Principles of Construction and Usage of

Pseudorandom Generators

Alexander Vakhitov

June 13, 2005

Abstract

In this report we try to talk about the main concepts and tools needed in pseudorandom generators creation. The report starts with a brief description of already mentioned in previous talks cryptographic ideas useful in the understanding of pseudorandom generators. Then different ways of their usage are discussed. In the next part one number-theoretical generator is presented. The report is finished with short description of the general approach to pseudorandomness from informational theory point of view.

Contents

1 Introduction

In the recent papers of Jass we talked about different ways to hide information. Here another one is discussed. The approach of pseudorandom generator using is more theoretical, but it can present the way of general analysis of cryptographic schemes using probability theory.

In the Part 2 we present the concept of pseudorandom generator and talk about some achievements shown in another Jass papers connected to the topic discussed here.

Next, in the Part 3 we specify the cryptographical problem which can be solved by pseudorandom generators and show the possible way to do it.

The Part 4 is about construction of pseudorandom generators. Here we present the example of pseudorandom generator (Blum-Blum-Shub generator) and then talk in brief about probabilistic approach to build and analyze the properties pseudorandom generators from one-way functions.

2 Useful Concepts and Tools

In this part we will talk about the concept of pseudorandom generator and some theoretical results, which are needed to construct the generator.

2.1 Pseudorandom Generator Concept

Pseudorandom Generator, as it can be seen from it's name, has to generate sequences which seem to be random. Somebody who doesn't know the algorithm which is used by generator cannot ever say about the sequence if it is random or pseudorandom.

Pseudorandom generator has an input called *random seed*. To make the probability proofs work, this seed should be truly random. But the main idea is to use comparatively small random seeds in the input and to produce long pseudorandom sequences of bits (or numbers) in the output.

Loosely speaking, pseudorandom generator expands the randomness of the input seed. Using this, we can build different useful computational and modeling schemes, removing random steps from randomized algorithms, etc. Here only the topic of using pseudorandomness in cryptography is discussed. But you can be sure that there are many other ways to use it.

To continue speaking about pseudorandom generators, we need some other information. The following ideas are introduced in details in other Jass papers, and we will only refer to those papers.

First of all comes the definition of *one-way function*.

2.2 One-Way Functions

Definition 1 *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is called* one-way *if holds:*

1. *There exist a polynomial-time algorithm computing* $f(x)$ *from every* $x \in \{0,1\}^*$*, and*
2. *For every probabilistic polynomial-time algorithm* A*, every polynomial* p*, and all sufficiently large* n*,*

$$Pr[A(f(x), 1^n) \in f^{-1}(f(x))] < \frac{1}{p(n)}$$

where the probability is taken uniformly over all possible choices of $x \in \{0,1\}^n$ *and all the possible outcomes of the internal coin tosses in* A*.*

In brief, there are several facts concerning this functions:

- *One-way function* is used in the construction of pseudorandom generator.

- Informally, f is one-way if it is easy to compute but hard to invert.
- If $P = NP$, then there are no one-way functions
- It is not ever known if $P \neq NP$ implies there are one-way functions.

Here are some hard problems which can be used in one-way functions construction: [1]

- Discrete logarithm problem ($g^x \bmod n$) for a large prime n and some e
- RSA: the same, but here $n = pq$ for large primes p and q.
- Quadratic residues problem
- Factoring a product of two large primes
- Nonnumber theoretic functions,including coding theory problems

The functions based on these problems are considered to be hardly-computed, but there is no such proof for every of them. We only think about them like about something computationally unsolvable. There is some analysis about their simple solving with some parameter (or set of them) special for the problem is known (called *trapdoor*), but without this parameter the task *seemstobe* very difficult.

2.3 Hidden Bit

Another useful idea in our topic is hidden bit (or hard-core predicate). Let's start with the definition:

Definition 2 *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core *(hidden bit) of a function f if for every probabilistic polynomial-time algorithm A, every positive polynomial p, and all sufficiently large n,*

$$Pr[A(f(x)) = b(x)] < \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken uniformly over all possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses in A.

If we have a one-way function (in the hidden bit definition it is referred as f) then we can say that it hides it's preimage. More precisely, it hides some information from it's preimage (or some bits of information). Hidden bit for this function f is just one of these bits.

The concept of one-way function accepts that some bits from x can be clearly found from $f(x)$. But if there are no hidden bits in the preimage of a function f, then clearly f is not $one - way$!

3 Using Pseudorandom Generator in Cryptography

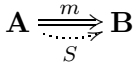
Here we talk about the problem of encryption and decription of information and it's solving with pseudorandom generator.

[1] These problems are discussed in this and other Jass papers. Look at *Information-Theoretic Cryptography; The RSA Cryptosystem and Factoring Integers; El-Gamal Cryptosystem and Probabilistic Encryption* papers.

3.1 Task Analysis

Let's briefly introduce the main problem. If you have already seen papers prepared by previous talks, then you know that ...

Observation 3 ... **A** *is sending some secret (for others) information to his (her) partner* **B**. *The problem is to hide the information in some way*[2].



Sixty-two years ago Shannon in his work about information theory (1943) [3] proved that *fully secure encryption system can exist if the size of the secret information* S *which* **A** *and* **B** *agree on prior is as large as the number of secret bits to be ever exchanged remotely using the encryption system.*[4]

You can think about this S as about a key which is xored with an information to make the encrypted message ($|S| = |I|$, $m = S \oplus I$ where I is the information to hide).

This fact seems to be dangerous for people who want to exchange secrets. But the concept of *fully secure encryption system* is very strict and gives us much more than we use. We only need our cryptographic algorithm to be unbreakable by people and their computers.

3.2 Pseudorandom Generator Using

Remember that:

Definition 4 Pseudorandom Generator *is a deterministic program used to generate a long sequence of bit which look like random sequences, given as input a short random sequence (the input seed).*

$$r \text{ truly random, } G \text{ is a pseudorandom generator, } \Rightarrow G(r) \text{ "looks like random" and } |G(r)| \gg |r|$$

Here is introduced the main concept of pseudorandom generator usage in cryptography. It is quite simple. At first, let's consider the situation when we have two different channels of information exchange according to Shannon's theory. First is secure channel, which is used by partners to agree on prior on some amount of information (this information is something like secret key). But it has only limited resource. Second is the unsecure channel, where partners want to transfer their encrypted information.[5]

The secure channel has limited resource. That's why we need cryptosystem to use non-secret channel to transfer encrypted information. Here comes an idea to use pseudorandom generator as a tool to produce long secret sequence from a short one. In detail, **A** and **B** can agree on prior to use some sort of pseudorandom generator and exchange with the input seed. Then, **A** produces a long pseudorandom sequence with given seed, XORes it with his (her) information and sends the result to **B**. **B** only needs to produce the same pseudorandom sequence and XOR it with the received message.

[2]Look to the first report about Classical Cryptography if you are interested in the history of this problem

[3]C.E. Shannon. Communication Theory of secrecy systems, 1949

[4]The second report in Jass by Herman was about Shannon's works and the theory of information

[5]Sometimes we don't need such a secure channel, for instance when we use RSA cryptosystem it is not needed. But here the case of this channel present is discussed.

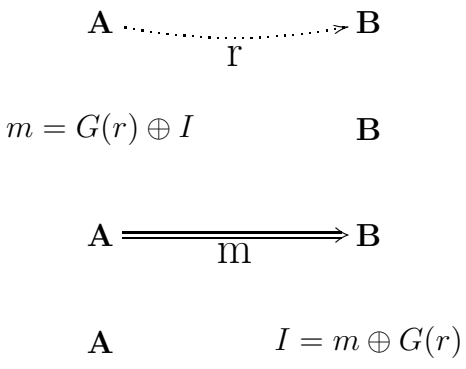


Figure 1: The usage of pseudorandom generator G with input seed r in encrypting the message m

4 Construction of a Pseudorandom Generator

4.1 Correct Definitions

The new definition of pseudorandom generator, more useful in proving and giving more precise information about it, will be discussed here.

We will use the concept of computational indistinguishability to formalize the pseudorandom generator concept. Computational indistinguishability is something from strict probability theory. We will say about two distributions that they are indistinguishable computationally, and it means that no algorithm can determine to which distribution belongs a sequence of values on it's input.

Definition 5 *We say that bit string sets $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm A, every polynomial p, and all sufficiently large n,*

$$|Pr[A(X_n) = 1] - Pr[A(Y_n) = 1]| < \frac{1}{p(n)}$$

where the probabilities are taken over the relevant distribution (X or Y) and over the internal coin tosses of algorithm A.

Pseudorandom generator can be presented as something that outputs a sequence computationally indistinguishable with uniformly distributed random sequence. Such definition is lower. The distribution there is defined over all the sequences which can be in the output of a generator with different random seeds on input.

Definition 6 *Let $l : \mathbb{N} \to \mathbb{N}$ satisfy $l(n) > n\,\forall n \in \mathbb{N}$. A* pseudorandom generator*, with stretch function l, is a (deterministic) polynomial-time algorithm G satisfying:*

1. *$\forall s \in \{0,1\}^*$, it holds that $|G(s)| = l(|s|)$*
2. *$\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{l(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable, where U_m denotes the uniform distribution over $\{0,1\}^m$.*

4.2 Pseudorandom Generator Based on a One-way Function

If we have an injective (one-to-one) one-way function $f : \{0,1\}^n \to \{0,1\}^{l_n}$ and $b : \{0,1\}^n \to \{0,1\}$ is a hidden bit of f then we can build a *pseudorandom generator* in a such way:

$$G(x) =< b(x), b(f(x)), b(f(f(x))), \ldots, b(f^{l(|x|)-1}(x)) >$$

It can be proved that the existence of one-way functions is equivalent to existence of pseudorandom generators. In the case shown upper we can prove security of this generator if our function f is strictly one-way (in the sense of Definition 1). Look that the stretch function can be as large as you want. Really, your insight can tell you that it is not really true.

If you have such a short input and very-very long output of a generator then it might be simple to predict next bits of generator output from previous ones. But the assumption that function is *one-way* gives us unlimited power to prove that it isn't so simple as you probably think.[6] Later it will be introduced a definition of one-way function *"with limited breakability"*.

4.3 Predictability of a Pseudorandom Generator

Speaking about predictability of a pseudorandom generator, it is clear that

Theorem 7 *Following conditions are equivalent:*

- *The distribution X, in our case it is $\{G(U_n)\}_n \in \mathbb{N}$, is computationally indistinguishable from a uniform distribution on $\{U_{l(n)}\}_{n\in\mathbb{N}}$*
- *The distribution X is unpredictable in polynomial-time; no feasible algorithm, given a prefix of sequence, can guess the next bit with a sufficient advantage over $\frac{1}{2}$*

In general, the sense of the theorem seems to be simple. If the sequence is random or nearly random, then you cannot predict it's next element, and vice versa - if the sequence is predictable then it is not random or nearly random.[7]

Easy to see that pseudorandomness implies polynomial-time unpredictability. Let's prove the inverse. At first, let's consider Hybrid Method of proofs.

Lemma 8 *Hybrid Method*

If some algorithm can distinguish distribution X from distribution Y, then it can distinguish a sample sequence of distribution X from a sequence of distribution Y.

Proof:

1. *Assume that we have multiple samples of distributions X and Y (that is, $\{\{X_n\}\}_m$ and $\{\{Y_n\}\}_m$ for $n, m \in \mathbb{N}$ (X_n is a single sample of a distribution with length n);*
2. *Consider sequence of samples $H_i = \{X_1, \ldots, X_i, Y_{i+1} \ldots Y_s\}$ for some $s \in \mathbb{N}$ - length of a* hybrid *H_i;*
3. *Distinguishing H_0 and H_s yields a procedure for distinguishing H_i from H_{i+1} for randomly chosen i (if D distinguishes X from Y, then it also distinguishes a pair of neighboring hybrids);*
4. *Then, we can build distinguisher D' for a single sample (S), which chooses i randomly, generates i samples $\{X_k\}$ from X and other samples $\{Y_k\}$ from Y, makes a sequence $\{X_1, \ldots, X_i, S, Y_1, \ldots\}$ and runs D on it.*

The Theorem Proof:

1. *$G(x)$ here is the generator.*
2. *Suppose that there exists algorithm $A : |Pr[A(x) = 1] - Pr[G(x) = 1]| > \epsilon, \epsilon > 0$.*
3. *Reverse $G'(s) = G(s)_{l(|s|),\ldots,1} =< b(f^{l(|x|)}(x), \ldots, b(x) >$.*

[6] Sometimes this power seems to be magic.

[7] Good, but here we will present you a method to prove something connected to computational indistinguishability and pseudorandomness. So, you should have patience and read the proof.

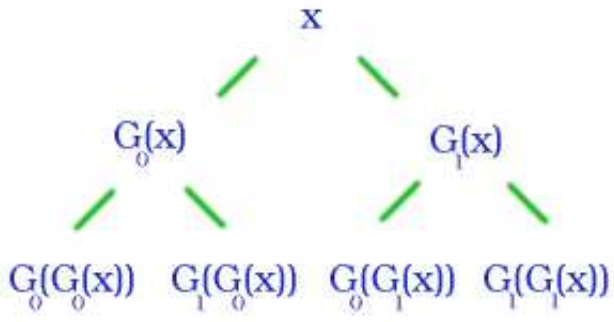
4. *Choose a random k. Then H_k is a* hybrid *built from G'(X) and $U_{l(n)}$: $H_k =< U_{l(n)}[1], \ldots, U_{l(n)}[k], G'(X)[1], \ldots, G'(X)[n] >$ ($G'(X) = H_n$ and $y = H_0$).*
5. *Given $b(f^{l-1}(x)), \ldots, b(f^{l-k}(x))$ A predicts $b(f^{l-k-1}(x))$*
6. *x is chosen from U_n. Then with given y=f(x) one can predict b(x) by invoking A on input $b(f^{k-1}(y)) \cdots b(y) = b(f^k(x)) \cdots b(f(x))$ which is polynomial-time computable from y.*

4.4 Pseudorandom Functions

There is some useful construction which you can build from pseudorandom generator. It is called *pseudorandom function* and is defined as:

Definition 9 *$f_s(x) : \{0,1\}^n \to \{0,1\}^n$ is* pseudorandom function *if it is infeasible to distinguish values of f_s for a random uniformly chosen s from values of truly random function $F : \{0,1\}^n \to \{0,1\}^n$*

We can build a pseudorandom function using a pseudorandom Generator G. Assume that PSRG G stretches in a factor of 2: $G(x) =< G_0(x), G_1(x) >$[8]; then let's build a binary tree:



Here you see that the way in the tree can be considered as a binary string. This string is a parameter s from the Definition 9, and our pseudorandom function is $f_s(x)$. Variable x here is the same x on the picture, and the image $f_s(x)$ is the leaf of the tree, concerning to s.

Pseudorandom functions can be used in different ways in hiding information. For instance, s is an event from the environment, which is not dependent on (it seems to be random for) the people participating in the hiding information scheme. Then one can encode his x with f_s and for everybody the result will seem to be random. But not for people participating: using s, they could reverse a function and find x.

4.5 Blum-Blum-Shub Generator

Let's look at $f_{BBS}(x) = x^2 \bmod n, n = pq$ for primes p and q congruent to 3 modulo 4.

Let's solve $a \equiv x^2 \bmod n$

$a \equiv x^2 \equiv (-x)^2 \bmod p$, and

$a \equiv (-y)^2 \equiv y^2 \bmod q$

Then there are four solutions for $a \equiv z^2 \bmod n$ ($z_{1,\ldots,4} = \pm cx \pm dy$), where

$$c \equiv \begin{cases} 1 \bmod p \\ 0 \bmod q \end{cases} \quad d \equiv \begin{cases} 1 \bmod q \\ 0 \bmod p \end{cases}$$

[8] if you have some pseudorandom generator, defined like in Definition 6, then you can build another generator with any stretch function you want. For the proof, look in my sources.

Squaring on $\mathbb{Z}_{n=pq}$ where $p \equiv q \equiv 3 \bmod 4$
$a^{p-1} \equiv 1 \rightarrow \sqrt{a} \equiv a^{\frac{p-1}{2}}$, if $p \equiv 3 \bmod 4 \rightarrow$
$a^{\frac{p-1}{2}} \equiv a^{2m+1}$ - unique square root in $Q_p = \{4m+3 \bmod p\} \subset \mathbb{Z}_p$; Squaring is a permutation on Q_p (every square has a unique square root, which is itself a square).

Then we can define one-way function for the generator: $f_{BBS}(x) = x^2 | x \in Q_p$.

The least significant bit of x is a hard bit for the one-way function f_{BBS}, because in squaring you loose the least significant bit and only if you kow the modulo you can reverse the function and find this bit.

This generator has a good additional property, that's why it is sometimes useful. The j-th bit of it's output can be computed directly, without computing previous j-1 bits:

$G_{BBS}(x)_{\{}j\} = lsb(x^{2^j} \bmod n) = lsb(x^{\alpha} \bmod \phi(n))$ where $\phi(n) = (p-1)(q-1)$
$G_{BBS}(x)_{\{j\}}$ is computed in time $O(\max\{|x|^3, |x|^2 \log j\})$

4.6 General Generator Analysis

In this part we will discuss the theoretical approach to pseudorandom generator construction. We will not give any proofs or serious analysis, but only a brief introduction to the topic of analysis. The way will be shown, and if you would like to start doing deeper in the theory, you should read "A Pseudorandom generator from any One-Way Function". by J. Hastad, R. Impagliazzo, L. Levin and M. Ruby.

The first concept in probability analysis of different schemes with non-limited length of input or output is polynomil parameter, which is used like a polynomial stretch function.

Definition 10 *Parameter k_n is called* polynomial *if there is a constant $c > 0$ such that $\forall n \in \mathbb{N}$*

$$\frac{1}{cn^c} \leq k_n \leq cn^c$$

k_n is called **P**-time polynomial parameter *if in addition there is a constant $c' > 0$ such that $\forall n$, k_n is computable in time at most $c'n^{c'}$*

This polynomial parameter helps to formalize cryptographical constructions as *function ensembles*:

Definition 11 *Let $f : \{0,1\}^{t_n} \rightarrow \{0,1\}^{l_n}$ denote a* function ensemble*, where t_n and l_n are integer-valued* **P***-time polynomial parameters and where f with respect to n is a function mapping $\{0,1\}^{t_n}$ to $\{0,1\}^{l_n}$.*

- *f is injective* $\Rightarrow$ one-to-one function ensemble
- *f is injective and $l_n = t_n$* $\Rightarrow$ permutation ensemble
- *$f : \{0,1\}^{t_n} \times \{0,1\}^{l_n} \rightarrow \{0,1\}^{m_n}$* $\Rightarrow$ ensemble with 2 inputs

At most every primitive (pseudorandom generator, one-way function, hidden bit) is a function ensemble.

For instance, next is the formalization of adversary as a function ensemble breaking the encryption algorithm with some probability depending on time.

Definition 12 *Adversary A is a function ensemble, it is breaking another function ensemble f. The* time-success ratio of A for f $\mathbf{R_{t_n}} = T_n / sp_n(A)$*, where t_n is the length of the private input to f, T_n is the worst-case running time of A. In this case, we say A is an* **R***-breaking adversary for f. We say f is* **R***-secure if there is no* **R***-breaking adversary for f.*

When analyzing probability distributions, it is common to use the entropy concept. The entropy is a value measuring uniformness of a distribution. When the entropy grows, the distribution becomes more and more close to uniform.

Definition 13 *Let D be a distribution on a set S. We define the information of x with respect to d to be $I_D(x) = -\log(D(x))$; Let X be a random value with distribution D ($X \in_D S$ The* Shannon Entropy *of D is $H(D) = E[I_D(X)]$*

Shannon entropy measures the "true entropy" of a distribution. It means that analyzing pseudorandomness we will use also entropy which can be computed from a distribution sample by some algorithm. It is called *computational entropy.*

Definition 14 *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a $\mathbf{P}$-time function ensemble and let s_n be a polynomial parameter. Then f has $\mathbf{R}$-secure* computational entropy *s_n if there is a $\mathbf{P}$-time function ensemble $f' : \{0,1\}^{m_n} \to \{0,1\}^{l_n}$ such that $f(U_{t_n})$ and $f'(U_{m_n})$ are $\mathbf{R}$-secure computationally indistinguishable and $\mathbf{H}(f'(U_{m_n})) \geq s_n$.*

The difference between Shannon entropy and computational entropy is the main tool in the basis of pseudorandom generator construction and analysis.

The generator can be constructed from any one-way function. This construction is only theoretical (it means that the pseudorandom generator constructed using this scheme is not practical). The main steps of the construction are interesting like an example of formalization of the pseudorandom generator concept for future analysis. You can look at them (try to understand the differences between generators used in this scheme):

- Any one-way function
- False-Entropy Generator

  Definition 15 *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a $\mathbf{P}$-time function ensemble and let s_n be a polynomial parameter. Then f is an $\mathbf{R}$-secure* false-entropy generator *with false entropy s_n if $f(U_{t_n})$ has $\mathbf{R}$-secure computational entropy $H(f(U_{t_n})) + s_n$.*

  False-entropy generator concept is that it's computational entropy g(X) is significantly greater than the Shannon entropy of g(X).

- Pseudoentropy generator

  Definition 16 *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a $\mathbf{P}$-time function ensemble and let s_n be a polynomial parameter. Then f is an $\mathbf{R}$-secure* pseudoentropy generator *with pseudoentropy s_n if $f(U_{t_n})$ has $\mathbf{R}$-secure computational entropy $t_n + s_n$.*

  Pseudoentropy generator concept is that it's computational entropy g(X) is significantly greater than the Shannon entropy of X.

- Pseudorandom generator

Here the paper is almost finished. We have started with general concept of a pseudorandom generator, continued with an example of a practicaly used (but theoretically built) generator. The last part was about an approach to pseudorandom generator analysis; it is not a complete story about generator creation, but only a scheme, which can use your mind to work on it.

References

[HILL99] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A Pseudorandom Generator from Any One-Way Function

[GB01] S. Goldwasser, M. Bellare Lecture Notes on Cryptography

[GOL04] O. Goldreich Foundations of Cryptography - A Primer