A Secured Key Generation Scheme Using Enhanced Entropy

M.S. Irfan Ahmed
Asst. Professor, VLB Engineering College, Coimbatore

E.R. Naganathan
Reader, Computer Science Department
Alagappa University, Karaikudi

Summary
This paper is an efficient and secure way to generate true random numbers. With an ever increasing repertoire of applications that incorporate Randomness, there arises a need to ensure that the Randomness is *truly* Random at all tenable circumstances. Through this paper we formulate a method to generate true random numbers by collecting entropy from devices such as mouse, keyboard and hard disk. Here, rather than using the data supplied by the user we concentrate on the timestamps generated by the user interaction which is an efficient way of generating the random numbers. The time stamps that are collected are stored in a separate text file. While processing, the contents of this text file is XOR'ed with the contents of another file whose contents include the current state of the system and the usage of the system by the user. The output from the XOR function is now given as input to the Secure Hash Algorithm (SHA), a one-way hashing algorithm that gives an output of 160 bits. This output is written in a separate file which is the entropy pool, from which the key of required length is retrieved as per the requirement of the application. The random numbers that are being used at present are generated by hardware random number generators which are almost costly and infeasible for many situations, one area where the random number generators using "entropy" gain an edge. Experimental results show that this method generates in an efficient manner, true random numbers which can be applied for long term key generation, seeding cryptographic random numbers used by websites hosting online poker games and seeding pseudo random numbers for statistical testing.

1. Introduction

Within security today the encryption methodologies and cryptographic techniques have exploded in both complexity and usage. The application of these technologies is to keep secrets safe and secure, but there are pitfalls involved with utilizing cryptography. This paper focuses on one of the vital components used in various security related technologies namely "randomness". This component is by nature, complex and easily misunderstood. One may say that randomness plays a "key" part in most cryptosystems today.

"A sequence of independent numbers with a specified distribution, each number being obtained by chance and not influenced by the other numbers in the sequence".

The definition specifies two important conditions.

- The values are uniformly distributed over a defined interval or set.
- It is impossible to predict future values based on past or present ones.

The next question will be, "Why do we need Random numbers?" The answer is **"Statistical and Security-critical applications often require well-chosen Random numbers for purposes ranging from cryptographic key-generation to shuffling a virtual deck of cards".** Thus the varied need of Random numbers forces the fact - "Random number generation is not trivial".

2. Existing Methodology

Most encryption algorithms require a source of random data, even some symmetric ciphers (where the secret is shared), either to generate new private/public key pairs, for session keys, for padding, or for other reasons. Most computers do not have a hardware based random number generator (RNG) available, so programmers have had to resort to software-based techniques, to generate random numbers as best they can. Because these random numbers are generated in software, they are very rarely truly random, they are typically *pseudo random* (that is they appear random, but are not totally random)[2]. To generate random data you need a source of entropy, or random input.

The reason why this is important is that, for example, the 168 bit 3DES encryption of network packets might not be working as advertised if the random data it uses to perform the encryption isn't truly random. If sensitive corporate data is being moved over the VPN, a good hardware based RNG is very important. Additionally most software based RNG's cannot create a lot of good data (most hardware based RNG's as well have limits), so if the machine is carrying a heavy crypto workload (say multiple tunnels, or many SSL based connections for an e-commerce site) we might need one of the higher end, more expensive hardware based RNG's. Properly deployed hardware RNG is a good security enhancement.

Entropy Sources

3. Random Number Generators

There are essentially three classes of Random number generators.

INSECURE RANDOM NUMBER GENERATORS

These are the non-cryptographic pseudo-random number generators. These start with an initial value, usually called the "SEED", and use that seed to produce a stream of numbers which are reproducible when started again with the same seed.

CRYPTOGRAPHIC PSEUDO-RANDOM NUMBER GENERATORS

These take a single secure seed and produce as many unguessable random numbers from that seed as necessary. These need a high degree of Entropy for them to be securely seeded.

ENTROPY HARVESTERS

These try to gather Entropy to seed a cryptographic pseudo-random number generator. The combination of an Entropy harvester with a Cryptographic pseudo-random number generator could be a good practical compromise.

4. Pseudorandom Number Generators (PRNG's)

The most widely used technique for pseudorandom number generation is an algorithm first proposed by Lehmer, which is known as the linear congruential method[1] . The algorithm is parameterized with four numbers, as follows:

m	the modulus	$m > 0$
a	the multiplier	$0 \le a < m$
c	the increment	$0 \le c < m$
X_0	the starting value, or seed	$o \le X_0 < m$

The sequence of random numbers $\{X_n\}$ is obtained via the following iterative equation: [6]

$$X_{n+1} = (aX_n + c) \bmod m$$

If m, a, c and X_0 are integers, then this technique will produce a sequence of integers with each integer in the range $0 \le X_n < m$.

The selection of values for a, c, m is critical in developing a good random number generator. The value of m should be large, so that there is the potential for producing a large series of distinct random numbers [6]. A common criterion is that m is nearly equal to the maximum representable nonnegative integer for a given computer. Thus the value of m equal to 2^{31} is chosen.

5. Cryptographically Generated Random Numbers

Cyclic Encryption

In this case the procedure is used to generate session keys from a master key. A counter with period N provides input to the encryption logic. For example, if 56-bit DES keys are to be produced, then a counter with period 2^56 can be used[3]. After each key is produced, the counter is incremented by 1.

ANSI X9.17 PRNG

One of the strongest PRNG's is specified in ANSI X9.17.A number of applications employ this technique, including financial security applications and PGP.

The ingredients are as follows:

- Input
- Keys

6. Methods For Gathering Entropy

There are essentially two methods for Random number generation with Entropy collection.

One method is to gather enough Entropy to securely seed a cryptographic pseudo-random number generator.

Several issues arise when this method of Random number generation is tried.

- The first issue is the **amount of Entropy** needed to securely seed a cryptographic generator. The shortest possible answer is that as much Entropy is to be given as the Random number generator can accept[5]. It should be noted that, for example, a 256-bit number generated with a 56-bit Entropy seed would contain no more Entropy than what a 56-bit number would have had. It is found incredibly hard to determine what actual amount of Entropy is required? This is similar to that of exactly estimating the Entropy in a set of data, discussed earlier, which is found as essentially impossible.

- Second issue is **to obtain a fixed sized seed** as required by the Random Number generators with maximum available Entropy from a collected set of Entropy. This is essentially the process of turning the data with Entropy into a seed. Whitening can effectively take care of this issue.

- Third, the **period of the random number generator** is to be considered if Security is a concern, where the period refers to the *number of unique Random numbers generated with a given seed*. The period can also be defined as *the sequence generated before any repetition occurs*. Usually, this period is required to be as long as it possibly can be, for example, a period of

2^{64} bytes before repetition can be considered enough if the number of unnoticed guesses that would be made to predict the Random number, are supposed to be not that much larger.

7. Human Interaction

One form of credential is monitoring the frequency of typing or moving the mouse; this is one type of entropy. Actual typing of characters; can also be used as there will be some distinct patterns. Passwords and random input from users could also be used, but all in all the entropy gained is quite low.

8. Environmental Gathering

An instance of environment gathering could be using a microphone to collect ambient noise from a room or even a remote location. It is best for only one machine to exist utilizing this form in a given environment, the danger of more is two may collect the same data and provide patterns.

9 Computer States

The suggestion for utilizing computer states usually focus around use of combining interrupts with microsecond clock data. One interesting idea is to utilize the inherent randomness of the video buffer or possibly the video image changes that appear under the mouse pointer. In addition there are other radical ideas involving the I/O and hard disk, such as measuring turbulence inside a drive that is spinning, that data is currently difficult to collect but all the same it provides entropy. Intel decided that post 8xx CPU support chips would have a random generator built in allowing for the collection of random data caused by frequency chip noise.

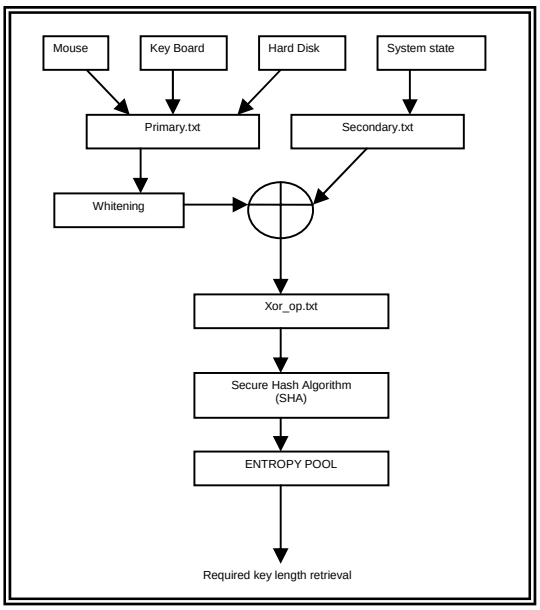
10. Quantum Events

This is a more difficult source of information, splitting atoms and other quantum type events usually only occur in a controlled scientific research facility.

11. Proposed Model

- The mouse coordinates and the timestamps of the mouse movements are collected and are written as long integers in a file called “primary.txt”.
- Similarly the keyboard timestamps along with characters pressed are also collected and are again converted as long integers and are written in the file “primary.txt”.
- The user interaction timestamps when a file is being read from the hard disk is also collected and again similarly written to the same file.
- Then Batch processing, namely processing in blocks is done in the inputs from mouse and keyboard.
- After that, the process of “whitening” is done so as to increase the entropy available in each bit.
- Then the system state and the user’s usage of the system are written in a file called “ntuser.txt” by the operating system. Instead of directly using this file we copy this file into another text file known as “system.txt”.
- Then the contents of the two files namely, “primary.txt” and “sytem.txt” are XOR’ed and the output is converted into long integers and written into a file called “xor_op.txt”.
- Then this file is fed as an input to the Secure Hash Algorithm (SHA), a one-way hashing function which produces a 160-bit output. This is done so as to hide the internal state of the pool. The output from the SHA is written into a file called as “pool.txt” which acts as the entropy pool.
- Then the retrieval of keys can be done for any length. The length of the keys retrieved depends on the requirement of the application.

The subtle point to be noted is that the Entropy *does not* come from the “data” supplied by the User interaction rather it comes from the corresponding “time-stamps”. The simple reason is that if it is actually the data that is considered for Entropy, then any user who has access to the system, not necessarily an administrator, can dump in his/her own data that at the end decides the series of bytes to be added to the Entropy pool. The User who caused it *overrides* the ultimate requirement - “unpredictability”.



12. Results and Discussions

When we hit the ESC key, the next screen opens where we collect the mouse entropy

Figure-1 Mouse entropy collection

3) If we want to collect keyboard entropy again, we need to press the ESC key.
4) Then when we press the RIGHT mouse button, we can collect the Hard disk entropy.

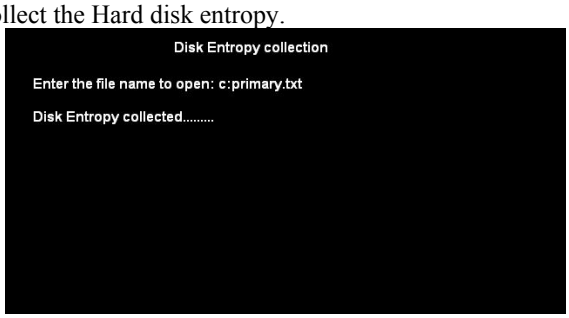


Figure-2 Disk Entropy Collection

5) Then the contents are sent to the XOR algorithm after which it is fed to the SHA.

Figure-3 XORing and giving input to SHA

6) Then the key is retrieved as per the requirements of application.

Though this kind of guessing Keystroke timing need effective brute force attacks and duly careful research in Keystroke studies, the possibility of such an attack is not infeasible. This requires a detection of such an attack and a subsequent counter measure by introducing further Randomness[7]. Suppose that the adversary somehow collected information about what actually is going to be typed. Consider the typed words to be "evil" and "good". The "| =" represent the guessed timing interval between Keystrokes. The antagonist can guess the Keystroke timings as follows.

1) "evil"

Left hand strokes: 'e', 'v'.
Right hand strokes: 'i', 'l'.

e press |
e release | =========
v press | ================
i press | =====
v release | ==
i release | =====
l press | ========
l release | ============

2) "good"

Left hand strokes: 'g', 'd'.
Right hand strokes: 'o'.

g press |
g release | =========
o press | ====
o release | ======
o press | ========
d press | =========
o release | ========
d release |
===============

13. Conclusion

Through our paper, we have suggested an efficient way to generate secure true random numbers. This collection of time stamps and comparing them offers an effective solution for "key stroke timing' attack which has been a problem to all the other similar random number generators. The usage of SHA to hash the contents before retrieving them makes guessing of the internal state of the entropy pool almost next to impossible. Also, as the system state is taken into consideration it is absolutely impossible to guess the key that is being generated. Therefore our paper upholds and follows strictly the main characteristic of a random number namely, "unpredictability".

References

[1] Seth Hardy, 2004," Pseudo Random Number generation, Entropy Harvesting and Provable security in Linux.
[2] S.Micali, and C.P.Schnorr, "Efficient, perfect random number generators" Crypto'88 conference Proceedings.
[3] Entacher, Karl. 1998. Bad Subsequences of Well-Known Linear Congruential Pseudorandom Number Generators. ACM transactions on Modeling and Computer Simulation. vol. 8, no. 1, pp.61-70.
[4] Fog, A. 2001. Pseudo random number generators. http://www.agner.org/random.
[5] Chris Thorn,2003, "Randomness and Entropy- An Introduction", SANS Institute.
[6] Leif Svalgaard, 2003, "Generating a truly Random Number"
[7] William Stallings," Cryptography and Network Security, Principles and Practices", Third Edition.