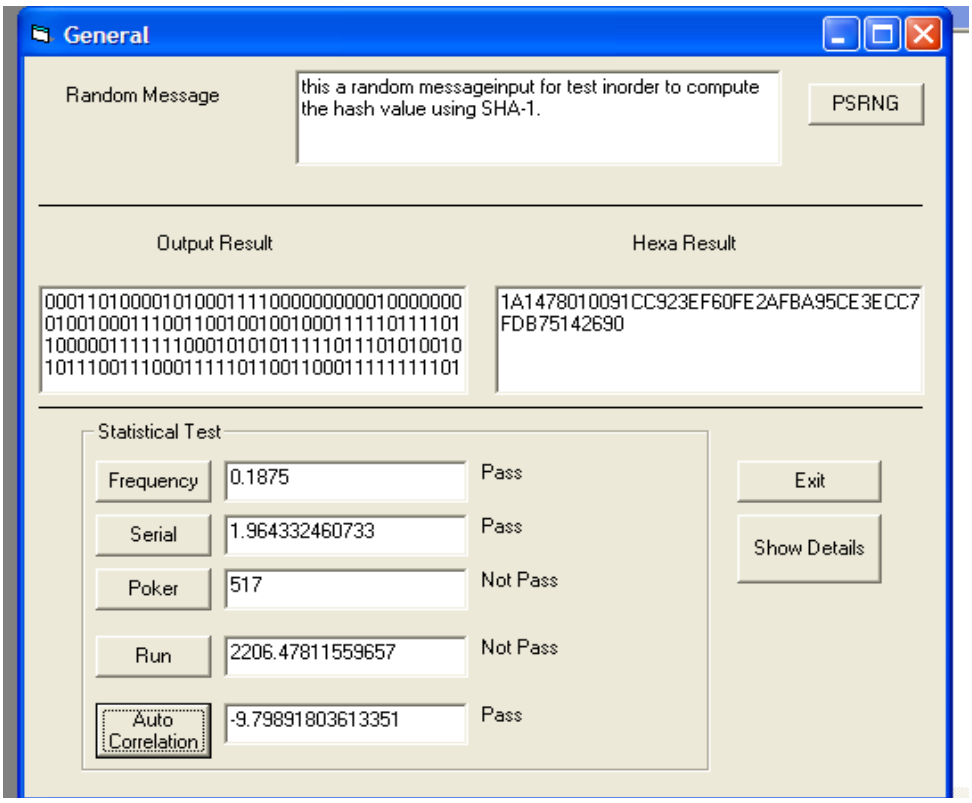
IDEA-CFB: Takes as input 288-bit combined from the SHA-1 output and the F function output and outputs 256 bit as a session key and IV.

SHA-256: Takes the output of the IDEA-CFB to produce new seed for the next generation

Experimental Result

This section present some examples that explain the implementation of the proposed pseudo random number generator, also show the initialization needed, finally, statistical test was computed to judge the correctness of the generated sequence, Figure (٣) shows the input data to generator and the output generated sequence with the evaluated test

.



As shown in Figure(٣) the evaluated test for the generated sequence are pass for the frequency, serial, auto correlation test and fail in poker, run test this mean that the generated sequence is acceptable sequence since it can pass three tests fro five tests.

Conclusion

This paper present pseudo random number generator to generate pseudo random sequence of length ٢٥٦-bit to be used with block cipher algorithm as a session key or the secret key for cryptographic application. The generator structure consists of preprocess stage to input random data to use in initialization stage, the generate stage which depend on the structure HMAC algorithm that yield the pseudo random sequence and post process stage to generate random seed to be used in next generation, the output sequence was tested using the statistical tests to check the correctness of the generated sequence and its show that its pass three testes from five tests each time which give an indication to accept the sequence as a correct sequence.

References

1.http://en.wikipedia.org/wiki/Random_number_generation"

2. THE INTEL® RANDOM NUMBER GENERATOR CRYPTOGRAPHY RESEARCH, INC. WHITE PAPER PREPARED FOR INTEL CORPORATION Benjamin Jun and Paul Kocher April 22, 1999

3. Stallings W., "NETWORK Security Essentials: Applications and standards", Prentice Hall, New Jersey, 2000.

4. N. Freed, N. Borenstein, "RFC2045: MIME part one: Format of Internet Message Bodies", Standard Track, Network working Group.
Site [www.fags.org/rfcs/rfc2045.html].

الملخص

عملية توليد الارقام العشوائية تعتبر ضرورية ومهمة وشائعة في برمجة الحاسوب. طالما ان التجفير وبعض الخوارزميات العددية تتطلب درجة عالية من العشوائية فان هناك عمليات تحتاج مقدار اكبر من عدم القدرة على التنبأ. الارقام العشوائية تلعب دورا مهما في التجفير لمختلف تطبيقات امن الشبكات. هذا البحث يصمم وينفذ مولد ارقام عشوائية مقترح لاستخدامه في خوارزميات التجفير الكتلي كمفتاح سري مع قيمة تهيئة ابتدائية IV ليمكن استخدامهما في تطبيقات التجفير هيكل المولد المقترح يستفيد من خوارزمية HMAC مع مراحل معالجة قبلية وبعدية لتوليد سلسلة شبه عشوائية بطول ٢٥٦ بت.