# Cryptology (course 1DT075) - Uppsala University
# Review paper: "How to construct random functions"

Deniz Kücükahmetler: deniz.kucukahmetler.4776@student.uu.se
Erik Krantz: erik.krantz.4152@student.uu.se

February – 2020 Spring

# Contents

# 1 Problem statement

True randomness can be both difficult to achieve, define and measure. The research paper "How to construct random functions" main focus is on what defines the randomness of a function. Additionally, how to efficiently generate said functions with the criteria that there exists a one-way function. The underlying issue is to create a pseudorandom function generator, i.e a deterministic algorithm that in polynomial-time transforms a string and a one-way function to polynomial-time computable functions (1).

# 2 Problem approach

To create a better view of the problem, they evaluate previous theories. One of these is the Kolmogorov complexity. It says that, for an individual string, the randomness is defined as the length of the shortest description of the string. Therefore, this approach is far from being pseudorandom (1).

Due to being quite remote from pseudorandomness, another approach using computational complexity was considered (3; 2). Assuming you have a program running in polynomial-time, a set of strings is defined as polynomial random, later referred to as poly-random, if the programs outputs do not differ between inputs that are either randomly chosen in the given set or a set containing all existing strings. The bit-size of the input will be returned as a poly-bit-size.

The research is therefore focused on expanding the poly-random approach. More so, on creating a way to measure the random properties of functions. This to further find a way of generating functions with poly-random properties.

# 3 Poly-random collection

Let D be a string set. When randomly chosen elements in D are given a computer program that works in polynomial time, it generates the output O1. Next, a randomly chosen string set is also given to the same computer program and output is generated as O2. If O1 and O2 are the exact same output, D can be defined as polynomial random (poly-random). Furthermore, a function is called poly-random if the difference between the output of the function (with the asked values) and random outcomes of a coin flip cannot be detectable. And poly-random collection is a multi-set of poly-random functions. (1)

In order to better understand the behavior of poly-random functions we must better define their required properties. Firstly, a unique k-bit index is attached to each function. So, it is easy to choose the function if k-bit index exists. Secondly, there must exist a polynomial time algorithm that can compute the function for each individual index. Finally, there cannot exist a probabilistic algorithm that can distinguish functions from the subset to the functions from the whole set. Thus, fulfilling the requirement of pseudorandomness(1).

In summary, with the assumption that a one-way function exist, a poly-random collection of functions can be created from a Cryptographically strong psuedorandom bit generator, later referred to as a CSB generator.

# 4 One-way functions

In short, a one-way function is easy to compute but difficult to invert. In terms of ciphers, you want to create a simple way of ciphering, but difficult way of deciphering. According to the definition which Levin put forward, a one-way function should satisfy three properties. First, it should be polynomial-time computable. Second, it should be hard to invert problematically. Third, $\bigcup_k D_k$ should be sampleble where $D_k$ is a subset of the set of all k-bit strings. Furthermore, the existence of a CSB generator is a requirement for a one-way function (4).

# 5 CSB-generators

Cryptographically strong pseudorandom bit (CSB) generators are deterministic programs, i.e. no randomness in further states of the program. They convert a random input into a longer pseudorandom output. It basically expands the random seed into a pseudorandom sequence. However, this sequence should satisfy some properties to be a truly random one. This pseudorandom sequence is called a CSB sequence since it is generated with a CSB generator. Yao suggested a general definition for statistical properties of CSB sequence.

**Theorem 1. (2)** Let $S = \bigcup_k S_k$ be a sampleble multiset of bit sequences. Then the following three statements are equivalent.

- $(i)$ S passes the next-bit-test.
- $(ii)$ S passes all polynomial-time statistical tests for strings.
- $(iii)$ S passes all polynomial-time statistical tests whose input consists of a single string in S.

CSB generators are efficient in terms of time since a pseudorandom bit can be generated with polynomial-time computations. So, coin tosses can be replaced by the sequence generated by a CSB generator. However, to generate a $k^t$ length string from a k length seed, in the worst-case CSB needs $k^t$ bits of storage. This is because CSB predicts every next bit statistically by taking into account the ordered series of previous bits. This is not the case for poly-random collections. Poly-random collections can generate k-bit long sequence with an oracle, and the required storage is only k-bits. In this approach, each bit is generated by using a selected function from the poly-random collection with respect to polynomial-time computation done by a random oracle. Therefore, CSB generators have an essential deficiency in terms of storage usage. In addition to the comparison of CSB and poly-random collections, the CSB sequence may not be truly "random" so it fails the statistical randomness-tests. This is explained in detail in the following section 5.4. It can be concluded that for a truly random sequence and without using more than k-bit storage, poly-random collections is more preferable compared to CSB generators.

## 5.1 Next-bit-test

Next-bit-test ensures that for known first i bits in CSB sequence, the bit in the position i+1 is statistically hard to predict. (2) (3)

## 5.2 Polynomial-time statistical tests for functions

It is known that all CSB sequences pass polynomial-time-statistical tests. (2)

## 5.3 How to implement CSB Generators?

In the early steps of the ideas for constructing CSB generators, they were constructed using a general complexity-theoretic assumption which defines necessary conditions to generate CSB generators. (5) Also, the idea of constructing CSB generators with a specific complexity assumption emerged. This was based on the predictability of the next bit in the sequence. But then, it is understood that there can be more efficient approaches built on factoring. Then, Levin showed that for generating CSB generators, the existence of a one-way function is enough. He even proved the theorem as "There exists a one-way function if and only if there exists a CSB generator."

**Conditions for construction of CSB generators**
With the implication that there exists a one-way permutation, conditions for constructing CSB generators have been shown by Blum and Micali (3) and is as follows:

1. The domain is accessible.
2. The permutations is easy to evaluate.
3. The predicate is inapproximable.
4. There exists a polynomial-time algorithm that computes the predicates of the permutation.

### 5.4 Easy Access Problem for CSB Generators

There could be a weak point in CSB sequence. There is a possibility that exponentially long bit-strings may include a pattern. It means that they are predictable and not random. Thus, the sequence loses its power with the emergence of "easy-access problem". Brassard and Blum first came up with this problem (7). Blum proved that, if the CSB sequence is created by using a one-way permutation generated with Blum-integers, it passes statistical tests (6). In conclusion, using CSB generators is not giving the desired result each time because exponential bits do not satisfy the "randomness preserving" property. However, the easy-access problem is solved when poly-random collections are used for random sequence generation. Additionally, poly-random sequences are able to create those truly random sequences without the necessity of using a Blum-integer in a one-way function. The only necessity is any kind of one-way function for poly-random sequences. To conclude, the easy-access problem is notable for CSB generators but can be solved for poly-random sequences.

## 6 Constructing poly-random collections

In the following subsections, how to construct poly-random collections is explained. The necessity for the functions in the collection is passing the statistical tests.

### 6.1 Statistical tests for functions

An oracle can be defined as a theoretical black box, such that it behaves differently for each state. To verify that the collection is indeed poly-random they need to perform tests using a polynomial-time algorithm with access to an oracle. Essentially, by looking at the relation between the input and output it is decided whether the function is random or not. The test is done in two phases.

- Check the output values based on input values decided by the Algorithm
- Then it checks if the function is in $F_k$ (given collection) or in $H_k$ (random collection)

However, it is worth noting the case where the given collection of functions passes the test, the origin of the collection of a function will be difficult to find. This due to the fact that the oracle will not provide sufficient information. Therefore, the outputs will be similar in terms of probability.

### 6.2 Construction of function collection

By using a CSB generator you can construct a poly-random collection. Explicitly, existence of pseudorandom strings provides pseudorandomness for functions. These pseudorandom functions form poly-random collection. It was already stated that CSB generators exist with one-way functions. Therefore, it can be concluded that poly-random collections can be constructed with the existence of poly-random collections (1).

### 6.3 Verification of statistical test for strings

The test is made in a two-step process using two algorithms. First, the probabilistic algorithm that utilizes the access of an oracle of a function, denoted as O. The second algorithm will be the test algorithm, denoted as T.

- The algorithm O selects an index in the range of $0 \leq k - 1$ using uniform probability, i.e. equal probability based on the size of k.
- Then algorithm O feeds the given k as an input to the test algorithm, T. By the use of the set of k-bit strings, O answers the oracle queries in T.

Essentially, the test algorithm writes queries on the oracle tape, and the probabilistic algorithm answers the oracle queries.

### 6.4 Generalized poly-random collections

To define a generalized view of poly-random collections can be very useful. For example, in hashing the essential part is to compress a string to a value of shorter length that still represent the original string. Furthermore, you want to convert over set of functions based on different polynomials. Moreover, you need to define a poly-random collection, which is created by using two different CSB generators.

- The first generator converting k-bit strings into 2k bit strings
- The second generator converting random input k-bits into psuedorandom bits.

With a given input and a random function chosen in the space of k-bit strings, which is also an element in the poly-random collection. It is then possible to verify that the collection satisfies the three properties of poly-random collection as explained in section [3].

## 7 Prediction problems and Poly-Random collections

If a function is easy to evaluate given a key, one might think its simple to give an approximate conclusion with access to an oracle for that function. However, this is not the case and is what is explained in the following section.

### 7.1 Polynomially inferring

Given a collection of functions $F$ and a probabilistic algorithm $A$ that can make oracle calls. With an input k and access to a oracle for a function in the collection we can conclude the following.

$A$ queries the oracle of functions and returns an element that is in the space of k-bit strings, but not in the set of elements that have been written to the oracle. Said element, further referred to as $x$, will be chosen for examination. $A$ now contains two values $f(x)$ and the random k-bit number, but in a random order. Additionally, we one can say that the examination is passed if $A$ can detect which of the values that resembles the function of $x$.

We can further conclude that a collection of functions is "polynomially inferred" if, for an infinite number of inputs, passes the exam.

### 7.2 Predictability

If there exists an algorithm A that infers a collection of functions F. Then the collection does not pass the statistical test for functions, $T_A$.

For every queue made by A, the test asks the oracle for the function and returns the answer to A. Furthermore, when A outputs a string as a chosen exam, the queries on the chosen exam randomly picks an input in the interval of indexes and randomly returns $f(x)$ and $y$. If $f(x)$ is correctly identified, $T_A$ returns 1, otherwise 0.

F does not pass the test for infinitely many k since the probability that $T_A$ returns 1 is greater than the requirement of "passing the exam". In the case where k is a finite set the probability for $T_A$ is $1/2$ for either case.

A strength of their construction of poly-random collections is that the functions cannot be polynomially inferred, regardless of the one-way function being polynomially inferrable.

## 8 Cryptographic Applications

Given a system that uses truly random functions. It is very easy to replace said functions with a poly-random collection since they should have the proper properties needed. Thus, it will most likely still adhere to the rules of the system. Furthermore, with an existing CSB generator that works in polynomial time, a poly-random collection can be generated from it.
Poly-random collections have an important role in cryptographic applications. Here is a list of some of them: (9)

- Authentication with time stamping
- Generating secret identification numbers without a need of storing them
- Strong hashing for cryptographic purpose
- Ideal private key cryptosystems (done by first generating poly-random permutations) (8)

One of the most featured property of poly-random collections is not only to generate truly random sequences, but to generate those sequences time efficiently (polynomial time) and also using memory efficiently. As mentioned by Levin and Goldreich, poly-random collections provides "memoryless" signature scheme. (10)

## 9 Conclusion

In conclusion, a multiset of polynomial-time generated random functions (poly-random collection) can be generated with the existence of one-way functions. For constructing poly-random collection, CBS generators are used. CBS generators are created using a one-way function. They are used to stretch a k-bit seed into a pseudorandom 2k-bit string. However, compared to the poly-random collections, they are less reliable for generating random bit strings. The reason is if some conditions are not proven, e.g. if a CBS generator is constructed using a poor quality one-way function, created 2k-bit string includes predictable bits. On the other hand, poly-random collections can be generated using any type of one-way function. The most important properties of the collection are, it is efficiently generated, only k-bit memory is used to generate the random k-bit string, and the string is truly random. Randomness is defined in terms of statistical tests in this report. The sequence generated with respect to the poly-random collection is unpredictable. It is impossible to know the next bit in the sequence with an algorithm that works in polynomial time. In addition, poly-random collections can be used in many applications in Cryptology.

# References


[1] "How to construct random functions" - Oded Goldreich, Shafi Goldwasser and Silvio Micali Massachusetts Institute of Technology, Cambridge, Massachusetts (August, 1986)

[2] "YAO, A. C. Theory and applications of trapdoor functions. In Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science. IEEE, New York, 1982, pp 80-91

[3] BLUM, M., AND MICALI, S. - "How to generate cryptographically strong sequences of pseudo-random bits."
SIAM J. Comput. 13 (Nov. 1984), 850-864.

[4] LEVIN, L. A. - "One-way function and pseudorandom generators." In Proceedings of the 17th ACM Symposium on Theory of Computing (Providence, R.I., May 6-8). ACM, New York, 1985, pp. 363-365.

[5] LONG, D. L., AND WIGDERSONA. - "How discreet is discrete log?" In preparation. A preliminary version appeared in Proceedings ofthe 15th ACM Symposium on Theory of Computing (Boston, Mass.,Apr. 25-27). ACM, New York, 1983,pp. 413-420.

[6] BLUM, L., BLUM, M., AND SHUB, M. "A simple unpredictable pseudo-random number generator." SIAM J. Comput. 15 (May 1986), 364-383.

[7] BRASSARD,G. "On computationally secure authentication tags requiring short secret shared keys." In Advances in Cryptology: Proceedings of Crypto-82, D. Chaum, R. L. Rivest and A. T. Sherman, Eds. Plenum Press, New York, 1983, pp. 79-86.

[8] LUBY, M., AND RACKOFF,C. "Pseudo random permutation generators and cryptographic composition." In Proceedings of the 18th ACM Symposium on Theory of Computing (Berkeley, Calif., May 28-30). ACM, New York, 1986,pp. 356-363.

[9] GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. "On the cryptographic applications of random functions." In Advances in Cryptology: Proceedings of Crypto-84. B. Blakely, Ed. Lecture Notes in Computer Science, vol. 196. Springer-Verlag, New York, 1985, pp. 276-288.

[10] GOLDWASSER,S., MICALI, S., AND RIVEST, R. L. "A "paradoxical" signature scheme. In Proceedings of the 25th IEEE Symposium on Foundations of Computer Science." IEEE, New York, 1984, pp. 441-448.