Randomness in Cryptography

Seminararbeit
(Bakkalaureatsseminar)

in der Studienrichtung

Technische Informatik

Angefertigt am *Institute for Information Processing and Microprocessor Technology*

Betreuung:

o. Univ.-Prof. Dr. Jörg Mühlbacher
DI Rudolf Hörmanseder

Eingereicht von:

Sonnleitner Erik

Linz, October 2007

Contents

Abstract

Cryptographic routines and algorithms often rely on randomness, which is an essential fundament, especially in key-generation applications. This paper discusses how pseudo and real random numbers may be generated and how threatening unconcerness due to lack of entropy may seriously risk security.

1 Introduction

1.1 The need for randomness

The science of computation and computer programming defines a scholarship based on absolute accuracy and precision. What results computers are expected to provide, is strongly constrained by a classified order of a finite set of instructions in dependency on the applied architecture and computational platform.

This fact clearly represents the exigence and importance of algorithmic determinism, where randomness is truly out of place. Moreover, randomness may not only be handled as objectionable non-deterministic occurence but also as an hard-to-produce scientific subject, where strictly deterministic algorithms should be able to produce seemingly random data.

Nevertheless, randomness has proven to be very powerful in a wide range of applications, as according to [HL93], like in chaos theory, pattern recognition, quantum mechanics and statistics. Futhermore, randomness constitutes the fundamental essentials of security and secure communication in the broad scope of applications in nowadays cryptography.

As particular example, the *One-time-pad* is the only known method of encipherment which has been proven to be ultimately secure when total randomness is preconditioned.

1.2 Formal definitions

When speaking about randomness, we commonly mean a *sequence of independent random numbers*, where each number was obtained completely random and has absolutely no correlation between any other numbers inside the sequence.

In computer science, we reduce the set of possible values for one digit to one and zero, so let our alphabet $A = \{0, 1\}$. In an N-bit random number sequence, the probability of the resulting value at any position n inside the bit string is exactly $1/|A| = 0.5 = 50\%$, completely indepedent of which digit we're talking about.

Stongly related to the term of randomness, we'll shortly discuss the forms of predictability. The importance of randomness is actually given by the fact, that true random sequences are completely unpredictable, which is precisly what particular fields of applications need for calculations.

As shown in [GNP06], a fully random process can be called unpredictable, but an unpredictable process doesn't mandatory has to be completely random. The first implication should be considered as trivial; the non-existence of the reverse implication may be shown by the following example: Let the numbers of a die be represented in bit-streams, so the possible results of rolling a die can only be elements of the set $\{001, 010, 011, 100, 101, 110\}$. Each time the die has been rolled, the result is completely independent of all previous and subsequent rolls. Notwithstanding, the consequential sequence of values is closely associated with a non-uniform distribution of our co-domain (in other words, we will never ever get the die to result in a value of the set $\{000, 111\}$).

Entropy Primiparously introduced by C. Shannon in 1948, *entropy* is defined as the quantity of information within a given block of data, and can mathematically be described as the "negative logarithm of the probability of the process's most likely output" ([GNP06]).

Therefore, maximum entropy assures complete (uniformly distributed) randomness and hence unpredictability. An interesting value is given by the entropy rate J:

$$J = -\sum_{n=0}^{n<|\langle X\rangle|} (p_{x_n} \log_2 p_{x_n}) \cdot \mid \langle X\rangle \mid^{-1}$$

providing the percentual average of unforeseeability per bit, taken from the bit-stream $\langle X\rangle$ where p_{xn} is the probability p of occurance of element n in stream x.

1.3 Motivation for non-deterministic cryptographic applications

The one-time-pad is the one and only perfectly secure cryptographic protocol, which implies that

1. (the shared secret is only known by trusted communication partners,)
2. the shared secret is truly random, and
3. every element of the sequence which defines the shared secret is only to be used exactly once.

In every symmetric cryptographic algorithm we assume 1) as a prerequisite. The weight of 2) and possible endangerments due to lack of randomness are discussed later.

The secureness of one-time-pads deeply relies on 3), because every attempt of decoding a ciphered bit-stream sequence will result in every possible plaintext of the length $|N|$, containing all possible permutations of $|N|$ values, as long as 3) is redeemed.

Another relevant field of activity in cryptography are stream ciphers, which basically utilize an initial random (*seed*) value to iteratively construct an enormous amount of pseudo-random successors of the chosen seed by algebraic and consequently fully deterministic processes. Therefore, the security of the ciphered stream totally depends on the randomness of the seed. Any possible influence which decreases the entropy of the seed value will undoubtedly prune the number of sequences which are to be examined against the ciphered stream when attacking via brute force methods.

2 Algebraical pseudo-randomness

2.1 Fully algorithmic generation of random sequences

2.1.1 Linear congruential generators

The matter of fact that an entirly deterministic process cannot produce randomness, forces us to survey mathematical approaches which result in *pseudo-random* sequences. In addition, those sequences are retrospectively to be testified to be uniformly distributed.

One of the first efforts on producing random numbers are *Linear Congruential Generators* ([Knu98]), which commonly underlie a recursive definition:

$$\langle X_n \rangle := x_n = (ax_{n-1} + b) \mod m$$

where m acts as module, a as multiplier, b as increment and n as counter variable.

x_n is taken as the remainder $\mod m$ (which defines the maximal value + 1) and represents the n-th element of the pseudo-random sequence. A linear congruential sequence will always be liable to a maximum period $\phi < m + 1$ as a repeating cycle of numbers, unattached to which seed number x_0 has been set.

Let $a = b = 3$, and let $m = 11$, and let the initial value x_{n-1} be zero, then $\langle X_n \rangle$ obtains a maximum period of 5, by reason of $\langle X_n \rangle = \langle \ldots 3, 1, 6, 10, 0 \ldots \rangle$.

Linear congruential generators, as well as quadratic and cubic congruential sequences like

$$x_n = (ax_{n-1}^2 + b_x n - 1 + c) \mod m$$

or

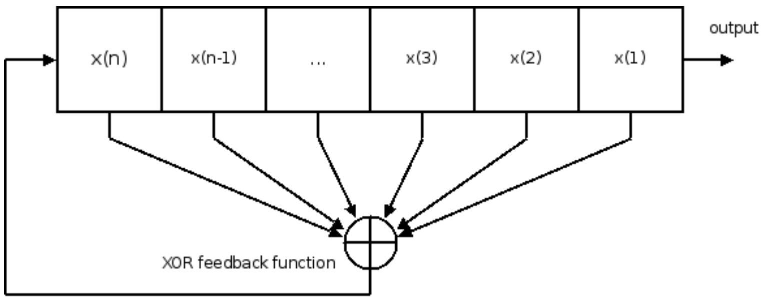
$$x_n = (ax_{n-1}^3 + bx_{n-1}^2 + cx_{n-1} + d) \mod m$$

and even polynomial ones with and without unknown parameters have been proven to be insecure due to predictability ([Sch05]) and therefore unusable for cryptography.

2.1.2 Linear feedback shift registers

Still heavily in use for a rich spectrum of utilisation, Linear Feedback Shift Registers (*LFSR*) incorporate a very useful alternative to getting uniformly distributed bit-sequences.

A shift-register is used to represent a bit vector $X_n = (x_n, \ldots, x_1)$ which gets shifted along the predefined direction (right, in our example) in conjunction with every instruction step, so that $x_n \to x_{n-1}, \ldots, x_2 \to x_1$ and x_1 of instruction step X_{n-1} becomes the newest output bit of our pseudo-random stream, before executing $X_{n-1+1} = X_n$ like shown in detail in [Ert03].



The initial assignment setup of the register describes the sequence, we will henceforth entitle as *key*. The feedback function is instantiated as composite configuration of particular bits inside the register which are (in its most elementary specification) associated by mutual exclusiveness, also known as XOR. The mapping of exclusively disjuncted bits is called the *tapping sequence*.

LFSRs may produce sequences by a maximum period of $2^n - 1$ (omitting a key holding only zeros). To force the creation of a maximum-period output sequence, the tapping sequence has to be an irreducible primitive polynomial modulo 2, according to [Sch05], where a rich list of such polynomials can be found.

2.1.3 Geffe Generators

In its simplest form, LFSRs can already be broken after $2n$ output bits. This fact leads to more complicated structures like the Geffe-Generator, which consists of two or three (N) LFSRs providing the input of a two-to-one multiplexer, and one LFSR as multiplexer control unit which produces bits according to the following forumla:

$$b = (x_1 \wedge x_2) \oplus (\neg x_1 \wedge x_3)$$

The conception of the Geffe-Generator makes sense, if

- N LFSRs are used with maximum periods, and
- the period-values of the LFSRs are coprime.

If the preceding conditions are given, the Geffe Generator will have a long period and linear complexity.

2.1.4 Advanced crypto-analysis of the Geffe Generator

Brute-forcing all possible key-values will result in a maximum of 2^n, and an average of $2^n \cdot \frac{1}{2}$ values which are to be tested.

As shown in [Luc05], there is a way to greatly decrease this number: If the keys are processed in order of the probability they may occur:

- The first n_1 bits of the stream S (the first LFSR) give the most probable sequences for key candidates K^* of the key K..
- Testify all possible keys K^*, beginning with Hamming-distance 1 of K^*, and increase the number continuously.
- In average, the final key K has a Hamming-distance of $n/4$ of K^*.
- The number of steps may therefore be reduced to

$$\binom{n_1}{0.75 \cdot n_1}$$

If the secret key has a strength of 120 bit and is built upon 3 40-bit registers, we have $n_1 = n_2 = n_3 = n_4$. Therefore we have a maximum number of 2^{40} keys to be tested, and an average of 2^{39} keys. The key-reduction process now may decrease the number of trials to

$$\binom{40}{30} = \frac{40!}{30! \cdot 10!} \approx 2^{29.7} \approx 2^{30}$$

which means a reduction to 1/10000.

2.2 User-interaction driven generation of random sequences

As we have already seen, pure algebaric pseudo-random number generators can not guarantee perfect security. Besides true unbiased random data sources, which are to be discussed later, hybrid randomisation systems are commonly used for gathering quite strong entropy.

A hybrid random sequence generator is classified by using raw data which is not completely random, but quite non-traceable in combination with ordinary randomisation algorithms and sustains upon

1. direct user interaction,
2. meta data about captured user interaction,
3. various statistics, and
4. commonly used non-human-interface I/O-devices.

The gathering of data from those near-random sources results in a random number pool, representing the main input conglomerate for further arithmetical operations from which correlations are tried to be suppressed from the output stream.

Category 1) primarily means keystrokes and mouse movements. The user is forced to provide random data via ordinary input devices. Because of the human understanding and habits when using such devices, full entropy cannot be reached, altough the unpredictability is better than using system internal values like process IDs. Nevertheless, inter-keystroke timings are already used for biometric identification and authentication software modules, clearly indicating a certain connection between the person giving the input strokes and the resulting output stream.

Meta data about captured user interaction is mostly done with timing parameters between keystrokes and mouse movements, as well as their derivation (acceleration); these may also be combined with network and process statistics, but latter are not to be used on their own.

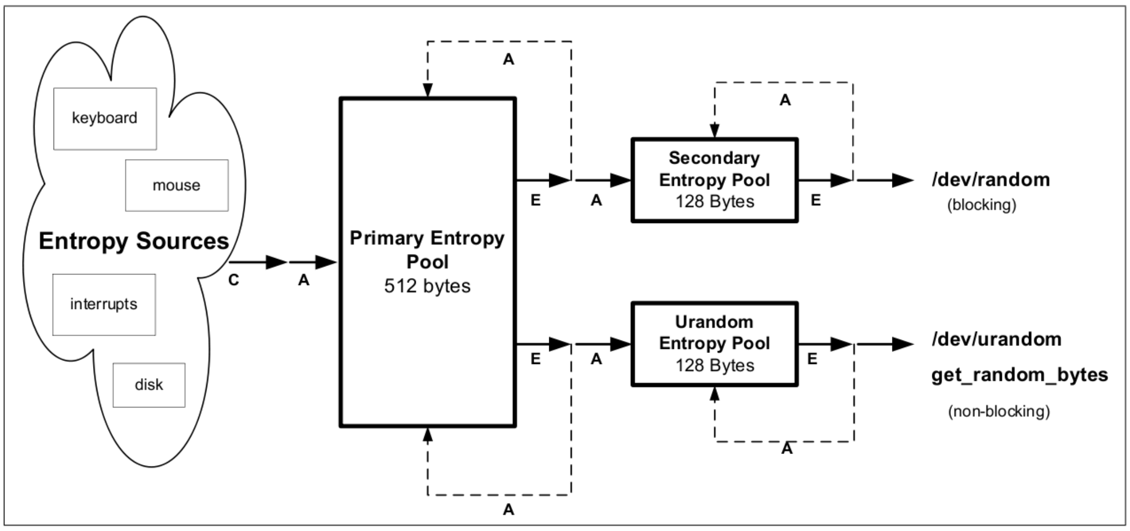
Finally, also non-human-interface I/O-devices may be used for gathering entropy, like microphones and webcams. The quality of possible randomness gathered from such devices, strongly depends on the environment they are applied in. If a camera only catches fairly still images of the user sitting in front of the computer, or even worse, just poiting at a wall, it can be classified as mostly useless due to lack of entropy. On the other hand, microphones may pick up interesting data flows, especially when certain frequencies, which are liable to have a high rate of entropy, are filtered [Ran95].

2.3 Case study: Analysis of Linux `/dev/random`

The Linux random number generator (LRNG) basically consists of three asynchronous procedures ([?]):

- Entropy collection,

- filling entropy-pools, and
- producing output if requested and updating the pools.



The Linux kernel gathers entropy by using information out of several internal resources. Namely, these are mouse and keyboard activities, diso I/O-operations and system interrupts. Every collected unit of information is carried by a timing variable, respresenting a 32bit value of Jiffies (dependent on the kernel configuration, a Jiffie may last from 1/250 - 1/1000 seconds).

The timings between events are used to calculate an estimation of the entropy which is provided by the gathered information. Based on the result of this calculations, more or less data may be added to the entropy pool.

While the `/dev/random` device produces strong randomness, its amount of entropy is very limited and if there is no entropy available anymore, further requests of random values are being blocked. On the other hand, `/dev/urandom` uses quite the same entropy resources as `/dev/random`, but recreates randomness by hashing the gathered entropy via the SHA-1 algorithm. Therefore, the latter is always capable of producing random numbers, but isn't that secure than it's limited pendant `/dev/random`.

Linux in embedded devices When Linux is used in embedded devices like PDAs or routers, most of the given resources may be unavailable to the system, because neither harddisks, nor mice or keyboards are connected to the system. OpenWRT, for example, is a Linux distribution for routers. The only available resource is given by network interrupts. Moreover, that current state of the random number generator (which is normally stored within a file while the system is shut down, and used as a seed at the next boot) often cannot be stored. Especially after hardware resets, the amount of entropy is very rare.

If an attackers is able to capture all network traffic (e.g. from wireless routers) since the device has been started up, it may be possible to reconstruct the initial state of the device and hence allows recalculation of the seed values and further entropy calculations.

When speaking about crypography, the entropy needed to produce SSL-certificates implicitely also relies on the algorithms described above, and may hence be created in an insecure environment with very low entropy, which largenes the contact surface for attackers.

3 True unbiased bitstreams in ordinary computers

3.1 Quantum mechanical physical randomness

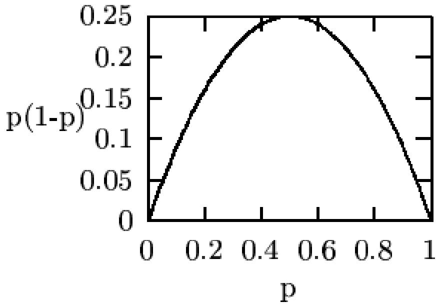
Atomic decay represents a good source for true randomness. Emissions, recognized by a Geiger counter can be used to gather random bit sequences by measuring the quantity of emissions within a fixed time-frame and saving the least significant bit ([Sch05]).

[SR07] exactly describes the construction of a true quantum random number generator, whose randomness relies on the "quantum physical process of photonic emission in semiconductors and subsequent detection of photoeletric effect". The image below shows a prototype of the generator, providing up to 12Mbit/s of random data, transfered via USB 2.0.

3.2 Randomness through ordinary disk drives

Firstly introduced on the 14th Annual International Cryptology Conference, even ordinary hard-disk drives may offer true random data. According to [DIF94], several tests have shown that rotational speed, disk spacing and cooling flow affect the disk flow. The algorithm basically measures the access-times for reading single sectors on the disk and uses the variation of these times as input source, which still deserves the removal of correlations. The occasion why the resulting streams are supposed to be random, is because the main reason for disk-jitter is found in natural air turbulences inside the drive.

As shown in [ES00], Maxtor has developed a related attempt for using hard disks as random source, by inspecting the magnetic readback signal amplitude of the servo bursts inside the harddisk, as well as the quality of the data stream as seen by the readback signal decoder (raw data), and reaches up to 835kbps.



3.3 Balancing asymmetric occurences

When a bit-stream has been collected, we have to focus on eliminating possibly occurring asymmetric appearences. A very effective method is the *Neumann-Filter*, which simply maps bit-pairs to either nothing (ϵ) or the most significant bit:

$$f = \begin{cases} 00 \to \epsilon \\ 11 \to \epsilon \\ 01 \to 0 \\ 10 \to 1 \end{cases}$$

Let p be the possibility of any bit x of the stream to be 1, then the possibility of bit-pairs 01 or 10 is exactly $p \cdot (1-p)$. Therefore, the possibility of getting the value 1 after using the Neumann-filter is defined through

$$p_n = \frac{p \cdot (1-p)}{2p \cdot (1-p)} = 0.5$$

4 Exceptional cryptographic endangerments due to lacks of entropy

The creation of useable random numbers is very important when used in cryptographic concerns, since those values are commonly used to generate a shared secret. Therefore, if the random stream can be reproduced, even the strongest cryptosystem won't increase security.

4.1 Early Netscape SSL implementations

Netscape's implementation of the Secure Socket Layer (which offers strong cryptography) in the Navigator-software originally only took three parameters to gather entropy for key generation and exchange:

- system ID of the current process,
- system ID of the parent process, and
- the current time of the day.

The following code shows the main SSL routine of gaining entropy:

```
global variable seed;

RNG_CreateContext()
    (seconds, microseconds) = time of day; /* Time elapsed since 1970 */
    pid = process ID;  ppid = parent process ID;
    a = mklcpr(microseconds);
    b = mklcpr(pid + seconds + (ppid << 12));
     seed = MD5(a, b);
```

Several security related papers have shown that this combination of pseudo-random sources is breakable within under one minute ([Gut98]).

4.2 TCP/IP sequence number prediction

In a great paper from S. M. Bellovin [Bel89], a detailed description can be found about how to predict TCP packet sequence numbers of a host without receiving any respones during an attack on Berkeley systems. TCP uses stateful connection and hence sequence numbers to assure correctness of the sequence. Both, server and client, chose their own random values as initial sequence number θ, which is to be acknoledged by the other partner:

1. C $\rightarrow$ S: SYN (θ_C)
2. S $\rightarrow$ C: SYN (θ_S), ACK (θ_C)
3. C $\rightarrow$ S: ACK (θ_S)

When the client IP address has been spoofed, the answer of the server will not reach the sender of the packet, therefore the client basically won't get θ_S and cannot acknoledge this number.

Bellovin pointed out, that, on Berkeley systems, the sequence number is permanentely incremented once a second by a constant amount; furthermore it is increased by half of this amount everytime a connection is initiated, what levels the ground for serious attacks by SYN flooding.

4.3 Other Software vulnerabilities

Developers often rely on the randomisation algorithms provided by operating sytems and/or software libraries on the one hand, and produce buggy code when using these mechanisms on the other hand.

- Certain versions of the Layer 2 Tunneling Protocol Daemon (*L2TPD*, defined in RFC 2661) fail to initialize the random number generator with a seed-value before calling the `rand()` function, which is part of the ANSI C `stdlib` library. Behaviour-prediction and man-in-the-middle attacks have been noticed (Bugtraq ID 5451).
- The Red Hat Linux `mkpasswd` tool originally claimed to use an own advanced random number generator, but initiated the generator with only its own process ID; only a very small set of possible passwords have been generated (Bugtraq ID 2632).
- Recently discovered, the NetBSD operating system driver for specific Intel hardware random number generators did not detect the hardware chipset correctly. Predictable keys are possibly being created. (Bugtraq ID 17496).
- Early version of the official Cisco VPN Client had a weak implementation of a random number generator, which has also been used for creating TCP sequence numbers and have therefore be vulnerable to man-in-the-middle attacks and packet injection (Bugtraq ID 5653).
- The random number generator in Linux kernels 2.6 before 2.6.21.4 do not proberly seed pools when there is not enough entropy available. Moreover, an incorrect casting may force the random number generator to produce the same values everytime after reboots on system without an entropy source (CVE-2007-2453).

5 Conclusion

Random numbers are used in a broad range of applications, but only a few of them may lead to such catastrophic results as when used within cryptography. Low-entropy sources may probably result in predictiveness of the random bit stream, which turns out to be utterly devastating if cryptographic keys are generated from these data flows.

References

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. Department of Computer Science and Engineering, Institute of Technology Kanpur, India, 2004. (URL http://www.cse.iitk.ac.in/users/manindra/algebra/primality.pdf).

[BD07] Cals Bosley and Yevgeniy Dodis. Does privacy require true randomness? International association for cryptologic research, 2007.

[Bel89] S.M. Bellovin. Security problems in the tcp/ip protocol suite. AT&T Bell Laboratories, 1989.

[DIF94] D. Davis, R. Ihaka, and P. Fenstermacher. Cryptographic randomness from air turbulence in disk drives. Proceedings of the 14th international cryptology conference, 1994.

[DR01] John Daemen and Vincent Rijmen. *The design of Rijndael.* Springer, 2001. (ISBN 3540425802).

[Ert03] Wolfgang Ertel. *Angewandte Kryptographie.* Hanser Fachbuchverlag, 2003. (ISBN 3446223045).

[ES00] Wolfgang Ertel and Erhard Schreck. Real random numbers produced by a maxtor disk drive. University of applied sciences, Weingarten, Germany, 2000.

[GNP06] Peter Gutmann, David Naccache, and Carles Palmer. Randomness in cryptography. IEEE security and privacy, 2006.

[GR03] Anna Gal and Adi Rosen. Lower bounds on the amount of randomness in private computation. ACM 1-58113-674-9, 2003.

[Gut98] Peter Gutmann. Software generation of practically strong random numbers. Department of computer science, University of Auckland, 1998.

[HL93] Amir Herzberg and Michael Luby. Public randomness in cryptography. Springer-Verlag Berlin Heidelberg, 1993.

[JG02] Ari Juels and Jorge Guajardo. Rsa key generation with verifiable randomness. Department of electrical engineering and information sciences, Ruhr-Universität Bochum, Germany, 2002.

[Knu98] Donald E. Knuth. *Seminumerical Algorithms.* Addison-Wesley, 1998. (ISBN 0201896842).

[Luc05] Stefan Lucks. Kryptographie ss05. University of Mannheim, 2005.

[LV00] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. Citibank N.A., Techn. Universiteit Eindhoven, 2000.

[MLT07] John Mitchcock, Jack Lutz, and Sebastiaan Terwijn. The arithmetical complexity of dimension and randomness. ACM Transactions on computational logic, Vol. 8, No. 2, Article 13, 2007.

[Ran95] Ranno. Cryptographic random numbers. ?, 1995.

[Rie94] H. Riesel. *Prime numbers and computer methods for facorization.* Birkhaeuser, 1994.

[Riv94] Ron Rivest. Rc4 algorithm revealed, 1994. (URL: http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca9).

[Sch94] Bruce Schneier. *Fast software encryption, Cambrindge Security Workshop Proceedings (Dec. 1993).* Springer, 1994. pp. 191-204.

[Sch05] Bruce Schneier. *Angewandte Kryptographie. Algorithmen, Protokolle und Sourcecode in C.* Pearson Studium, 2005. (ISBN 0471117099).

[SKW$^+$98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher, 1998.

[SR07] M. Stipcevic and B. Medved Rogina. Quantum random number generator. Rudjer Boskovic Institute, Bijenicka, Zagreb, Croata, 2007.

[Wae03] Dietmar Waetjen. *Kryptographie. Grundlagen, Algorithmen, Protokolle.* Spektrum Adakemischer Verlag, 2003. (ISBN 3827414318).