Scientific and Practical Cyber Security Journal (SPCSJ) 4(4): 10-14 ISSN 2587-4667
Scientific Cyber Security Association (SCSA)

PSEUDO RANDOM NUMBER GENERATORS AND ITS USES

მაქსიმ იავიჩი - კავკასიის უნივერსიტეტი
Maksim Iavich – Caucasus University

გიორგი პაპავა სკოლა: 35-ე საჯარო სკოლა

Giorgi papava: 35 Public School
ნიკუშა ნადარაია სკოლა: სკოლა ლიცეუმ "მწიგნობართუხუცესი"
Nikusha Nadaraia. School "Mtsignobartukhutsesi"
გიორგი გოგუაძე სკოლა: 1 გიმნაზია
Giorgi Goguadze. School 1
ბექა პარასკებაშვილი სკოლა:154-ე საჯარო სკოლა
Beqa Paraskebashvili. 154 Public School

ABSTRACT: Growing interest and need in technology made it necessary to make complicated systems and entertainment sources right in your device. As suggested in the paper, many of the services use pseudorandom number generators as services. In this paper, it is described how PRNG started off and its use in the current modern world, as well as its vulnerabilities and Cryptographically secure PRNGs.

KEYWORDS: *pseudo random number generator, PRNG*

აბსტრაქტი: გაზრდილი ინტერესი ტექნოლოგიაში და კომპიუტერული მოწყობილობებისის პოპულარობა, იძულებულს გვხდის შევქმნათ დახვეწილი სისტემები და მრავალი სხვა რამ ჩვენი ყოველდღიური გამოყენების გაჯეტებზე. როგორც ამ კვლევაში არის აღწერილი, მრავალი სერვისი იყენებს PRNG-ს. კვლევაში არის აღწერილი ადრეული ნაბიჯები PRNG-ს და თუ რაში იყენებენ მას დღესდღეობით. ასევე ნახსენებია მისი სუსტი წერტილები და კრიპტოგრაფიულად უსაფრთხო PRNG-ები

PRNG

A PRNG (pseudorandom number generator) is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers. The PRNG-generated sequence is not truly random, because it is completely determined by an initial value, called the PRNG's seed (which may include truly random values). Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed in number generation and their reproducibility.

PRNGs are central in applications such as simulations, electronic games and cryptography. Generating random numbers is an essential task in cryptography. Random numbers are necessary not only for generating cryptographic keys, but are also needed in steps of cryptographic algorithms or protocols (e.g. initialization vectors for symmetric encryption, password generation, nonce generation, ...). In a PRNG

with input, one only assumes that users can store a secret internal state and have access to a (potentially biased) random source.

EARLY STAGES OF PRNG

An early computer-based PRNG, suggested by John von Neumann in 1946, is known as the middle-square method. The algorithm is as follows: take any number, square it, remove the middle digits of the resulting number as the "random number", then use that number as the seed for the next iteration. For example, squaring the number "1111" yields "1234321", which can be written as "01234321", an 8-digit number being the square of a 4-digit number. This gives "2343" as the "random" number. Repeating this procedure gives "4896" as the next result, and so on. Von Neumann used 10 digit numbers, but the process was the same.

A problem with the "middle square" method is that all sequences eventually repeat themselves, some very quickly, such as "0000". Von Neumann was aware of this, but he found the approach sufficient for his purposes and was worried that mathematical "fixes" would simply hide errors rather than remove them.

Von Neumann judged hardware random number generators unsuitable, for, if they did not record the output generated, they could not later be tested for errors. If they did record their output, they would exhaust the limited computer memories then available, and so the computer's ability to read and write numbers. If the numbers were written to cards, they would take very much longer to write and read. On the ENIAC computer he was using, the "middle square" method generated numbers at a rate some hundred times faster than reading numbers in from punched cards.

The middle-square method has since been supplanted by more elaborate generators.

A recent innovation is to combine the middle square with a Weyl sequence. This method produces high-quality output through a long period (see Middle Square Weyl Sequence PRNG).

Also In the second half of the 20th century, the standard class of algorithms used for PRNGs comprised linear congruential generators.Linear congruential generators (LCGs) are a class of pseudorandom number generator (PRNG) algorithms used for generating sequences of random-like numbers. The generation of random numbers plays a large role in many applications ranging from cryptography to Monte Carlo methods. The quality of LCGs was known to be inadequate, but better methods were unavailable.

PROBLEMS WITH PRNG

In practice, the output from many common PRNGs exhibit artifacts that cause them to fail statistical pattern-detection tests. These include:

- Shorter-than-expected periods for some seed states (such seed states may be called "weak" in this context);
- Lack of uniformity of distribution for large quantities of generated numbers;
- Correlation of successive values;
- Poor dimensional distribution of the output sequence;

- Distances between where certain values occur are distributed differently from those in a random sequence distribution.

Defects exhibited by flawed PRNGs range from unnoticeable (and unknown) to very obvious. An example was the RANDU random number algorithm used for decades on mainframe computers. It was seriously flawed, but its inadequacy went undetected for a very long time.

In many fields, research work prior to the 21st century that relied on random selection or on Monte Carlo simulations, or in other ways relied on PRNGs, were much less reliable than ideal as a result of using poor-quality PRNGs. Even today, caution is sometimes required, as illustrated by the following warning in the *International Encyclopedia of Statistical Science* (2010).

The list of widely used generators that should be discarded is much longer [than the list of good generators]. Do not trust blindly the software vendors. Check the default RNG of your favorite software and be ready to replace it if needed. This last recommendation has been made over and over again over the past 40 years. Perhaps amazingly, it remains as relevant today as it was 40 years ago.

consider the widely used programming language Java. As of 2017, Java still relies on a linear congruential generator (LCG) for its PRNG, which are of low quality.

One well-known PRNG to avoid major problems and still run fairly quickly was the Mersenne Twister, which was published in 1998. Other higher-quality PRNGs, both in terms of computational and statistical performance, were developed before and after this date; these can be identified in the List of pseudorandom number generators

An Ethereum lottery game, 1000 Guess, had a vulnerability that it generated predictable random numbers. This game decides a winner by a random number when the number of players who bet on the contract reaches to the predetermined number. The contract generated the random number using Sha256() function with private variables and the current block variables, such as block.timestamp, block.coinbase and block.difficulty. However, they are easily readable. First Private variable is accessible by using web3.eth.getStorageAT command. Second, it is well known that block variables can be easily manipulated by malicious miners. So hackers certainly capitalized on it and got a lot of profit from the service.

CRYPTOGRAPHICALLY SECURE PRNGS

A PRNG suitable for cryptographic applications is called a *cryptographically secure PRNG* (CSPRNG). A requirement for a CSPRNG is that an adversary not knowing the seed has only negligible advantage in distinguishing the generator's output sequence from a random sequence. In other words, while a PRNG is only required to pass certain statistical tests, a CSPRNG must pass all statistical tests that are restricted to polynomial time in the size of the seed. Though a proof of this property is beyond the current state of the art of computational complexity theory, strong evidence may be provided by reducing the CSPRNG to a problem that is assumed to be hard, such as integer factorization. In general, years of review may be required before an algorithm can be certified as a CSPRNG.

Some classes of CSPRNGs include the following:

- stream ciphers
- block ciphers running in counter or output feedback mode
- PRNGs that have been designed specifically to be cryptographically secure, such as Microsoft's Cryptographic Application Programming Interface function CryptGenRandom, the Yarrow algorithm (incorporated in Mac OS X and FreeBSD), and Fortuna
- combination PRNGs which attempt to combine several PRNG primitive algorithms with the goal of removing any detectable non-randomness
- special designs based on mathematical hardness assumptions: examples include the *Micali–Schnorr generator*, Naor-Reingold pseudorandom function and the Blum Blum Shub algorithm, which provide a strong security proof (such algorithms are rather slow compared to traditional constructions, and impractical for many applications)
- generic PRNGs: while it has been shown that a (cryptographically) secure PRNG can be constructed generically from any one-way function, this generic construction is extremely slow in practice, so is mainly of theoretical interest.

It has been shown to be likely that the NSA has inserted an asymmetric backdoor into the NIST certified pseudorandom number generator Dual_EC_DRBG

Most PRNG algorithms produce sequences that are uniformly distributed by any of several tests. It is an open question, and one central to the theory and practice of cryptography, whether there is any way to distinguish the output of a high-quality PRNG from a truly random sequence. In this setting, the distinguisher knows that either the known PRNG algorithm was used (but not the state with which it was initialized) or a truly random algorithm was used, and has to distinguish between the two.The security of most cryptographic algorithms and protocols using PRNGs is based on the assumption that it is infeasible to distinguish use of a suitable PRNG from use of a truly random sequence. The simplest examples of this dependency are stream ciphers, which (most often) work by exclusive-or-ing the plaintext of a message with the output of a PRNG, producing ciphertext. The design of cryptographically adequate PRNGs is extremely difficult because they must meet additional criteria. The size of its period is an important factor in the cryptographic suitability of a PRNG, but not the only one.

ACKNOWLEDGEMENT

The work was conducted as a part of SPG-19-133 financed by Shota Rustaveli National Science Foundation of Georgia.

CONCLUSION

According to our research, we confirm that PRNG is truly extremely useful and reliable to use. From computer science to Gambling, It is relevant in various industries, But you should always use the secure PRNG algorithms to ensure that your business doesn’t collapse because of the parasites that want to exploit you. As the technology advances PRNG will advance too, and we will get closer and closer to true randomness.

REFERENCES:

1. Y. Dodis, D.Pointcheval, S. Ruhault, D. Vergnaud , D. Wichs Security Analysis of Pseudo-Random Number Generators with Input: /dev/random is not Robust ? eprint.iacr.org 2013
2. J.Song, Attack on Pseudo-random number generator (PRNG) used in 1000 Guess, an Ethereum lottery game (CVE-2018–12454) medium.com 2018
3. R.Saucier Computer Generation of Statistical Distributions (1st ed.). Aberdeen, MD. Army Research Lab. . (2000).
4. G Iashvili, Y Polishchuk, D Prysiazhnyy, Improved Post-quantum Merkle Algorithm Based on Threads, Advances in Computer Science for Engineering and Education III 1247, 454