How to strengthen pseudo-random generators by using compression[⋆]

Aline Gouget[1,⋆⋆], Hervé Sibert[2]

France Telecom Research and Development,
42 rue des Coutures, BP6243, F-14066 Caen Cedex 4, France
{[1]aline.gouget,[2]herve.sibert}@francetelecom.com

Abstract. Sequence compression is one of the most promising tools for strengthening pseudo-random generators used in stream ciphers. Indeed, adding compression components can thwart algebraic attacks aimed at LFSR-based stream ciphers. Among such components are the Shrinking Generator and the Self-Shrinking Generator, as well as recent variations on Bit-Search-based decimation. We propose a general model for compression used to strengthen pseudo-random sequences. We show that there is a unique (up to length-preserving permutations) construction that reaches an optimal trade-off between output rate and security against several attacks,

1 Introduction

The huge amount of work impulsed by the ECRYPT call for stream ciphers [5] shows how much progress has been made in stream ciphers analysis in the recent years. While researchers in the area are still willing to design new proposals with innovative, yet not always secure, ideas. If cryptanalysis seems to put the fate of stream ciphers at stake, this is also the consequence of a lack of theoretical security results for stream ciphers and pseudo-random generators.

Compression of sequences can strengthen pseudo-random generators used in stream ciphers. In particular, adding compression components can thwart algebraic attacks aimed at LFSR-based stream ciphers [1, 4]. Such components include decimation components such as the Shrinking Generator [3] and the Self-Shrinking Generator [15]. Decimation has come back in focus recently with the Bit-Search Generator [9] and subsequent variations on it [10].

Compression mechanisms may suffer from timing attacks [12] since the speed of the output is variable in a manner that depends on the generator's state. Thus, LFSR-based ciphers involving a decimation mechanism may be easily breakable in case of leakage of the number of times LFSRs are clocked for each output. However, such side channel attacks are usually alleviated by buffering the output, as described for instance in [14]; these issues are not discussed in this paper.

[⋆] Work partially supported by the French Ministry of Research RNRT X-CRYPT Project and by the European Commission under contract IST-2002-507932 via the ECRYPT Network of Excellence.

[⋆⋆] Current e-mail address: aline.gouget@gemplus.com

Our main purpose is to propose a general model for compression used in the generation of pseudo-random sequences, in order to build compression components upon theoretical results. In Section 2, we detail related work on the subject including the Shrinking Generator and the Bit-Search generator variation used in the DECIM proposal to the ECRYPT stream cipher project. In Section 3, we construct our framework for compression components using prefix codes dedicated to pseudo-random generation. In Section 4, we focus on the case when the compression output is 0 or 1. We show that there is then a unique (up to length-preserving permutations) construction that reaches an optimal trade-off between output rate and security against several attacks, including entropy-based reconstruction, linear equations retrieval, and FBDD attacks. In Section 5, we apply our results to the Self-Shrinking Generator and Bit-Search based decimation.

2 Related work

Generation of pseudo-random sequences using *compression techniques* relies on the use of a *compression function.* A *compression function* is a function that compresses m-bit inputs (m is not necessarily a fixed value) to n-bit outputs, where $m \geq n$. The properties required for such functions depend on the application context. For instance, one-wayness is required for cryptographic hash functions, whereas compression functions for data compression must not be one-way. The properties of a compression function to be used to *shrink* pseudo-random sequences are yet to be defined.

Decimation components are a particular case of compression components. The Shrinking Generator (SG) [3] compresses two sources of pseudo-random bits to create a third source of potentially better quality than the original sources; the term quality stands for the difficulty of predicting the pseudo-random sequence. Similarly, the Self-Shrinking Generator (SSG) [15], the Bit-Search Generator (BSG) [9] and its variants such as the ABSG [10] all compress a single source of pseudo-random bits in order to produce a second source of potentially better quality. The ABSG is used in the DECIM stream cipher proposed to the ECRYPT stream cipher project. The general running of DECIM is to produce a pseudo-random bit sequence from an LFSR filtered by a Boolean function which is next compressed by the ABSG.

The *output rate* is usually considered to compare the efficiency of compression components. The BSG and the ABSG have the advantage over the SG and the SSG that they operate at a rate 1/3 instead of 1/4 (i.e. producing n bits in the output requires on average $3n$ bits of the input sequence instead of $4n$ bits).

Security criteria are crucial for cryptographic compression components. Since many stream ciphers are LFSR-based, most theoretical results on compression components concern the period or the linear complexity of the sequences obtained by applying these components on the output of a maximal length LFSR. First, algebraic results show that regular decimation is not suitable [16]. Then, several attacks on stream cipher based on a compression component are known. The first type of attack focuses on the properties of the compression function when assuming that the input sequence is uniformly chosen. For instance, FBDD-attacks, proposed by Krause [13], rely on properties of the compression function in the context of LFSR-based generators. The attacks given in [10, 11] use the most probable case (when it exists) in order to reconstruct the input sequence in the context of LFSR-based generators. A second type of attack exploits more information on LFSR-based generators. For instance, the attack on the SG given in [3] exploits the knowledge of the feedback polynomials, and the attack on the SSG given in [6, 7] applies only for particular feedback polynomials.

3 A compression model for pseudo-random generation

One usually expects data compression techniques to transform an input sequence into a very short output sequence while keeping the ability to recover the input from the output, which means no information on the input shall be lost.

In the context of pseudo-random generation, the purpose is different. We focus on the use of the compressed output as the keystream used to cipher a message in a stream cipher. The input sequence $\mathbf{s}$ is supposed to be the pseudo-random output of a public mechanism with secret parameters (e.g., the output of an LFSR initialized with a secret key and an initialization vector). This mechanism may have weaknesses with respect to attacks aiming at correlation or algebraic properties of its output. Our aim is to delete from $\mathbf{s}$ enough information to prevent such attacks that may apply to $\mathbf{s}$, by hiding algebraic properties of the input sequence. At the same time, our output should not be too short compared with its input, so that it can be used for the same applications as $\mathbf{s}$.

Thus, our aim is opposite to usual data compression: we expect the compression algorithm to process the input into an output sequence which delivers as little information on the input as possible, while remaining as long as possible.

In the sequel, we call *random input sequences* those sequences that follow the uniform distribution of binary words: each word w is a prefix of a random input sequence with probability $1/2^{|w|}$, and all words are assumed to be independent.

3.1 Prefix codes and binary trees

A *binary code* is a subset of words of $\{0,1\}^+$. The language $\mathcal{C}^*$ of a binary code $\mathcal{C}$ is the set of all binary words that are concatenation of words in $\mathcal{C}$. A code $\mathcal{C}$ is a *prefix code* if no codeword has a strict prefix in $\mathcal{C}$. Notice that, in this case, the words of $\mathcal{C}^*$ parse into codewords in a unique manner. A code is *maximal prefix* when no other prefix code properly contains it. A code $\mathcal{C}$ is *right complete* if every word w can be completed into a word $v = ww'$ that belongs to $\mathcal{C}^*$ or, equivalently, if every word w with no prefix in $\mathcal{C}$ has a multiple $v = ww'$ in $\mathcal{C}$.

Proposition 1. *A code is maximal prefix if, and only if, it is prefix and right complete.*

Proof. Suppose $\mathcal{C}$ is maximal prefix. Let w be a non-empty word which has no prefix in $\mathcal{C}$. As $\mathcal{C}$ is maximal prefix, $\mathcal{C} \cup \{w\}$ is not a prefix code, so w has a right multiple in $\mathcal{C}$. Hence, $\mathcal{C}$ is right complete.

Conversely, let $\mathcal{C}$ be prefix and right complete, and $\mathcal{C}'$ be a prefix code that contains $\mathcal{C}$. Let $w \in \mathcal{C}'$. As $\mathcal{C}$ is right complete, w has a right multiple w' in $\mathcal{C}^*$. Let then m be the smallest prefix of w' in $\mathcal{C}$. As $\mathcal{C}'$ is prefix, this implies $m = w$, so we have $w \in \mathcal{C}$, and consequently $\mathcal{C}' = \mathcal{C}$. Therefore, $\mathcal{C}$ is maximal prefix. $\square$

Throughout the paper, we will see that all suitable codes for our constructions are maximal prefix codes. There is a natural bijection between binary prefix codes and binary trees called *coding trees*, in which a node either is a leaf, or it has two children. This bijection links the words of the code and the leaves of the tree. Thus, we often use the equivalence between binary prefix codes and binary trees in the sequel. An example of a coding tree is given in Figure 1.

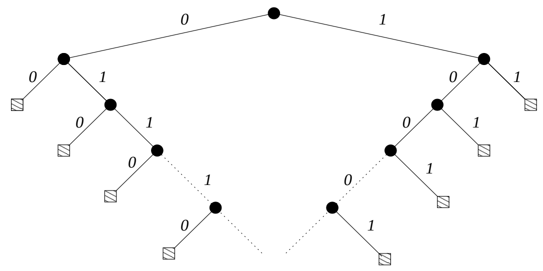

Fig. 1. ABSG code tree

3.2 General framework

We consider an infinite input sequence of bits $\mathbf{s} = (s_i)_{i\geq 0}$, a binary prefix code $\mathcal{C}$ and a mapping $f : \mathcal{C} \to \{0,1\}^*$, called the *compression function*. We call $f(\mathcal{C})$ the *output set*. The sequence $\mathbf{s}$ parses into a sequence of codewords $\mathbf{w} = (w_i)_{i\geq 0}$, each w_i being the unique codeword such that $w_0 \dots w_i$ is a prefix of $\mathbf{s}$ that belongs to $\mathcal{C}^*$. Each w_i is then mapped by f to its image in $f(\mathcal{C})$. The output sequence is $(f(w_i))_{i\geq 0}$, seen as a bit sequence. We denote this output sequence by

$$\mathbf{y} = Enc_{\mathcal{C},f}(\mathbf{s}).$$

The framework extends to finite input sequences, by parsing the input sequence in the same way, until the remainder has no prefix in $\mathcal{C}$.

Definition 1. *The* output rate *of the pair $(\mathcal{C}, f)$, denoted by $Rate(\mathcal{C}, f)$, is the average number of output bits generated by one bit of a random input sequence.*

Obviously, not all binary codes and functions are suitable for this framework. For instance, choosing $\mathcal{C} = \{00\}$ does not enable to process a sequence containing ones. As for the function, choosing the projection onto the empty word ε produces an empty output sequence. In order to apply the framework to every possible input sequence, it is then necessary to determine what the requirements on the following components are:

1. the choice of $\mathcal{C}$ must enable the parsing of every random input sequence,
2. the choice of f must be such that, for uniformly distributed input sequences, the corresponding output sequences also follow the uniform distribution.

3.3 Requirements on $\mathcal{C}$

First, there are some straight requirements on $\mathcal{C}$. In our framework, we consider prefix codes only. Indeed, if $\mathcal{C}$ contained two distinct words w and w' with w a prefix of w', then w' would never appear in the decomposition $\mathbf{w}$ of $\mathbf{s}$. Therefore, we may delete from $\mathcal{C}$ all the codewords that already have a prefix in $\mathcal{C}$ with no loss of generality, thus transforming $\mathcal{C}$ into a binary prefix code. Next, we want every random input sequence to be processable. This implies that $\mathcal{C}$ is right complete. Overall, in order to effectively process any random input, we introduce the following definition:

Definition 2. *A binary code $\mathcal{C}$ is* suitable *if it is prefix and if the expected length $E(\mathcal{C})$ of an element of $\mathcal{C}$ in the decomposition of a random input sequence is finite.*

Proposition 2. *For a suitable code $\mathcal{C}$, the following equality holds:*

$$\sum_{w\in\mathcal{C}} \frac{1}{2^{|w|}} = 1.$$

Proof. Let us consider the binary tree corresponding to $\mathcal{C}$. We denote by L_n and N_n respectively the number of leaves and nodes of depth n. Then, we have $L_0 = 0$, $N_0 = 1$, and for every $n \geq 1$, the relation $L_n + N_n = 2N_{n-1}$ holds. Let $S_n = \sum_{0\leq k\leq n} \frac{L_n}{2^n}$. Then, we have $\frac{L_n}{2^n} = \frac{N_{n-1}}{2^{n-1}} - \frac{N_n}{2^n}$, which gives $S_n = N_0 - \frac{N_n}{2^n} = 1 - \frac{N_n}{2^n}$, so we only have to prove $N_n = o(2^n)$.

Now, N_n is the number of nodes of depth n, and a random input sequence begins with n bits corresponding to such a node with probability $\frac{N_n}{2^n}$. For each one of these nodes, the first word of the input sequence recognized as a word of $\mathcal{C}$ has length at least n. Thus, these nodes contribute at least $n\frac{N_n}{2^n}$ to $E(\mathcal{C})$. As $E(\mathcal{C})$ is finite, this implies that $\frac{N_n}{2^n}$ tends to 0 when n tends to ∞. $\square$

Therefore, in the case of a suitable code, $E(\mathcal{C})$ is equal to the mean length of the words of $\mathcal{C}$ for the uniform distribution on the alphabet $\{0, 1\}$, so we have:

Proposition 3. *Let $\mathcal{C}$ be a suitable code. Then, we have the equality*

$$E(\mathcal{C}) = \sum_{w\in\mathcal{C}} \frac{|w|}{2^{|w|}}.$$

Remark 1. If a prefix code $\mathcal{C}$ satisfies the equality in Proposition 2 (for binary codes, otherwise 2 is replaced by the size of the alphabet), then it is maximal prefix, and the equivalence holds when $\mathcal{C}$ is finite (see for instance [2]). Thus, suitable codes are maximal prefix, and the converse is true for finite codes, with $E(\mathcal{C})$ then being given in Proposition 3. However, being maximal prefix may not be sufficient for $E(\mathcal{C})$ to converge when $\mathcal{C}$ is infinite, as Example 1 will show.

Example 1. Let us consider the code $\mathcal{C}$ defined iteratively as follows: for every n starting from $n = 0$, $\mathcal{C}$ contains all the words $w1^{2^n}$, where w is a word of length 2^n with no prefix already in $\mathcal{C}$. The defined code is prefix, and every word w with no prefix in $\mathcal{C}$ with $2^{n-1} < |w| \leq 2^n$ can be completed into a word of $\mathcal{C}$ by concatenating enough 1's to reach length 2^{n+1}, so $\mathcal{C}$ is right complete. Therefore, $\mathcal{C}$ is maximal prefix. However, the number of words of length 2^{n+1} in $\mathcal{C}$ being at most 2^n, we get

$$\sum_{w \in \mathcal{C}} \frac{1}{2^{|w|}} \leq \sum_{n \geq 0} \frac{1}{2^{2^n}},$$

this last sum being strictly less than 1. Thus, a random binary sequence may never fall into $\mathcal{C}$ with non-zero probability. Hence, $E(\mathcal{C})$ is infinite, and it is no longer equal to the mean length of code words, which, here, is finite. The code $\mathcal{C}$ is an example of maximal prefix code which is not suitable.

3.4 Requirements on *f*

As for $\mathcal{C}$, there are also several immediate requirements on $f(\mathcal{C})$. However, they are more practical than theoretical: at first glance, $f(\mathcal{C})$ may be any set of binary words, including ε. Now, it must obviously contain at least two non-empty words, one beginning with 0, and the other with 1, in order to make it possible for the output to look random for random inputs. Moreover, it must be possible to construct every binary sequence with the elements of $f(\mathcal{C})$.

In order to be able to process every random input sequence, we introduce the following definition, which corresponds to the requirement of Definition 2:

Definition 3. *Suppose $\mathcal{C}$ is a suitable code. Let f be a compression function $f : \mathcal{C} \to \{0,1\}^*$. We say that the pair $(\mathcal{C}, f)$ is a* proper encoder *if the expected length $E(f(\mathcal{C}))$ of the image by f of an element of $\mathcal{C}$ in the decomposition of a randomly chosen input sequence is finite and nonzero.*

As we review the properties of the output sequences with respect to uniformly distributed input sequences, we have:

Proposition 4. *For a proper encoder $(\mathcal{C}, f)$, the expected length of the image by f of an element of $\mathcal{C}$ in the decomposition of a randomly chosen input sequence, denoted by $E(f(\mathcal{C}))$, is given by*

$$E(f(\mathcal{C})) = \sum_{w \in \mathcal{C}} \frac{|f(w)|}{2^{|w|}}.$$

Definitions 2 and 3 ensure the finiteness of $E(\mathcal{C})$ and $E(f(\mathcal{C}))$, so we get:

Proposition 5. *The* output rate *of a proper encoder $(\mathcal{C}, f)$ is given by*

$$Rate(\mathcal{C}, f) = \frac{E(f(\mathcal{C}))}{E(\mathcal{C})}.$$

Now, we are going to determine an optimal choice for the output set $f(\mathcal{C})$ against reconstruction of the input. To every word in the output corresponds a set of preimages in $\mathcal{C}$. Knowing an output word thus reduces the possible choices of preimages to one particular set. We will show that, in order to minimize the information rate, the set $f(\mathcal{C})$ should be as small as possible.

In order to ensure that the distribution of the output sequences satisfies randomness properties such as those described in [8], each bit of the output sequence must have equal probability to be 0 or 1. Therefore, we need, for every $n \geq 1$:

$$\sum_{w\in\mathcal{C},|f(w)|\geq n, f(w)_n=0} \frac{1}{2^{|w|}} = \sum_{w\in\mathcal{C},|f(w)|\geq n, f(w)_n=1} \frac{1}{2^{|w|}},$$

where $f(w)_n$ is the n-th bit of the word $f(w)$.

The prefix code output case. First, we consider the case where $f(\mathcal{C})$ is a prefix code. If it contains two elements, the only possible choice such that the probability distribution of the output for random inputs is that of a random sequence is $f(\mathcal{C}) = \{0, 1\}$. In this case, 0 and 1 must have probability $\frac{1}{2}$ to appear in the output sequence for a random input sequence.

Suppose now that $f(\mathcal{C})$ has more than 2 elements. We want to prove that, given a random input sequence, knowing the output sequence, we can retrieve more information on the first element of $\mathcal{C}$ than in the case $f(\mathcal{C}) = \{0, 1\}$.

Proposition 6. *Let $(\mathcal{C}, f)$ be a proper encoder, and, for $x \in f(\mathcal{C})$, let $P(x) = \sum_{w\in f^{-1}(x)} \frac{1}{2^{|w|}}$. Then, for a random input sequence $\mathbf{s}$, each word of the decomposition of $\mathbf{s}$ over $\mathcal{C}$ has average length $E(\mathcal{C})$, and it is known with an average entropy*

$$E(\mathcal{C}) + \sum_{x\in f(\mathcal{C})} P(x)\log P(x).$$

Proof. For $x \in f(\mathcal{C})$, let us denote by $\mathcal{C}_x$ the preimage of x in $\mathcal{C}$. Then, the probability that the first element of $\mathcal{C}$ recognized in a random input sequence is mapped by f to x is $P(x) = \sum_{w\in\mathcal{C}_x} \frac{1}{2^{|w|}}$. Similarly, the expected length of an element in the preimage of x is $E(\mathcal{C}_x) = \frac{1}{P(x)} \sum_{w\in\mathcal{C}_x} \frac{|w|}{2^{|w|}}$. At last, we compute the entropy on the elements in $\mathcal{C}_x$:

$$\begin{aligned} H(\mathcal{C}_x) &= -\sum_{w\in\mathcal{C}_x} \frac{1}{P(x)2^{|w|}} \log \frac{1}{P(x)2^{|w|}} = \sum_{w\in\mathcal{C}_x} \frac{1}{P(x)2^{|w|}} (\log P(x) + |w|) \\ &= \frac{1}{P(x)} \sum_{w\in\mathcal{C}_x} \frac{|w|}{2^{|w|}} + \frac{\log P(x)}{P(x)} \sum_{w\in\mathcal{C}_x} \frac{1}{2^{|w|}} = E(\mathcal{C}_x) + \log P(x). \end{aligned}$$

The average number of bits retrieved is therefore $\sum_{x\in f(\mathcal{C})} P(x)E(\mathcal{C}_x) = E(\mathcal{C})$ for a random input sequence, so it does not depend on $f(\mathcal{C})$. The average entropy is

$$\sum_{x\in f(\mathcal{C})} P(x)(E(\mathcal{C}_x) + \log P(x)) = E(\mathcal{C}) + \sum_{x\in f(\mathcal{C})} P(x)\log P(x),$$

with $\sum_{x\in f(\mathcal{C})} P(x) = 1$. □

It is always possible, given a suitable code $\mathcal{C}$, to divide $\mathcal{C}$ into two equiprobable subsets (the probabilities of leaves in the tree being of the form $\frac{1}{2^n}$ with $n \geq 1$, and their sum being 1). Thus, for every suitable code, there exists a mapping $f : \mathcal{C} \to \{0, 1\}$ such that 0 and 1 are output with probability $\frac{1}{2}$.

Therefore, in order to maximize the entropy for a given suitable code $\mathcal{C}$, the value of $|\sum_{x\in f(\mathcal{C})} P(x) \log P(x)|$ should be as small as possible, which implies $\#(f(\mathcal{C})) = 2$. Therefore, the optimal choice of the output set is $f(\mathcal{C}) = \{0, 1\}$, with 0 and 1 having probability $\frac{1}{2}$ to be output for a random input sequence.

The non-prefix output case. We now consider the case where $f(\mathcal{C})$ does not contain the empty word ε, but $f(\mathcal{C})$ is not a prefix code. Let $\mathcal{C}(\mathbf{y})$ be the set of words of $\mathcal{C}$ such that, for every $w \in \mathcal{C}(\mathbf{y})$, the sequence $\mathbf{y}$ begins with w. Then, the probability that $\mathbf{s}$ begins with w depends on $\mathbf{y}$.

Example 2. Suppose $f(\mathcal{C}) = \{0, 01, 10, 11\}$ with $P_0 = P_{11} = \frac{1}{3}$ and $P_{01} = P_{10} = \frac{1}{6}$. Then, the first word of the finite output sequence 010 corresponds to a pair of words (w, w') of $\mathcal{C}$, with either $f(w) = 0$ and $f(w') = 10$, or $f(w) = 01$ and $f(w') = 0$. As we have $P_0P_{10} = P_{01}P_0$, the probabilities that $f(w) = 0$ and $f(w) = 01$ are equal whereas $P_0 > P_{01}$.

Thus, it is no longer possible to determine with certainty each word $(f(w_i))$ in the image of the input sequence. However, a path similar to that of Section 3.4 can be followed. The corresponding Proposition 12 and its proof are provided in Appendix A. They lead to the same conclusion as Section 3.4, namely that the optimal choice of the output set is $f(\mathcal{C}) = \{0, 1\}$ (thus being prefix), with 0 and 1 having probability $\frac{1}{2}$ to appear in the output sequence for a random input sequence. This case is discussed in Section 4.

General case. We now suppose that ε can belong to the output set $f(\mathcal{C})$.

Proposition 7. *Let $(\mathcal{C}, f)$ be a proper encoder such that $f(\mathcal{C})$ contains ε. Then, there exists a proper encoder $(\mathcal{C}', f')$ such that $f'(\mathcal{C}')$ does not contain ε and that, for every infinite binary sequence $\mathbf{s}$, we have*

$$Enc_{\mathcal{C},f}(\mathbf{s}) = Enc_{\mathcal{C}',f'}(\mathbf{s}).$$

Moreover, defining $P_\varepsilon = \sum_{w\in f^{-1}(\varepsilon)} \frac{1}{2^{|w|}}$, we have

$$E(\mathcal{C}') = \frac{1}{1-P_\varepsilon} E(\mathcal{C}), \text{ and } E(f'(\mathcal{C}')) = \frac{1}{1-P_\varepsilon} E(f(\mathcal{C})).$$

Proof. Denote by $\mathcal{C}_\varepsilon$ the set of preimages of ε, and by $\mathcal{C}_{\bar{\varepsilon}}$ the complement of $\mathcal{C}_\varepsilon$ in $\mathcal{C}$. Let $\mathcal{C}'$ be the binary code defined by $\mathcal{C}' = \mathcal{C}_\varepsilon^* \mathcal{C}_{\bar{\varepsilon}}$, that is, the set of binary words that parse into a sequence of words of $\mathcal{C}_\varepsilon$, followed by a word of $\mathcal{C}_{\bar{\varepsilon}}$. Consider the function f' that maps each element ww' of $\mathcal{C}'$, with $w \in \mathcal{C}_\varepsilon^*$, and $w' \in \mathcal{C}_{\bar{\varepsilon}}$, to $f(w')$. As the decomposition is unique, f' is well-defined. Moreover, for every input sequence $\mathbf{s}$, the equality $Enc_{\mathcal{C},f}(\mathbf{s}) = Enc_{\mathcal{C}',f'}(\mathbf{s})$ is obviously satisfied. At last, we have $f(\mathcal{C}') = f(\mathcal{C})\backslash\{\varepsilon\}$, so the image of f' does not contain ε.

There remains to show that the new pair $(\mathcal{C}', f')$ is also a proper encoder. First, $\mathcal{C}'$ is also a prefix code because of unicity of the decomposition over $\mathcal{C}$. Next, as the length of ε is 0, we have

$$E(f'(\mathcal{C}')) = \sum_{v\in\mathcal{C}_\varepsilon^*, w\in\mathcal{C}_{\bar{\varepsilon}}} \frac{|f(w)|}{2^{|v|+|w|}} = \sum_{n\geq 0}\Big(\sum_{v\in\mathcal{C}_\varepsilon}\frac{1}{2^{|v|}}\Big)^n \times \sum_{w\in\mathcal{C}_{\bar{\varepsilon}}}\frac{|f(w)|}{2^{|w|}} = \frac{E(f(\mathcal{C}))}{1-P_\varepsilon}.$$

As the two encoders $(\mathcal{C}, f)$ and $(\mathcal{C}', f')$ are equivalent, they have the same output rate, which yields the same relation between $E(\mathcal{C}')$ and $E(\mathcal{C})$. Hence, $(\mathcal{C}', f')$ is a proper encoder. $\square$

Proposition 7 shows that we can suppose without loss of generality that $f(\mathcal{C})$ does not contain ε. Therefore, the optimal choice for $f(\mathcal{C})$ is

$$f(\mathcal{C}) = \{0, 1\}.$$

4 The $\{0, 1\}$-case

In this section, we focus on the optimal choice of the proper encoder $(\mathcal{C}, f)$ when $f(\mathcal{C}) = \{0, 1\}$, with 0 and 1 equiprobable relatively to the uniform distribution over the input sequence. We first give the results that arise from Section 3 in this case, and we study the security of the framework against well-known attacks: exhaustive reconstruction, most probable case reconstruction, equations retrieval and FBDD attacks. Then, using these security results, we deduce the optimal choice for $(\mathcal{C}, f)$ against these attacks.

4.1 Parameters of the $\{0, 1\}$-case

Firstly, we give some general properties of the framework in the $\{0, 1\}$ case. We denote by $\mathcal{C}_0$ and $\mathcal{C}_1$ the two sets of preimages of respectively 0 and 1 by f. We also define $\mathcal{C}_b^n = \{w \in \mathcal{C}_b, |w| = n\}$ and $D_b^n = \#(\mathcal{C}_b^n)$.

Proposition 8. *Let $(\mathcal{C}, f)$ be a proper encoder with $f(\mathcal{C}) = \{0, 1\}$. Then, for a random input sequence* $\mathbf{s}$*, the average length and entropy of each word of the decomposition of* $\mathbf{s}$ *over $\mathcal{C}$ are respectively $E(\mathcal{C})$ and $E(\mathcal{C}) - 1$.*

This result comes from Proposition 6 when applied to the case $f(\mathcal{C}) = \{0, 1\}$, with 0 and 1 being equiprobable. This equiprobability also implies:

Proposition 9. *Given a bit b of the output sequence, a word $w \in \mathcal{C}_b$ is the preimage of b with probability $\frac{1}{2^{|w|-1}}$.*

Proof. Each word w of $\mathcal{C}$ appears in the input sequence with probability $\frac{1}{2^{|w|}}$, and the probability that w belongs to $\mathcal{C}_b$ is $\frac{1}{2}$, which gives the result. $\square$

4.2 Security analysis

This section is dedicated to the general analysis of the security provided by the compression component. We also focus on the case when the input sequence is the output of a maximal length LFSR.

Exhaustive reconstruction. Exhaustive reconstruction consists in reconstructing consecutive bits of the input sequence from the output sequence starting from a fixed point in the output sequence. When a bit b appears in the output, the expected length and the entropy on the preimage of b in the input sequence are respectively equal to

$$E_b = \sum_{w \in \mathcal{C}_b} \frac{|w|}{2^{|w|-1}} \text{ and } H_b = -\sum_{w \in \mathcal{C}_b} \frac{1}{2^{|w|-1}} \log_2\left(\frac{1}{2^{|w|-1}}\right).$$

Developing H_b gives

$$H_b = \sum_{w \in \mathcal{C}_b} \frac{|w|-1}{2^{|w|-1}} = E_b - 1.$$

Therefore, for a bit b in the output, one can deduce E_b bits in the input, with entropy $E_b - 1$.

Suppose that the input sequence is given by a LFSR of length L with a public feedback polynomial and with the secret key as its initial state. Let $E = \frac{E_0+E_1}{2}$. It is therefore possible to retrieve the complete state of the LFSR with an attack of average complexity $\mathcal{O}(2^{\frac{E-1}{E}L})$, requiring $\mathcal{O}(\frac{L}{E})$ consecutive output bits.

Moreover, when $E_0 \neq E_1$ holds, the complexity of the attack can be reduced by seeking for a sequence where mostly bits b appear, with b such that $E_b < E_{\bar{b}}$. This yields an attack with better complexity, but requiring the knowledge of more output bits. The general running of this attack consists in taking a window of consecutive bits in the keystream sequence where most bits are b. The difficulty when mounting this attack is to determine the better trade-off between the length of the window and the required number of bits b in this window in order to retrieve L equations involving consecutive bits of the input sequence. Such an attack is described in [10] in the case of the BSG decimation algorithm.

Reconstruction based on the most probable case. Another reconstruction attack consists in betting each time that the preimage of a bit b is (one of) the most probable. Consequently, for each bit b, we set $\ell_b = \min\{|w|, w \in \mathcal{C}_b\}$, and $\mathcal{C}_b^{\text{short}} = \{w \in \mathcal{C}_b, |w| = \ell_b\}$. Contrary to the previous attack, we cannot choose the point from which consecutive input bits will be effectively reconstructed.

For a bit b in the output, the preimage of b is $w \in \mathcal{C}_b^{\text{short}}$ with probability $1/2^{\ell_b - 1}$. Thus, we recover ℓ_b bits of the input with probability $1/2^{\ell_b-1}$.

Suppose now that the input sequence is given by a LFSR of length L. Let $\ell = \frac{\ell_0+\ell_1}{2}$. It is then possible to retrieve the complete state of the LFSR with an attack of average complexity $\mathcal{O}(2^{\frac{\ell-1}{\ell}L})$, requiring $\mathcal{O}(2^{\frac{\ell-1}{\ell}L})$ output bits (namely, enough for the bet to succeed). In the case where not all the preimages of b have the same length, we have $\ell_b < E_b$, so the complexity of this attack is less than that of exhaustive reconstruction.

Like in exhaustive reconstruction, when $\ell_0 \neq \ell_1$ holds, the attack complexity can be reduced by seeking sequences where most bits are b, such that $\ell_b < \ell_{\bar{b}}$.

Equations retrieval. In some cases, and in particular when the input sequence is given by a maximum-length LFSR, it is sufficient to retrieve linear equations on bits that are not consecutive in the input sequence.

However, it is not necessarily easier to retrieve bits that are apart in the input sequence, because the compression process creates entropy on the length of the preimages of words in the output sequence. Thus, retrieving bits that are apart means that we are able to control the length of the gaps between the bits retrieved in the input sequence.

For a bit b in the output, the preimage of b has length n with probability $\frac{D_b^n}{2^{n-1}}$, where D_b^n is the number of preimages of b of length n. Now, if the preimage of b has length n, then we can derive a number ϕ_b^n of linear equations on the input bits satisfying

$$\max(0, n - (D_b^n - 1)) \leq \phi_b^n \leq n - \lceil \log(D_b^n) \rceil.$$

Therefore, we can retrieve at least $n + 1 - D_b^n$ equations with probability $\frac{D_b^n}{2^{n-1}}$. For a bit b in the output, the average number of retrieved linear equations is thus

$$\bar{\phi}_b = \sum_{n \geq 1} \frac{D_b^n \phi_b^n}{2^{n-1}},$$

the entropy on the length of the preimage of b being

$$H_b^{\text{length}} = -\sum_{n \geq 1} \frac{D_b^n}{2^{n-1}} \log(\frac{D_b^n}{2^{n-1}}).$$

In the best case (which can always be achieved by properly choosing $\mathcal{C}$ and f), where ϕ_b^n is the least possible, we obtain:

Proposition 10. *Consider a proper encoder $(\mathcal{C}, f)$ such that $f(\mathcal{C}) = \{0, 1\}$, with 0 and 1 having the same probability for random input sequences. Let $\bar{\phi}_b$ and H_b^{length} be the average number of retrieved linear equations for a bit b and the associated entropy on the length of the preimage of b. Then, we have*

$$\bar{\phi}_b = E_b - \delta_b^{\phi}, \text{ with } \delta_b^{\phi} = \sum_{n \geq 1} \frac{D_b^n}{2^{n-1}} \min(n, D_b^n - 1),$$

and

$$H_b^{length} = E_b - 1 - \delta_b^H, \text{ with } \delta_b^H = \sum_{n \geq 1} \frac{D_b^n}{2^{n-1}} \log D_b^n.$$

Moreover, δ_b^{ϕ} and δ_b^{ϕ} are both positive, and they satisfy $\delta_b^{\phi} \geq \delta_b^H$.

Proof. The formulas for δ_b^ϕ and δ_b^H both follow from straight computation. Now, we always have $D_b^n \leq 2^n$, so $\log D_b^n$ is always at most n. Moreover, for every integer $x > 1$, we have $x - 1 \geq \log x$. So, for every n such that $D_b^n \neq 0$, we have $\min(n, D_b^n - 1) \geq \log D_b^n$. □

These results link the complexity of equations retrieval attacks with exhaustive reconstruction by way of E_b. As a consequence of this proposition, when 0 and 1 have the same number of preimages of each given length, retrieving L equations has complexity at least $\mathcal{O}(2^{\frac{E-1-\delta^\phi}{E-\delta^\phi}L})$, while exhaustive reconstruction of L bits has complexity $\mathcal{O}(2^{\frac{E-1}{E}L})$. Thus, for $\delta^\phi = 0$, equations retrieval is not more effective than exhaustive reconstruction. This happens only when each bit has at most one preimage of each length.

Suppose now the input sequence is given by a LFSR of length L. It is therefore possible to retrieve L linear equations on the input bits of the LFSR with an attack of average complexity

$$\mathcal{O}(2^{(\frac{H_0^{\text{length}}}{\bar{\phi_0}}+\frac{H_1^{\text{length}}}{\bar{\phi_1}})\frac{L}{2}}),$$

requiring $\mathcal{O}(L)$ consecutive output bits.

Like in the previous attacks, when $\frac{\bar{\phi_0}}{H_0^{\text{length}}} \neq \frac{\bar{\phi_1}}{H_1^{\text{length}}}$ holds, the complexity of the attack can be reduced by seeking for a sequence where mostly bits 0 or 1 appear (depending on the inequality direction). The attack thus obtained has better complexity, but requires the knowledge of more output bits.

Example 3. We consider the ABSG code tree. For every length $n \geq 2$, there is exactly one preimage of 0 and one preimage of 1 of length n. We obtain

$$\bar{\phi_b} = \sum_{n\geq 2} \frac{n}{2^{n-1}} = 3 = E_b, \text{ and } H_b^{\text{length}} = \sum_{n\geq 2} \frac{n-1}{2^{n-1}} = \bar{\phi_b} = H_b.$$

The equations retrieval attack is thus as difficult as exhaustive reconstruction for the ABSG.

FBDD attacks. Krause [13] introduced the FBDD-attack (standing for Free Binary Decision Diagram) which is a cryptanalysis method for LFSR-based generators, i.e., a generator LG that, for each initial state $x \in \{0,1\}^n$, outputs a linear bitstream $LG(x)$, and a compression function which compresses the linear bitstream. The cryptanalysis method relies on two assumptions called the *FBDD Assumption* and the *Pseudo-randomness Assumption* (see [13] for details).

The cost of the cryptanalysis depends on two properties of the compression function that are a parameter γ linked to the maximal length of the sequence output by the compression function when applied on all sequences of length m, and some parameter α (see [13] and some details in [10]); the two parameters α and γ are reals between 0 and 1. Then, the time and space complexity of the FBDD-attack is $L^{\mathcal{O}(1)}2^{\frac{1-\alpha}{1+\alpha}L}$ and it requires $\lceil \gamma\alpha^{-1}L \rceil$ consecutive bits of the keystream in order to compute L consecutive bits of the input sequence.

When the probability that the image of a randomly chosen finite input sequence is a prefix of a given output sequence varies according to the output sequence, it is not clear whether the original FBDD-attack may be improved to be more efficient.

4.3 Optimal choices

In this part, we construct an optimal proper encoder in light of the attacks considered previously.

Requirements based on security analysis. In order to thwart attacks based on asymmetry between the preimage of 0 and that of 1, each output bit must have the same number of preimages of a given length.

Next, in order to maximize the complexity of most probable case attacks while keeping a good output rate, the length of the shortest word in $\mathcal{C}$ should be as close as possible to the average length of the words in $\mathcal{C}$.

Example 3 shows that the ABSG compression mechanism is optimal regarding equations retrieval attacks, meaning that it is not easier to retrieve equations than to reconstruct consecutive bits of the input sequence.

In the general case, equations retrieval attacks can have a better complexity than exhaustive reconstruction. However, as shown in Proposition 10, in order to lessen their efficiency, each bit should have at most one preimage of each length.

Construction of an optimal framework. For an output rate at least $\frac{1}{2}$, the number of choices for the proper encoder are finite, because of symmetry requirements, and the output rate is either equal to 1 or $\frac{1}{2}$ exactly. For $\mathit{Rate} = 1$, there are two proper encoders, with $\mathcal{C} = \{0, 1\}$, which is insecure. For $\mathit{Rate} = \frac{1}{2}$, one can construct 6 proper encoders. The suitable code is $\mathcal{C} = \{00, 01, 10, 11\}$, and the function f is such that 0 and 1 have two preimages each. For each choice, as the length of the preimages is constant, we can apply the equations retrieval attack and solve the corresponding system. The complexity is then $\mathcal{O}(L)$.

Let then h be the minimal depth of leaves in the tree. As each output bit must have the same number of preimages of a given length, the number of preimages of 0 and 1 of depth h in $\mathcal{C}$ is the same. Then, the complexity of reconstruction using the most probable case is $O(\frac{h-1}{h}L)$. In order to maximize the output rate, we have to choose $h = 2$, and no level in the tree should have only internal nodes. This implies that, at every depth more than 2, the tree must have exactly 2 leaves, until the last level with depth d, where it has 4 leaves. We denote by T_2^d the set of code trees of depth d, and exactly 2 leaves of depth $2, 3, \dots, d-1$ (hence 4 leaves of depth d for $d < \infty$). The ABSG code tree belongs to T_2^∞. In order to obtain proper encoders using these codes, one only has to use functions f such that the number of preimages of 0 and 1 of each depth in $\mathcal{C}$ is the same.

This optimal code can be adapted for smaller output rates, beginning at depth $h > 2$. This makes most probable case attacks more complex, though another way of complexifying them is to act on the input sequence using, for instance, a longer LFSR. The tree considered then has exactly 2^{h-1} leaves of depth $h, \dots, d-1$ and maximal depth d (reached by exactly 2^h leaves when d is finite). Notice that the trees T_h^d can be constructed by putting 2^{h-2} trees of T_2^{d+2-h} at depth $h-2$ in a tree with all internal nodes until depth $h-2$.

However, equations retrieval attacks are more efficient for $h > 2$:

Proposition 11. *Consider a proper encoder $(\mathcal{C}, f)$ such that the code tree of $\mathcal{C}$ is a T_h^d tree, and that 0 and 1 have the same number of preimages of each given length. Suppose also that $\mathcal{C}$ is such that the number of equations linking the preimages of b of length n is the least possible, namely $\phi_b^n = \max(0, n+1-D_b^n)$. Then, we have:*

1. *for every $h \geq 2$, the entropy on the length of the preimage of a given output bit b is $H^{length} = 2 - 2^{h+1-d}$,*
2. *for $2 \leq h \leq 4$, we have $\bar{\phi_b} = h + 2 - 2^{h-2} - 2^{h-d} - 2^{2h-d-2}$, which is equal to 3 for $h = 3$ and $d = \infty$.*

Proof. These are the results of straight, yet tedious computations. □

As a consequence of these results, and namely of the entropy remaining less than 2, the complexity of equations retrieval attacks does not grow fast when the output rate decreases. Therefore, the optimal framework against these attacks is reached when the code tree belongs to T_2^∞. However, the attack complexity remains at least $\mathcal{O}(2^{\frac{L}{2}})$ for trees in T_2^d with $d > 2$.

Definition 4. *We say that a proper encoder $(\mathcal{C}, f)$ is an* optimal encoder *if the associated code tree belongs to T_2^∞, and if 0 and 1 have exactly one preimage by f of length ℓ, for $\ell \geq 2$.*

In Table 1, we provide the characteristics of proper encoders constructed on the basis of general T_h^d trees as defined in Proposition 11. We also provide a comparison with the SSG. We left aside polynomial terms in the computational complexity. One should also note than most probable case attacks require much known keystream, whereas the other attacks considered require only a number of bits linear in L, where L is the number of bits we want to retrieve. The results for $FBDD$ attacks are taken from [13, 10] for the SSG and ABSG. Moreover, the complexity of FBDD attacks is the same for all optimal encoders, including the ABSG. We see that equations retrieval attacks are more powerful against T_2^d trees than exhaustive reconstruction, which is why we did not consider them as optimal. However, they may be easier to protect against timing attacks than optimal encoders, because the length of their codewords is bounded.

5 Applications

5.1 Bit-search-based generators

In [9], the BSG algorithm was proposed, and was presented together with the ABSG, which was then described in [10]. Both share the same code tree presented in Figure 1, which belongs to T_2^∞, and thus fits in our framework. The corresponding code is $\mathcal{C} = \{01^k0, 10^k1, k \geq 0\}$.

	Output rate	Exhaustive reconstruction	Most probable case	Equations retrieval	FBDD attacks
T_h^d	$\frac{1}{h+1-2^{h-d}}$	$\frac{h-2^{h-d}}{h+1-2^{h-d}}L$	$\frac{h-1}{h}L$	see Prop.11	n/a
T_h^∞	$\frac{1}{h+1}$	$\frac{h}{h+1}L$	$\frac{h-1}{h}L$	see Prop.11	n/a
T_2^d	$\frac{1}{3-2^{2-d}}$	$\frac{2-2^{2-d}}{3-2^{2-d}}L$	$\frac{1}{2}L$	$\frac{2-2^{3-d}}{3-2^{3-d}}L$	n/a
T_2^∞ (ABSG)	$\frac{1}{3}$	$\frac{2}{3}L$	$\frac{1}{2}L$	$\frac{2}{3}L$	$\simeq 0.532L$
T_3^∞	$\frac{1}{4}$	$\frac{3}{4}L$	$\frac{2}{3}L$	$\frac{2}{3}L$	$\simeq 0.615L$
SSG	$\frac{1}{4}$	$\frac{3}{4}L$	$\frac{1}{2}L$	$\frac{2}{3}L$(see Section 5.2)	$\simeq 0.656L$

Table 1. Characteristics and attack exponent against T_h^d trees filtering LFSRs

In the case of the BSG, the compression function f_{BSG} maps codewords of length 2 to 0, and the other codewords to 1. Therefore, it is not an optimal encoder. This asymmetry resulted in several attacks [10, 11]. For instance, the equations retrieval attack takes advantage of it and it is especially efficient against the BSG, with complexity $\mathcal{O}(2^{\frac{1}{3}L})$.

In the case of the ABSG, the compression function f_{ABSG} maps codewords to their second bit, so it is an optimal encoder. Therefore, the ABSG is optimal against the attacks we described. Their complexity is given in table 1.

5.2 Self-Shrinking Generator

Let us set $\mathcal{C} = \{00, 01, 10, 11\}$, and define $f : \mathcal{C} \to \{0, 1, \varepsilon\}$ by : $f(00) = f(01) = \varepsilon$, $f(10) = 0$ and $f(11) = 1$. The Self-Shrinking Generator is exactly the scheme corresponding to the pair $(\mathcal{C}, f)$ in our framework.

The pair $(\mathcal{C}, f)$ is a proper encoder, but it contains ε. Following the transformation described in Proposition 7, we set $\mathcal{C}' = \{(0\{0,1\})^*1\{0,1\}\}$, and we define $f' : \mathcal{C}' \to \{0, 1\}$ by $f(w) = b$ for $w \in \{(0\{0,1\})^*1b\}$.

The pair $(\mathcal{C}', f')$ is a proper encoder that has an optimal output set and satisfies the symmetry requirement: at every level of the corresponding tree, described in Figure 2, there are exactly as many preimages of 0 and 1.

The SSG is neither an optimal encoder, nor is it optimal among proper encoders having the same output rate. This comes from the fact that one out of two levels in the tree is empty. Let us compare the corresponding scheme to the optimal choice for the same output rate ($\frac{1}{4}$ for the SSG), whose code tree is a T_3^∞ tree. For both schemes, the complexity of the exhaustive reconstruction attack is the same, namely $\mathcal{O}(2^{\frac{3}{4}L})$. However, the complexity of the most probable case attack against the SSG is $\mathcal{O}(2^{\frac{L}{2}})$, requiring $\mathcal{O}(2^{\frac{L}{2}})$ bits of the output. For the T_3^∞ choice, this attack has complexity $\mathcal{O}(2^{\frac{2}{3}L})$, and requires $\mathcal{O}(2^{\frac{2}{3}L})$ output bits. Therefore, the SSG is not optimal against most probable case attacks.

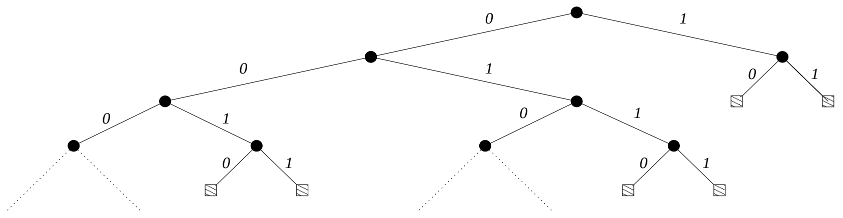

Fig. 2. SSG code tree

Moreover, for each output bit, the input has length $2n$ with probability $\frac{1}{2^n}$, in which case one can recover $n+1$ equations. This yields that 3 equations are known on average, with an entropy of 2. Therefore, the equations retrieval attack has complexity $\mathcal{O}(2^{\frac{2}{3}L})$, which is the same as T_3^∞, but also as T_2^∞ (ABSG). As this attack requires a number of bits linear in L, it is as practical as exhaustive reconstruction. Notice that the equations retrieval attack against the SSG has almost the same complexity as the FBDD attack of Krause [13], while not requiring a large amount of memory.

Therefore, an optimal encoder such as the ABSG is as secure against the attacks considered in this paper as the Self-Shrinking Generator (apart from FBDD-attacks), while providing a better output rate ($\frac{1}{3}$ instead of $\frac{1}{4}$).

6 Conclusion and further work

In this paper, we have extensively studied how to compress efficiently and securely the output of pseudo-random generators. It turns out that the ABSG, which was introduced in [9, 10], and is part of the DECIM proposal to the ECRYPT stream cipher project [5], has the optimal properties against several well-known attacks. But it is also possible to design several other optimal encoders with the same properties, using code trees taken from the T_2^∞ infinite family. At last, we have also shown compression components based on these trees are almost as secure as the Self-Shrinking Generator [15], while providing an output rate of $\frac{1}{3}$ instead of $\frac{1}{4}$. We consider two main directions for research in this area. First, one could use another generator to choose the compression function at each iteration, while keeping the same code tree. The idea is thus to generalize this framework by using other pseudo-random generators to control compression. This should provide us with comparisons with the Shrinking Generator [3]. Second, if the compression function and the code are chosen properly, a compression component may also erase the bias of a pseudo-random generator that does not produce every bit sequence with equal probability. It then seems possible to construct a general design for bias-erasing compression.

References

1. F. Armknecht, M. Krause, *Algebraic Attacks on Combiners with Memory*, Advances in Cryptology – CRYPTO'03 Proceedings, LNCS **2729**, Springer-Verlag, (2003), 162–176.
2. J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, (1985).
3. D. Coppersmith, H. Krawczyk, Y. Mansour, *The Shrinking Generator*, Advances in Cryptology – CRYPTO'93 Proceedings, LNCS **773**, Springer-Verlag, (1993), 22–39.
4. N. Courtois, W. Meier, *Algebraic Attacks on Stream Ciphers with Linear Feedback* Advances in Cryptology – EUROCRYPTO'03 Proceedings, LNCS **2656**, Springer-Verlag, (2003), 345–359.
5. eStream, Stream cipher project of the European Network of Excellence in Cryptology ECRYPT, `http://www.ecrypt.eu.org/stream/`.
6. P. Ekdahl, T. Johansson, W. Meier, *Predicting the Shrinking Generator with Fixed Connections*, Advances in Cryptology – EUROCRYPT 2003 Proceedings, LNCS **2656**, Springer-Verlag, E. Biham, ed., (2003), 330–344.
7. P. Ekdahl, T. Johansson, W. Meier, *A note on the Self-Shrinking Generator*, In *Proc. of International Symposium on Information Theory*, page 166, IEEE, (2003).
8. S. Golomb, *Shift Register Sequences*, Revised Edition, Aegean Park Press, (1982).
9. A. Gouget, H. Sibert, *The Bit-Search Generator*, In *The State of the Art of Stream Ciphers: Workshop Record, Brugge, Belgium, October 2004*, pages 60–68, (2004).
10. A. Gouget, H. Sibert, C. Berbain, N. Courtois, N. Debraize and C. Mitchell, *Analysis of the Bit-Search Generator and sequence compression techniques*, Proceedings of FSE'05, LNCS **3557**, Springer-Verlag, (2005).
11. M. Hell, T. Johansson, *Some attacks on the Bit-Search Generator* Proceedings of FSE'05, LNCS **3557**, Springer-Verlag, (2005).
12. P. Kocher, *Timings attacks on implementations of Diffie–Hellman, RSA, DSS and other systems*, Proceedings of Crypto 1996, LNCS **1109**, Springer-Verlag, (1996).
13. M. Krause. *BDD-based Cryptanalysis of Keystream Generators*, In EUROCRYPT 2002, pp. 222-237, LNCS 2332, Springer, (2002).
14. I. Kessler, H. Krawczyk, *Minimum Buffer Length and Clock Rate for the Shrinking Generator Cryptosystem*, IBM Research Report, RC 19938 (88322), (1995).
15. W. Meier, O. Staffelbach, *The Self-Shrinking Generator*, Advances in Cryptology – EUROCRYPT'94 Proceedings, LNCS **950**, Springer-Verlag, (1994), 205–214.
16. R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, (1986).

A Choice of the output set: non-prefix case

We consider the case where $f(\mathcal{C})$ is not a prefix code and does not contain the empty word ε. Thus, it is no longer possible to determine with certainty each word $(f(w_i))$ of the image of the input sequence. For statistical reasons, $f(\mathcal{C})$ contains at least one word beginning with 0, and one beginning with 1. Moreover, as it is not prefix, it also contains two words beginning with the same bit. Therefore, $f(\mathcal{C})$ contains at least three elements.

Proposition 12. *Let $(\mathcal{C}, f)$ be a proper encoder such that $f(\mathcal{C})$ is a non-prefix set that does not contain the empty word ε. Then, the average expected length of the first word of the decomposition of the input sequence over $\mathcal{C}$ is $E(\mathcal{C})$ when the input sequence is chosen uniformly. This word is known with average entropy $E(\mathcal{C}) + \Delta(\mathcal{C})$, with $\Delta(\mathcal{C}) < -1$.*

Proof. For $x \in f(\mathcal{C})$, we denote by $\mathcal{C}_x$ the set of preimages of x in $\mathcal{C}$, and we define $P(x) = \sum_{w \in \mathcal{C}_x} \frac{1}{2^{|w|}}$. Let $\mathbf{y}$ be the output sequence corresponding to a randomly chosen input sequence $\mathbf{s}$. Let $\mathcal{C}(\mathbf{y})$ be the set of words of $\mathcal{C}$ such that, for every $w \in \mathcal{C}(\mathbf{y})$, the sequence $\mathbf{y}$ begins with w.

Let $P_{\mathbf{y}}(x)$ denote the probability that the image by f of the first element of $\mathcal{C}$ recognized in $\mathbf{s}$ is x, given $\mathbf{y}$. Then, we have $\displaystyle\sum_{x \in \mathcal{C}(\mathbf{y})} P_{\mathbf{y}}(x) = 1$.

Now, each element in $\mathcal{C}_x$ has probability $\frac{1}{P(x)2^{|w|}}$ to be the preimage of x. Thus, the average length of the first element of $\mathcal{C}$ recognized in $\mathbf{s}$, given $\mathbf{y}$, is $\displaystyle E_{\mathbf{y}}(\mathcal{C}) = \sum_{x \in \mathcal{C}(\mathbf{y})} P_{\mathbf{y}}(x) \sum_{w \in \mathcal{C}_x} \frac{|w|}{P(x)2^{|w|}}$.

Output sequences are chosen following the uniform distribution on input sequences, so the average length of the first element of $\mathcal{C}$ recognized in a random input sequence knowing the output is $E(\mathcal{C}) = \sum_{w \in \mathcal{C}} \frac{|w|}{2^{|w|}}$. Hence, the average value of $E_{\mathbf{y}}$ for random input sequences is $E(\mathcal{C})$, which is the first result.

Next, the entropy on the first element of $\mathcal{C}$ recognized in $\mathbf{s}$, given $\mathbf{y}$, is:

$$\begin{aligned} H_{\mathbf{y}}(\mathcal{C}) &= - \sum_{x \in \mathcal{C}(\mathbf{y})} \sum_{w \in \mathcal{C}_x} P_{\mathbf{y}}(x) \frac{1}{P(x)2^{|w|}} \log \frac{P_{\mathbf{y}}(x)}{P(x)2^{|w|}} \\ &= \sum_{x \in \mathcal{C}(\mathbf{y})} \sum_{w \in \mathcal{C}_x} \frac{P_{\mathbf{y}}(x)}{P(x)2^{|w|}} (|w| + \log P(x) - \log P_{\mathbf{y}}(x)) \\ &= E_{\mathbf{y}}(\mathcal{C}) + \sum_{x \in \mathcal{C}(\mathbf{y})} P_{\mathbf{y}}(x)(\log P(x) - \log P_{\mathbf{y}}(x)). \end{aligned}$$

The average value of $H_{\mathbf{y}}$ for uniformly chosen input sequences is thus the sum of $E(\mathcal{C})$ and of the average value of $\displaystyle \Delta_{\mathbf{y}}(\mathcal{C}) = \sum_{x \in \mathcal{C}(\mathbf{y})} P_{\mathbf{y}}(x)(\log P(x) - \log P_{\mathbf{y}}(x))$. for random input sequences. Let b be the first bit of $\mathbf{y}$. Then, $\mathcal{C}(\mathbf{y})$ is included in the subset $\mathcal{C}(b*)$ of $\mathcal{C}$ consisting of those words that are mapped by f to a word whose first bit is b. For statistical reasons, we have $\sum_{x \in \mathcal{C}(b*)} \frac{1}{2^{|x|}} = \frac{1}{2}$, which yields

$$\sum_{x \in \mathcal{C}(\mathbf{y})} \frac{1}{2^{|x|}} \leq \frac{1}{2} \quad . \tag{1}$$

Moreover, as there are at least 3 elements in $f(\mathcal{C})$, there are some output sequences $\mathbf{y}$ such that the inequality in Equation (1) is strict.

Using equality $\sum_{x \in \mathcal{C}(\mathbf{y})} P_{\mathbf{y}}(x) = 1$ and inequality (1), we obtain $\Delta_{\mathbf{y}} \leq -1$ for every output $\mathbf{y}$, the inequality being strict when that of (1) is. Therefore, the average value of $\Delta_{\mathbf{y}}$ for all random input sequences is strictly less than -1. □

Hence, the optimal choice of the output set, even if non-prefix output sets are considered, is still $f(\mathcal{C}) = \{0, 1\}$, with 0 and 1 having probability $\frac{1}{2}$ to appear in the output sequence for a random input sequence.