Stream Ciphers Using a Random Update Function: Study of the Entropy of the Inner State

Andrea Röck

Team SECRET, INRIA Paris-Rocquencourt, France
andrea.roeck@inria.fr,
http://www-rocq.inria.fr/secret/Andrea.Roeck

Abstract. Replacing random permutations by random functions for the update of a stream cipher introduces the problem of entropy loss. To assess the security of such a design, we need to evaluate the entropy of the inner state. We propose a new approximation of the entropy for a limited number of iterations. Subsequently, we discuss two collision attacks which are based on the entropy loss. We provide a detailed analysis of the complexity of those two attacks as well as of a variant using distinguished points.

Keywords: Entropy, Random Functions, Stream Cipher, Collisions.

1 Introduction

Recently, several stream ciphers have been proposed with a non-bijective update function. Moreover, in some cases the update function seems to behave like a random function as for the Mickey stream cipher [BD05]. Using a random function instead of a random permutation induces an entropy loss in the state. An attacker might exploit this fact to mount an attack. Particularly, we will study some attacks which apply the approach of Time–Memory tradeoff [Hel80] and its variants [HS05]. At first we introduce the model with which we are going to work.

Stream cipher model. The classical model of an additive synchronous stream cipher (Fig. 1) is composed of an internal state updated by applying a function Φ. Then a filter function is used to extract the keystream bits from the internal state. To obtain the ciphertext we combine the keystream with the plaintext.

The particularity of our model is that Φ is a random mapping which allows us to make some statistical statements about the properties of the stream cipher.

Definition 1. *Let $\mathcal{F}_n = \{\varphi \mid \varphi : \Omega_n \rightarrow \Omega_n\}$ be the set of all functions which map a set $\Omega_n = \{\omega_1, \omega_2, \ldots, \omega_n\}$ of n elements onto itself. We say that Φ is a* random function *or a* random mapping *if it takes each value $\varphi \in \mathcal{F}_n$ with the same probability $Pr[\Phi = \varphi] = 1/n^n$.*

© Springer-Verlag Berlin Heidelberg 2008

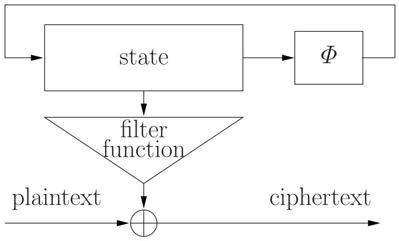

Fig. 1. Model of a simple stream cipher

For an extended definition of a random function we refer to the book of Kolchin [Kol86].

Let S_k be the random variable denoting the value of the state after k iterations of Φ, for $k \geq 0$. From the model in Fig. 1 we see that $S_k = \Phi(S_{k-1})$ where the value of Φ is the same for all iterations $k > 1$. The probability distribution of the initial state S_0 is $\{p_i\}_{i=1}^n$ such that

$$p_i = Pr[S_0 = \omega_i] \ .$$

If we do not state otherwise, we assume a uniform distribution thus $p_i = 1/n$ for all $1 \leq i \leq n$. By

$$p_i^{\Phi}(k) = Pr[\Phi^k(S_0) = \omega_i]$$

we describe the probability of the state being ω_i after applying k times Φ on the initial state S_0. If we write only $\mathrm{p}_i^{\varphi}(k)$ we mean the same probability but for a specific function $\varphi \in \mathcal{F}_n$. The notation above allows us to define the entropy of the state after k iterations of Φ

$$H_k^{\Phi} = \sum_{i=1}^{n} p_i^{\Phi}(k) \log_2 \left(\frac{1}{p_i^{\Phi}(k)} \right) \ .$$

If $p_i^{\Phi}(k) = 0$ we use the classical convention in the computation of the entropy that $0 \log_2(\frac{1}{0}) = 0$. This can be done, since a zero probability has no influence on the computation of the entropy. In this article we are interested in expectations where the average is taken over all functions $\varphi \in \mathcal{F}_n$. To differentiate between a value corresponding to a random mapping Φ, to a specific function φ, and the expectation of a value, taken over all functions $\varphi \in \mathcal{F}_n$, we will write in the following the first one normal (*e.g.* H_k^{Φ}), the second one upright (*e.g.* H_k^{φ}), and the last one bold (*e.g.* $\mathbf{H_k}$). For instance, the formula:

$$\mathbf{H_k} = \mathbf{E}(H_k^{\Phi})$$

denotes the expected state entropy after k iterations.

The subsequent article is divided in two main sections. In Section 2, we discuss ways of estimating the state entropy of our model. We give a short overview of previous results from [FO90a] and [HK05] in Section 2.1. Subsequently in

Section 2.2, we present a new estimate which is, for small numbers of iterations, more precise than the previous one. In Section 3, we examine if it is possible to use the entropy loss in the state to launch an efficient attack on our model. We discuss two collision attacks against Mickey version 1 [BD05] presented in [HK05]. We give a detailed evaluation of the costs of these attacks applied on our model. For this evaluation we consider the space complexity, the query complexity and the number of different initial states needed. By the space complexity we mean the size of the memory needed, by the query complexity we mean the number of times we have to apply the update function during the attack. For the first attack, we show that we only gain a factor on the space complexity by increasing the query complexity by the same factor. For the second attack, we demonstrate that, contrary to what is expected from the results in [HK05], the complexities are equivalent to a direct collision search in the initial values. In the end, we present a new variant of these attacks which allows to reduce the space complexity; however the query complexity remains the same.

2 Estimation of Entropy

The entropy is a measure of the unpredictability. An entropy loss in the state facilitates the guessing of the state for an adversary. In this section, we therefore discuss different approaches to estimate the expected entropy of the inner state.

2.1 Previous Work

Flajolet and Odlyzko provide, in [FO90a], a wide range of parameters of random functions by analyzing their functional graph. A functional graph of a specific

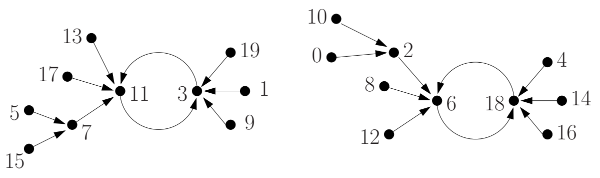

Fig. 2. Example of a functional graph for $\varphi : x \mapsto x^2 + 2 \pmod{20}$

function φ is a graph which has a directed edge from vertex x to vertex y if and only if $\varphi(x) = y$. An example for $\varphi(x) = x^2 + 2 \pmod{20}$ can be seen in Fig. 2. For functions on a finite set of elements, such a graph consists of one or more separated components, where each component is build by a cycle of trees, i.e. the nodes in the cycle are the root of a tree.

To find the expected value of a given parameter of a random function, Flajolet and Odlyzko construct the generating function of the functional graph associated with this parameter. Subsequently, they obtain an asymptotic value of the expectation by means of a singularity analysis of the generating function. All asymptotic values are for n going to $+\infty$. In the following, we present some examples of the examined parameters. The maximal tail length is, for each graph, the maximal number of steps before reaching a cycle. An r–node is a node in the graph with exactly r incoming nodes which is equivalent to a preimage of size r. By the image points we mean all points in the graph that are reachable after k iterations of the function. The asymptotic values of these parameters are:

- the expected number of cycle point $\mathbf{cp}(n) \sim \sqrt{\pi n/2}$,
- the expected maximal tail length $\mathbf{mt}(n) \sim \sqrt{\pi n/8}$,
- the expected number of r–nodes $\mathbf{rn}(n, r) \sim \frac{n}{r!e}$ and
- the expected number of image points after k iterations $\mathbf{ip}(n, k) \sim n(1 - \tau_k)$ where $\tau_0 = 0$ and $\tau_{k+1} = e^{-1+\tau_k}$.

For all these values, the expectation is taken over all functions in $\mathcal{F}_n$.

In [HK05], Hong and Kim use the expected number of image points to give an upper bound for the state entropy after k iterations of a random function. They utilize the fact that the entropy is always less or equal than the logarithm of the number of points with probability larger than zero. After a finite number of steps, each point in the functional graph will reach a cycle, and thus the number of image points can never drop below the number of cycle points. Therefore, the upper bound for the estimated entropy of the internal state

$$\mathbf{H_k} \leq \log_2(n) + \log_2(1 - \tau_k) \tag{1}$$

is valid only as long as $\mathbf{ip}(n, k) > \mathbf{cp}(n)$. We see that for this bound the loss of entropy only depends on k and not on n.

In Fig. 3 we compare, for $n = 2^{16}$, the values of this bound with the empirically derived average of the state entropy.

To compute this value we chose 10^4 functions, using the HAVEGE random number generator [SS03], and computed the average entropy under the assumption of a uniform distribution of the initial state. Even if n is not very big, it is sufficient to understand the relation between the different factors. We can see in the graph that if k stays smaller than $\mathbf{mt}(n)$ this bound stays valid and does not drop under $\log_2(\mathbf{cp}(n))$.

2.2 New Entropy Estimation

The expected number of image points provides only an upper bound (1) for the expected entropy. We found a more precise estimation by employing the methods stated in [FO90a].

For a given function $\varphi \in \mathcal{F}_n$, let ω_i be a node with r incoming nodes (an r–node). The idea is that this is equivalent to the fact that ω_i is produced by exactly r different starting values after one iteration. Thus, if the initial distribution of the state is uniform, this node has the probability $\mathrm{p}_i^{\varphi}(1) = r/n$. The same idea works also for more than one iteration.

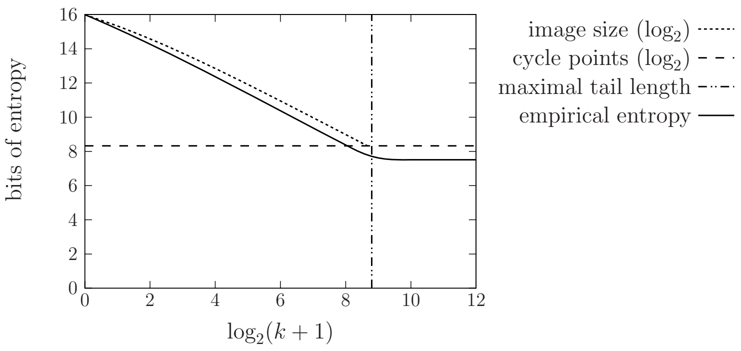

Fig. 3. Upper bound and empirical average of the entropy for $n = 2^{16}$

Definition 2. *For a fixed n let us choose a function $\varphi \in \mathcal{F}_n$. Let $\varphi^{-k}(i) = \{j|\varphi^k(\omega_j) = \omega_i\}$ define the* preimage *of i after k iterations of φ. By $\mathrm{rn}_k^\varphi(r) = \#\{i|\ |\varphi^{-k}(i)| = r\}$ we denote the number of points in the functional graph of φ which are reached by exactly r nodes after k iterations.*

For a random function Φ on a set of n elements, we define by $\mathbf{rn_k}(n, r)$ the expected value of $\mathrm{rn}_k^\varphi(r)$, thus

$$\mathbf{rn_k}(n, r) = \frac{1}{n^n} \sum_{\varphi \in \mathcal{F}_n} \mathrm{rn}_k^\varphi(r) \ .$$

A small example might help to better understand these definitions. For $n = 13$

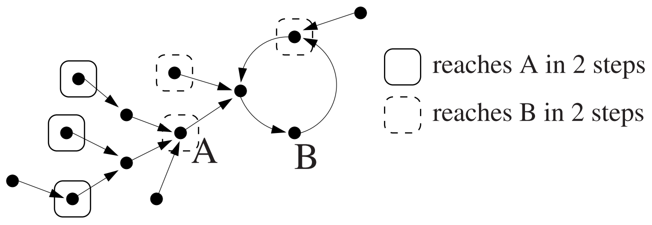

Fig. 4. Example of a functional graph to illustrate $\mathrm{rn}_k^\varphi(r)$

we consider a function φ with a functional graph as displayed in Fig. 4. The only points that are reached by $r = 3$ points after $k = 2$ iterations are A and B. Thus, in this case we have $\mathrm{rn}_2^\varphi(13, 3) = 2$. The value $\mathbf{rn_2}(13, 3)$ is then the average taken over all functions $\varphi \in \mathcal{F}_{13}$.

Using Def. 2 we can state the following theorem.

Theorem 1. *In the case of a uniform initial distribution the expected entropy of the inner state after k iterations is*

$$\mathbf{H_k} = \log_2(n) - \sum_{r=1}^{n} \mathbf{rn_k}(n,r)\,\frac{r}{n}\,\log_2(r)\ . \tag{2}$$

Proof. Let us fix a function φ. We use the idea that after k iterations of φ we have $\mathrm{rn}_k^{\varphi}(r)$ states with probability $\frac{r}{n}$. Thus, the entropy after k iterations for this specific function is

$$\begin{aligned}\mathrm{H}_k^{\varphi} &= \sum_{r=1}^{n} \mathrm{rn}_k^{\varphi}(r)\frac{r}{n}\log_2\left(\frac{n}{r}\right)\\ &= \log_2(n)\frac{1}{n}\sum_{r=1}^{n} r\,\mathrm{rn}_k^{\varphi}(r) - \sum_{r=1}^{n}\mathrm{rn}_k^{\varphi}(r)\frac{r}{n}\log_2(r)\ .\end{aligned}$$

We ignore the case $r = 0$ since it corresponds to a probability zero, which is not important for the computation of the entropy. Each $1 \leq j \leq n$ appears exactly in one preimage of φ after k iterations. We can thus see directly from the definition of $\mathrm{rn}_k^{\varphi}(r)$ that $\sum_{r=1}^{n} r\,\mathrm{rn}_k^{\varphi}(r) = n$. Therefore, we can write

$$\mathrm{H}_k^{\varphi} = \log_2(n) - \sum_{r=1}^{n}\mathrm{rn}_k^{\varphi}(r)\frac{r}{n}\log_2(r)\ .$$

By using this equation, we can give the expected entropy after k iterations as

$$\begin{aligned}\mathbf{H_k} &= \frac{1}{n^n}\sum_{\varphi\in\mathcal{F}_n}\mathrm{H}_k^{\varphi}\\ &= \frac{1}{n^n}\sum_{\varphi\in\mathcal{F}_n}\left[\log_2(n) - \sum_{r=1}^{n}\mathrm{rn}_k^{\varphi}(r)\frac{r}{n}\log_2(r)\right]\\ &= \log_2(n) - \frac{1}{n^n}\sum_{\varphi\in\mathcal{F}_n}\left[\sum_{r=1}^{n}\mathrm{rn}_k^{\varphi}(r)\frac{r}{n}\log_2(r)\right]\ .\end{aligned}$$

Since we only have finite sums we can change the order:

$$\mathbf{H_k} = \log_2(n) - \sum_{r=1}^{n}\left[\frac{1}{n^n}\sum_{\varphi\in\mathcal{F}_n}\mathrm{rn}_k^{\varphi}(r)\right]\frac{r}{n}\log_2(r)\ .$$

We conclude our proof by applying Def. 2.

In the same way we can compute the entropy for any arbitrary initial distribution.

Theorem 2. *For a given n, let $P = \{p_1, p_2, \ldots, p_n\}$ define the distribution of the initial state. Then, the expected entropy of the state after k iterations is given by*

$$\mathbf{H}_{\mathbf{k}}^{\mathbf{P}} = \sum_{r=1}^{n} \mathbf{rn}_{\mathbf{k}}(n,r) \frac{1}{\binom{n}{r}} \sum_{1 \le j_1 < \cdots < j_r \le n} (p_{j_1} + \cdots + p_{j_r}) \log_2 \frac{1}{p_{j_1} + \cdots + p_{j_r}} . \quad (3)$$

Proof. Let us choose a specific φ and an index i. After k iterations of φ, the state ω_i has the probability $\sum_{j \in \varphi^{-k}(i)} p_j$. Therefore, the expected entropy after k iterations is given by

$$\mathbf{H}_{\mathbf{k}}^{\mathbf{P}} = \frac{1}{n^n} \sum_{\varphi \in \mathcal{F}_n} \sum_{i=1}^{n} \left(\sum_{j \in \varphi^{-k}(i)} p_j \right) \log_2 \frac{1}{\sum_{j \in \varphi^{-k}(i)} p_j} . \quad (4)$$

For a given r we fix a set of indices $\{j_1, \ldots, j_r\}$. Without loss of generality we assume that they are ordered, *e.i.* $1 \le j_1 < \cdots < j_r \le n$. We now want to know how many times we have to count $(p_{j_1} + \cdots + p_{j_r}) \log_2 \frac{1}{p_{j_1} + \cdots + p_{j_r}}$ in (4). This is equivalent to the number of pairs (i, φ) where $\varphi^{-k}(i) = \{j_1, \ldots, j_r\}$.

From Def. 2 we know that $n^n \mathbf{rn}_{\mathbf{k}}(n,r)$ is the number of pairs (i, φ) such that $|\varphi^{-k}(k)| = r$. Due to symmetry, each set of indices of size r is counted the same number of times in (4). There are $\binom{n}{r}$ such sets. Thus, $(p_{j_1} + \cdots + p_{j_r}) \log_2 \frac{1}{p_{j_1} + \cdots + p_{j_r}}$ is counted exactly $\frac{n^n \mathbf{rn}_{\mathbf{k}}(n,r)}{\binom{n}{r}}$ times and we can write

$$\mathbf{H}_{\mathbf{k}}^{\mathbf{P}} = \frac{1}{n^n} \sum_{r=1}^{n} \frac{n^n \mathbf{rn}_{\mathbf{k}}(n,r)}{\binom{n}{r}} \sum_{1 \le j_1 < \cdots < j_r \le n} (p_{j_1} + \cdots + p_{j_r}) \log_2 \frac{1}{p_{j_1} + \cdots + p_{j_r}} ,$$

which is equivalent to (3).

Theorem 1 can also be shown by using Theorem 2; however the first proof is easier to follow. Finally, we want to consider a further special case.

Corollary 1. *For a given n let the distribution of the initial state be $P_m = \{p_1, p_2, \ldots, p_n\}$. From the n possible initial values only m occur with probability exactly $\frac{1}{m}$. Without loss of generality we define*

$$p_i = \begin{cases} \frac{1}{m} & 1 \le i \le m \\ 0 & m < i \le n . \end{cases}$$

In this case we get

$$\mathbf{H}_{\mathbf{k}}^{\mathbf{P_m}} = \sum_{r=1}^{n} \mathbf{rn}_{\mathbf{k}}(n,r) \frac{1}{\binom{n}{r}} \sum_{\ell=0}^{r} \binom{m}{\ell} \binom{n-m}{r-\ell} \frac{\ell}{m} \log_2 \frac{m}{\ell} . \quad (5)$$

Proof. For a given r, let us consider the sum $(p_{j_1} + \cdots + p_{j_r})$ for all possible index tuples $1 \le j_1 < \cdots < j_r \le n$. In $\binom{m}{\ell}\binom{n-m}{r-\ell}$ cases we will have $(p_{j_1} + \cdots + p_{j_r}) = \frac{\ell}{m}$ for $0 \le \ell \le r$. Thus, (5) follows directly from Theorem 2.

To approximate $\mathbf{rn_k}(n,r)$ for one iteration, we use directly the results for the expected number of r–nodes, given in [FO90a], since $\mathbf{rn_1}(n,r) = \mathbf{rn}(n,k) \sim \frac{n}{r!e}$. We can see that already for $k = 1$, a uniform initial distribution, and n large enough, there is a non negligible difference between our estimate (2)

$$\mathbf{H_1} \sim \log_2(n) - e^{-1} \sum_{r=1}^{n} \frac{1}{(r-1)!} \log_2(r) \approx \log_2(n) - 0.8272$$

and the upper bound (1)

$$\mathbf{H_1} \leq \log_2(n) + \log_2(1 - e^{-1}) \approx \log_2(n) - 0.6617 .$$

For more than one iteration we need to define a new parameter.

Theorem 3. *For $n \to \infty$ we can give the following asymptotic value*

$$\mathbf{rn_k}(n,r) \sim n\, c_k(r) \tag{6}$$

of the expected number of points in the functional graph which are reached by r points after k iterations, where

$$c_k(r) = \begin{cases} \frac{1}{r!e} & \textit{for } k = 1 \\ D(k,r,1) f_1(k) \frac{1}{e} & \textit{for } k > 1 \end{cases}$$

$$D(k,r,m) = \begin{cases} 1 & \textit{for } r = 0 \\ 0 & \textit{for } 0 < r < m \\ \sum_{u=0}^{\lfloor r/m \rfloor} \frac{c_{k-1}(m)^u}{u!} D(k, r - mu, m+1) & \textit{otherwise} \end{cases}$$

and

$$f_1(k) = \begin{cases} 1 & \textit{for } k = 1 \\ e^{e^{-1} f_1(k-1)} & \textit{for } k > 1 . \end{cases}$$

Proof. The concept of this proof is that we see the functional graph as a combinatorial structure. We are going to build the generating function corresponding to this structure where we mark a desired parameter. By means of the singularity analysis of the generating function we obtain the asymptotic value of this parameter. The difficulty is to mark the right property in the generation function. The rest of the proof is just following the method described in [FO90a].

For an arbitrary structure, let a_n define the number of elements of this structure with size n for $n \geq 1$. Then, the exponential generating function of the infinite sequence $\{a_n\}_{n \geq 1}$ is defined as

$$A(z) = \sum_{n \geq 1} a_n \frac{z^n}{n!} .$$

By $[z_n]A(z)$ we mean the n'th coefficient a_n of $A(z)$. The nice property of a generating function is that many combinatorial constructions on the structure correspond to simple manipulation of the generating function. We refer the reader to [FS96] for a deeper introduction to the area of generating functions.

The functional graph of a function which maps a finite set onto itself can be described in the following recursive way:

$$\begin{aligned} FuncGraph &= SET(Component)\ , \\ Component &= CYCLE(Tree)\ , \\ Tree &= Node \times SET(Tree)\ . \end{aligned}$$

Each of this constructions: SET, $CYCLE$ and $\times$ (concatenation) can be applied directly on a generating function.

We are interested in the average value of a specific parameter, where the average is taken over all functions of size n. For this purpose we need a bivariate generating function. Let $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ be the set of all functions which map a finite set onto itself. For a specific $\varphi \in \mathcal{F}$, we denote by $|\varphi|$ the size n of the finite set. With $\xi(\varphi)$ we define a specific property of the function. In our case we are interested in $\xi_{r,k}(\varphi) = \mathrm{rn}_k^{\varphi}(|\varphi|, r)$. The bivariate generating function for this parameter, marked by the variable u, is then defined by

$$\xi_{r,k}(u,z) = \sum_{\varphi\in\mathcal{F}} u^{\xi_{r,k}(\varphi)} \frac{z^{|\varphi|}}{|\varphi|!}\ .$$

By $\xi_{r,k,n} = \sum_{\varphi\in\mathcal{F}_n} \xi_{r,k}(\varphi)$ we mean the sum of $\xi_{r,k}(\varphi)$ taken over all $\varphi \in \mathcal{F}_n$. Let

$$\varXi(z) = \sum_{n\geq 1} \xi_{r,k,n} \frac{z^n}{n!}$$

be the generating function for $\xi_{r,k,n}$. Thus, it is clear that the average value of $\xi^{k,r}$ is given by

$$\mathbf{E}(\xi^{k,r}|\mathcal{F}_n) = \frac{\xi_n^{k,r}}{n^n} = \frac{n!}{n^n}[z^n]\varXi(z)\ .$$

To obtain the function $\varXi(z)$ we use that

$$\varXi(z) = \frac{\partial}{\partial_u} \xi_{r,k}(u,z)\bigg|_{u=1}\ .$$

Since in our case the evaluation of $[z^n]\varXi(z)$ is not directly possible, we can use a singularity analysis to get an asymptotic value for $n \to \infty$. More information about singularity analysis can be found in [FO90a] and [FO90b].

We now have to define the function $\xi_{r,k}(u,z)$. For this, we start by a tree. A node in a tree which is reached by r nodes after k iterations can be described by the concatenation of three elements:

1. A *node*.
2. A *SET* of *trees* where each tree has a depth smaller than $k-1$.
3. A *concatenation* of j *trees* where the order of the concatenation is not important and where $1 \leq j \leq r$. Each of these trees has a depth larger or equal to $k-1$ and their roots are reached by respectively $i_1, \ldots, i_j$ nodes after $k-1$ iterations such that $i_1 + \cdots + i_j = r$.

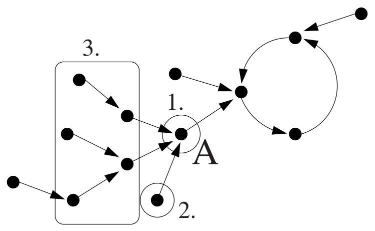

Fig. 5. Example of the structure explained in 1.-3. for the node A

In Fig. 5 these three elements are marked for the node A.

To write the corresponding generating function we need some notations:

The generating function of a set of trees of depth smaller than $k-1$, as described in 2., is given by $f_1(k,z)$ where

$$f_1(k,z) = \begin{cases} 1 & \text{for } k = 1 \\ e^{z\, f_1(k-1,z)} & \text{for } k > 1 \ . \end{cases}$$

By $Par(r)$ we mean the integer partition of r, *i.e.* the set of all possible sequences $[i_1, \ldots, i_j]$ for $1 \leq j \leq r$ such that $1 \leq i_1 \leq \cdots \leq i_j \leq r$ and $i_1 + \cdots + i_j = r$. For example, for $r = 4$ we have $Par(4) = \{[1,1,1,1], [1,1,2], [2,2], [1,3], [4]\}$.

Since the order of the concatenation in 3. is not important, we need a correction term $f_2([i_1, \ldots, i_j])$. If there are some $i_{x_1}, \ldots, i_{x_\ell}$ with $1 \leq x_1 < \cdots < x_\ell \leq j$ and $i_{x_1} = \cdots = i_{x_\ell}$ we have to multiply by a factor $1/\ell!$ to compensate this repeated appearance, e.g. $f_2([1,1,1,1,2,2,3]) = \frac{1}{4!2!1!}$.

Let $t_{r,k}(u,z)$ be the generation function of a tree. By $c_k(r,z)$ we define a variable such that

$$c_k(r,z) t_{r,k}(u,z)^r$$

is the generating function of a tree where the root is reached by r nodes after k iterations. For $k = 1$, such a tree has r children, where each child is again a tree. In terms of generating functions, this structure can be represented by $z\frac{t_{r,k}(u,z)^r}{r!}$. Thus, we get

$$c_1(r,z) = \frac{z}{r!} \ .$$

For $k > 1$ we can use the structure given in 1.-3. and our notations to write:

$$\begin{aligned} & c_k(r,z) \\ &= \tfrac{1}{t_{r,k}(u,z)^r} \overbrace{z}^{1.} \overbrace{f_1(k,z)}^{2.} \\ & \overbrace{\sum_{[i_1,\ldots,i_j] \in Par(r)} \left[c_{k-1}(i_1,z) t_{r,k}(u,z)^{i_1}\right] \cdots \left[c_{k-1}(i_j,z) t_{r,k}(u,z)^{i_j}\right] f_2([i_1,\ldots,i_j])}^{3.} \\ &= z f_1(k,z) \textstyle\sum_{[i_1,\ldots,i_j] \in Par(r)} c_{k-1}(i_1,z) \cdots c_{k-1}(i_j,z) f_2([i_1,\ldots,i_j]) \ . \end{aligned}$$

In total we get

$$c_k(r,z) = \begin{cases} z/r!, \text{ for } k=1 \\ \\ z\ f_1(k,z) \displaystyle\sum_{[i_1,\ldots,i_j]\in Par(r)} c_{k-1}(i_1,z)\ \cdots\ c_{k-1}(i_j,z)\ f_2([i_1,\ldots,i_j]) \\ \text{for } k>1\ . \end{cases} \tag{7}$$

We can now write the generation function of a tree where we mark with the variable u the nodes which are reached by r other nodes after k iterations.

$$t_{r,k}(u,z) = ze^{t_{r,k}(u,z)} + (u-1)t_{r,k}(u,z)^r c_k(r,z)\ ,$$

The first part describes a tree as a node concatenated with a set of trees. The second part correspond to our desired parameter. By applying the properties that a graph of a random function is a *set of components* where each component is a *cycle of trees* we get the generating function for a general functional graph

$$\xi_{r,k}(u,z) = \frac{1}{1-t_{r,k}(u,z)}\ .$$

Now, we can follow the steps as described at the beginning of this proof. We will use the fact that the general generating function of a tree $t_{r,k}(1,z) = t(z) = ze^z$ has a singularity expansion

$$t(z) = 1 - \sqrt{2}\sqrt{1-ez} - \frac{1}{3}(1-ez) + O((1-ez)^{3/2})$$

for z tending to e^{-1}. Finally, by applying the singularity analysis for $z \to e^{-1}$ we get

$$\mathbf{E}(\xi_{r,k}|\mathcal{F}_n) \sim n\ c_k(r,e^{-1})\ .$$

Remark 1. In the construction of our generating function we only count the nodes in the *tree* which are reached by r points after k iterations (*e.g.* node A in Fig. 4). We ignore the nodes on the cycle (*e.g.* node B in Fig. 4). However, the average proportion of the number of cycle points in comparison to the image size after k iterations is

$$\frac{\mathbf{cp}(n)}{\mathbf{ip}(n,k)} \sim \frac{\sqrt{\pi n/2}}{n(1-\tau_k)}\ .$$

For a fixed k and $n \to \infty$ it is clear that this proportion gets negligible.

Thus, we can write

$$\mathbf{rn_k}(n,r) \sim n\ c_k(r,e^{-1})\ .$$

The computation of $c_k(r,e^{-1})$ as defined in (7) is not very practical. In this paragraph, we will show that we can do it more efficiently using dynamic programming. For simplicity we write in the following $c_k(r, e^{-1}) = c_k(r)$ and $f_1(k, e^{-1}) = f_1(k)$. We define the new value $D(k, r, m)$ by

$$D(k, r, m) = \sum_{[i_1,\ldots,i_j] \in Par_{\geq m}(r)} c_{k-1}(i_1) \cdots c_{k-1}(i_j) f_2([i_1, \ldots, i_j])$$

where $Par_{\geq m}(r)$ is the set of all partitions of the integer r such that for each $[i_1, \ldots, i_j] \in Par_{\geq m}(r)$ must hold that $i_\ell \geq m$ for all $1 \leq \ell \leq j$. Using this, we can give the recursive definition of $D(k, r, m)$ and $c_k(r)$ as described in this theorem.

Proposition 1. *For fixed values R and K we can compute $c_k(r)$, as described in Theorem 3, for all $r \leq R$ and $k \leq K$ in a time complexity of $O\left(KR^2 \ln(R)\right)$.*

Proof. We use dynamic programming to compute $c_k(r)$.

The computation of $f_1(k)$ can be done once for all $k \leq K$ and then be stored. Thus, it has a time and space complexity of $O(K)$. For $k = 1$, if we start with $r = 1$ we can compute $c_1(r)$ for all $r \leq R$ in R steps. The same is true for $1 < k \leq K$ if we already know $D(k, r, 1)$ and $f_1(k)$.

The most time consuming factor is the computation of $D(k, r, m)$. For a given k', let us assume that we have already computed all $c_{k'-1}(r)$ for $1 \leq r \leq R$. In the computation of $D(k, r, m)$ we will go for r from 1 to R, and for m from r to 1. This means that

- For a given r' we already know all $D(k', r, m)$ with $r < r'$.
- For a fixed r' and m' we already know all $D(k', r', m)$ with $m > m'$.
- To compute
$$\sum_{u=0}^{\lfloor r/m \rfloor} \frac{c_{k-1}(m)^u}{u!} D(k, r - mu, m + 1)$$
we need $\lfloor r/m \rfloor$ steps.

Thus in total, for each $1 < k \leq K$ we need

$$\sum_{r=1}^{R} \sum_{m=1}^{r} \left\lfloor \frac{r}{m} \right\rfloor$$

steps to compute all $D(k, r, m)$. By using that

$$\sum_{m=1}^{r} \frac{1}{m} = ln(r) + C + O\left(\frac{1}{r}\right)$$

where $C = 0.5772\ldots$ is the Euler constant, and

$$\begin{aligned}\sum_{r=1}^{R} r \ln(r) &\leq \ln(R) \sum_{r=1}^{R} r \\ &= \ln(R) \frac{R(R+1)}{2}\end{aligned}$$

we obtain the final time complexity of $O(KR^2 \ln(R))$.

Let us go back to the expected entropy in (2). By using (6) we can write for $n \to \infty$

$$\mathbf{H_k} \sim \log_2(n) \underbrace{- \sum_{r=1}^{R} c_k(r)\ r\ \log_2(r)}_{(a)} \underbrace{- \sum_{r=R+1}^{n} c_k(r)\ r\ \log_2(r)}_{(b)} , \qquad (8)$$

where (a) represents an estimation of the entropy loss which does not depend on n and (b) is an error term. In Fig. 6, we see that the value $c_k(r)\ r\ \log_2(r)$

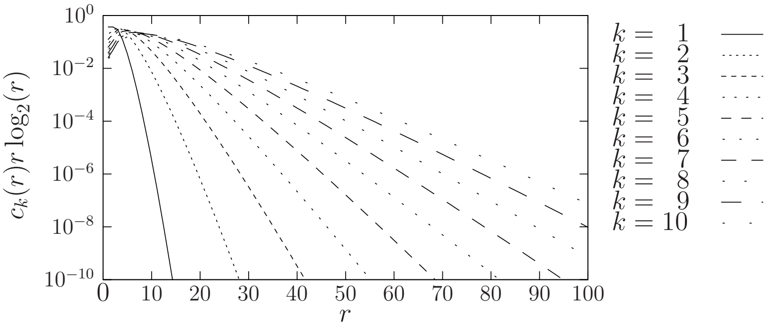

Fig. 6. The course of $c_k(r)\ r\ \log_2(r)$ for different values of k and r

decreases fast with growing r. However, for larger k this decrease becomes slower. If we want (b) to be negligible for larger k we also need a larger value for R. In Table 1, we compare our entropy estimator

$$H_k(R) = \log_2(n) - \sum_{r=1}^{R} c_k(r)\ r\ \log_2(r) \qquad (9)$$

with the estimated lower bound of the loss given by the expected number of image points (1) and the empirical results from the experiment presented in Fig. 3. From (6) and (8) we know that

$$\mathbf{H_k} \sim H_k(R)$$

for $n \to \infty$ and $R \to n$. We can see that for small k, in the order of a few hundred, we reach a much better approximation than the upper bound (1). For example, for most of the modern stream ciphers, the number of iterations for a key/IV–setup is in this order of magnitude. However, for increasing values of k we also need bigger values of R and, thus, this method gets computationally expensive. For $k = 100$ and $R = 1000$ the result of our estimate is about 0.02 larger than the empirical data. The fact that our estimate is larger shows that it is not due to the choice of R (it does not change a lot if we take $R = 2000$)

Table 1. Comparison of different methods to estimate the entropy loss

k		1	2	3	$\cdots$	10	$\cdots$	50	$\cdots$	100
empirical data, $n = 2^{16}$		0.8273	1.3458	1.7254	$\cdots$	3.1130	$\cdots$	5.2937	$\cdots$	6.2529
image points (1)		0.6617	1.0938	1.4186	$\cdots$	2.6599	$\cdots$	4.7312	$\cdots$	5.6913
$H_k(R)$**, (9)**	$R = 50$	0.8272	1.3457	1.7254	$\cdots$	3.1084	$\cdots$	2.6894	$\cdots$	1.2524
	$R = 200$	0.8272	1.3457	1.7254	$\cdots$	3.1129	$\cdots$	5.2661	$\cdots$	5.5172
	$R = 1000$	0.8272	1.3457	1.7254	$\cdots$	3.1129	$\cdots$	5.2918	$\cdots$	6.2729

but to the fact that our $n = 2^{16}$ is relatively small and, thus, the proportion of cycle points which is about

$$\frac{\sqrt{\pi n/2}}{n(1-\tau_k)} \approx 0.253$$

is not negligible.

In this section we presented a new entropy estimator. We could show that if the number of iterations is not too big, it is much more precise than the upper bound given by the image size. In addition, the same method can be used for any arbitrary initial distribution.

3 Collision Attacks

In the previous section, we studied the loss of entropy in the inner state of our stream cipher model. In this section, we examine if it is possible to exploit this loss for a generic attack on our model. Hong and Kim present in [HK05] two attacks on the Mickey stream cipher [BD05], based on the entropy loss in the state. This stream cipher has a fixed update function; however Hong and Kim state, due to empirical results, that the update function behaves almost like a random function with regard to the expected entropy loss and the expected number of image points. Thus, these attacks are directly applicable on our model. We will give a detailed complexity analysis of these attacks and will show that in the case of a real random function they are less efficient than what one might assume from the argumentation of Hong and Kim.

Let us take two different initial states S_0 and S'_0 and apply the same function iteratively onto both of them. We speak about a collision if there exists k and k' such that $S_k = S_{k'}$, for $k \neq k'$, or $S_k = S'_{k'}$ for any arbitrary pair k, k'. The idea of Hong and Kim was that a reduced entropy leads to an increased probability of a collision. Once we have found a collision, we know that the subsequent output streams are identical. Due to the birthday paradox, we assume that with an entropy of m-bits we reach a collision, with high probability, by choosing $2^{\frac{m}{2}}$ different states.

The principle of the attacks is that we start from m different, randomly chosen, initial states and that we apply iteratively the same update function k times on each of them. In the end, we search for a collision and hope that our costs are less than for a search directly in the initial states. We will study the two attacks proposed in [HK05] as well as a variant using distinguished points. For each of these attacks we provide a detailed complexity analysis where we examine the query and the space complexity as well as the number of necessary initial states to achieve a successful attack with high probability. By the query complexity we mean the number of all states produced by the cipher during the attack which is equivalent to the number of times the updated function is applied. By the space complexity we mean the number of states we have to store such that we can search for a collision within them. Each time we compare the results to the attempt of finding a collision directly within the initial states which has a space and query complexity of $\sim \sqrt{n}$.

All these attacks consider only the probability of finding a collision in a set of states. This is not equivalent to an attack where we have $m-1$ initial states prepared and we want the probability that if we take a new initial state, it will be one of the already stored. In such a scenario, the birthday paradox does not apply. We also never consider how many output bits we would really need to store and to recognize a collision, since this value depends on the specific filter function used. In the example of Mickey, Hong and Kim states that they need about 2^8 bits.

3.1 States After k Iterations

The first attack of Hong and Kim takes randomly m different initial states, applies k times the same instance of Φ on each of them, and searches a collision in the m resulting states. Using (1) we know that the average entropy after k iterations is less than $\log_2(n) + \log_2(1-\tau_k)$. Hong and Kim conjecture, based on experimental results, that this is about the same as $\log_2(n) - \log_2(k) + 1$. Thus, with high probability we find a collision if $m > 2^{(\log_2(n)-\log_2(k)+1)/2} = \sqrt{2n/k}$.

This attack stores only the last value of the iterations and searches for a collision within this set. This leads to a space complexity of $m \sim \sqrt{2n/k}$ for large enough k. However, Hong and Kim did not mention that we have to apply k times Φ on each of the chosen initial states, which results in a query complexity of $mk \sim \sqrt{2kn}$. This means that we increase the query complexity by the same factor as we decrease the space complexity and the number of initial states.

The question remains, if there exists any circumstances under which we can use this approach without increasing the query complexity. The answer is yes, if the stream cipher uses a set of update functions which loose more than $2\log_2(k)$ bits of entropy after k iterations. Such a characteristic would imply that we do not use random functions to update our state, since they have different properties as we have seen before. However, the principle of the attack stays the same.

3.2 Including Intermediate States

The second attack in [HK05] is equivalent to applying $2k-1$ times the same instance of Φ on m different initial states and searching for a collision in all intermediate states from the k-th up to the $(2k-1)$-th iteration. Since after

$k-1$ iterations we have about $\log_2(n) - \log_2(k) + 1$ bits of entropy, Hong and Kim assume that we need a m such that $mk \sim \sqrt{n/k}$. They state that this result would be a bit too optimistic since collisions within a row normally do not appear in a practical stream cipher. However, they claim that this approach still represents a realistic threat for the MICKEY stream cipher. We will show that for a random function, contrary to their conjecture, this attack has about the same complexities as a direct collision search in the initial states.

Let us take all the $2km$ intermediate states for the $2k-1$ iterations. Let $Pr[A]$ define the probability that there is no collision in all the $2km$ intermediate states. If there is no collision in this set then there is also no collision in the km states considered by the attack. Thus, the probability of a successful attack is even smaller than $1 - Pr[A]$. By means of only counting arguments we can show the following proposition.

Proposition 2. *The probability of no collision in the $2km$ intermediate states is*

$$Pr[A] = \frac{n(n-1)\cdots(n-2k+1)}{n^{2km}} \ , \tag{10}$$

where the probability is taken over all functions $\varphi \in \mathcal{F}_n$ and all possible choices of m initial states.

Proof. Let $Pr[I]$ be the probability of no collision in the m initial states. We can see directly that

$$\begin{aligned} Pr[A] &= Pr[A \cap I] \\ &= Pr[A|I]\ Pr[I] \ . \end{aligned}$$

Let us assume that we have chosen m different initial states. This happens with a probability of

$$Pr[I] = \frac{\binom{n}{m} m!}{n^m} \ . \tag{11}$$

In this case we have

- n^n different instances $\varphi \in \mathcal{F}_n$ of our random functions, where each of them creates
- $\binom{n}{m} m!$ different tables. Each table can be produced more than once. There exists
- $n\ (n-1)\dots(n-2km+1)$ different tables that contain no collisions. Each of them can be generated by
- $n^{n-(2k-1)m}$ different functions, since a table determines already $(2k-1)m$ positions of φ.

Thus, we get the probability

$$Pr[A|I] = \frac{n\ (n-1)\dots(n-2km+1)\ n^{n-(2k-1)m}}{n^n \binom{n}{m} m!} \tag{12}$$

for $m > 0$ and $2km \leq n$. By combining (11) and (12) we can conclude our proof.

The probability of $Pr[A]$ given in (10) is exactly the probability of no collision in $2km$ random points, which means that we need at least an m such that $2km \sim \sqrt{n}$. This leads to a query complexity of $\sim \sqrt{n}$ and a space complexity of $\sim \sqrt{n}/2$.

3.3 Improvement with Distinguished Points

By applying the known technique of distinguished points [DQ88] we can reduce the space complexity in the second attack; however the query complexity stays the same.

By *distinguished points* (DPs) we mean a subset of Ω_n which is distinguished by a certain property, e.g. by a specific number of 0's in the most significant bits. In our new variant of the second attack we iterate Φ in each row up to the moment where we reach a DP. In this case we stop and store the DP. If we do not reach a DP after k_{MAX} iterations we stop as well but we store nothing. If there was a collision in any of the states in the rows where we reached a DP, the subsequent states would be the same and we would stop with the same DP. Thus it is sufficient to search for a collision in the final DPs.

Let d be the number of distinguished points in Ω_n. We assume that the ratio $c = \frac{d}{n}$ is large enough that with a very high probability we reach a DP before the end of the cycle in the functional graph. This means that the average number of iterations before arriving at a DP is much smaller than the expected length of the tail and the cycle together (which would be about $\sqrt{\frac{\pi\, n}{2}}$ due to [FO90a]). We assume that in this case the average length of each row would be in the range of $1/c$ like in the case of random points. We also suppose that that we need about $m/c \sim \sqrt{n}$ query points to find a collision, like in the previous case. This leads to a query complexity of $\sim \sqrt{n}$ and a space complexity of only $\sim c\sqrt{n}$. Empirical results for example for $n = 2^{20}$, $0.7 \leq \frac{\log_2(d)}{\log_2(n)} \leq 1$ and $k_{MAX} = \sqrt{n}$ confirm our assumptions.

A summary of the complexities of all attacks can be found in Table 2, where we marked by *(new)* the results that where not yet mentioned by Hong and Kim. In the case where we consider only the states after k iterations, we have to substantially increase the query complexity to gain in the space complexity and the number of initial states. We were able to show that even when we consider all intermediate states, the query complexity has a magnitude of $\sqrt{n}$. The variant using the distinguished points allows to reduce the space complexity by leaving the other complexities constant.

Table 2. Complexities of attacks

attack	**# initial states**	**space complexity**	**query complexity**
after k iterations, 3.1	$\sim \sqrt{2n/k}$	$\sim \sqrt{2n/k}$	$\sim \sqrt{2kn}$ *(new)*
with interm. states, 3.2	$\sim \sqrt{n}/2k$ *(new)*	$\sim \sqrt{n}/2$ *(new)*	$\sim \sqrt{n}$ *(new)*
with DPs, 3.3	$\sim c\sqrt{n}$ *(new)*	$\sim c\sqrt{n}$ *(new)*	$\sim \sqrt{n}$ *(new)*

4 Conclusion

In this article, we studied a stream cipher model which uses a random update function. We have introduced a new method of estimating the state entropy in this model. This estimator is based on the number of values that produce the same value after k iterations. Its computation is expensive for large numbers of iterations; however, for a value of k up to a few hundred, it is much more precise than the upper bound given by the number of image points.

In this model, we have also examined the two collision attacks proposed in [HK05] which are based on the entropy loss in the state. We pointed out that the first attack improves the space complexity at the cost of significantly increasing the query complexity. We proved that the complexity of the second attack is of the same magnitude as a collision search directly in the starting values. In addition we discussed a new variant of this attack, using distinguished points, which reduces the space complexity but leaves the query complexity constant.

The use of a random function in a stream cipher introduces the problem of entropy loss. However, the studied attacks based on this weakness are less effective than expected. Thus, the argument alone that a stream cipher uses a random function is not enough to threaten it due to a collision attack based on the entropy loss.

References

[BD05] Babbage, S., Dodd, M.: The stream cipher MICKEY (version 1). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/015 (2005), `http://www.ecrypt.eu.org/stream`

[DQ88] Delescaille, J.-P., Quisquater, J.-J.: Other cycling tests for DES (abstract). In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 255–256. Springer, Heidelberg (1988)

[FO90a] Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Advances in Cryptology, Proc. Eurocrypt 1998, vol. 434, pp. 329–354 (1990)

[FO90b] Flajolet, P., Odlyzko, A.M.: Singularity analysis of generating functions. SIAM J. Discrete Math. 3(2), 216–240 (1990)

[FS96] Flajolet, P., Sedgewick, R.: An introduction to the analysis of algorithms. Addison-Wesley Longman Publishing Co., Inc. Boston (1996)

[Hel80] Hellman, M.: A cryptanalytic time-memory trade-off. Information Theory, IEEE Transactions on 26(4), 401–406 (1980)

[HK05] Hong, J., Kim, W.H.: TMD-tradeoff and state entropy loss considerations of streamcipher MICKEY. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 169–182. Springer, Heidelberg (2005)

[HS05] Hong, J., Sarkar, P.: New applications of time memory data tradeoffs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005)

[Kol86] Kolchin, V.F.: Random Mappings. Optimization Software, Inc. (1986)

[SS03] Seznec, A., Sendrier, N.: HAVEGE: A user-level software heuristic for generating empirically strong random numbers. ACM Trans. Model. Comput. Simul. 13(4), 334–346 (2003)