Security Evaluation of Stream Cipher Enocoro-128v2

Martin Hell and Thomas Johansson

Abstract. This report presents a security evaluation of the Enocoro-128v2 stream cipher. Enocoro-128v2 was proposed in 2010 and is a member of the Enocoro family of stream ciphers. This evaluation examines several different attacks applied to the Enocoro-128v2 design. No attack better than exhaustive key search has been found.

1 Introduction

This report contains a security evaluation of the stream cipher Enocoro-128v2 as specified in the work order specification from CRYPTREC dated Sept 15, 2010. Security evaluations of previous versions of Enocoro, as well as the current Enocoro-128v2 version has previously been conducted [17, 24–28, 30]. As these results are in Japanese, these have not been available to us during our evaluation work.

We start by giving a short description of the algorithm in Section 2. Then, the following sections provide a summary of our findings. After some input during the work, the different attacks have been addressed as follows.

- In Section 3 we examine the keystream produced by Enocoro-128v2 using standard statistical tests. The NIST test suite is used.
- In Section 4 we examine generic Time-Memory-Data tradeoff attacks. Complexities for recovering the internal state, as well as complexities for recovering the secret key are given.
- In Section 5 we examine chosen IV attacks of differential type and also its statistical nature.
- In Section 6 we examine the more recent chosen IV attack denoted the maximum degree monomial test.
- In Section 7 we examine cube attacks.
- In Section 8 we examine some properties around linear distinguishing attacks and give a biased relation in the keystream bits.
- In Section 9 we examine guess-and-determine type of attacks.
- In Section 10 we examine at algebraic attacks on the generator.

We end the report by some giving some general conclusions in Section 11. Moreover, in Sections 3-10 we also give ending conclusions for each attack.

2 Description of Enocoro-128v2

This section gives a short introduction to Enocoro, a pseudorandom number generator (PRNG) for use in a stream cipher, related to the PANAMA construction [8]. Enocoro is actually a family of generators, where a number of parameters can give different instantiations. A first specification of Enocoro was published in [35], which is referred to as Enocoro v1. Two versions, one for 80-bit security and one for 128-bit security, was recommended, referred to as Enocoro-80v1 and Enocoro-128v1. Later, a new version for 128-bit security appeared in [3] and it is referred to as Enocoro-128v1.1.

This document considers the most recent version which is again a family of generators referred to as Enocoro v2. The evaluation concerns an instantiation for 128-bit security which is called Enocoro-128v2. It was proposed in [36] and [37]. It can be noted that Enocoro-128v2 and Enocoro-128v1.1 differ only in the choice of the characteristic polynomial over $\mathbb{F}_2^8$ and in the way initialization is done.

The Enocoro v2 generator is a finite state machine, i.e., it contains an internal state and two functions, one function updating the current state to a new state and a second function producing an output from the current state. These operations occur once every time instance. Before the output is produced, a special initialization function is applied, that from a given key and IV value places the finite state machine in a particular state. This initialization function is in general a very important part of the generator as the cryptanalyst can obtain several different keystreams from using several different IV values under the same key.

The Enocoro-128v2 generator is explicitly described as follows. The generator is byte-oriented and most variables/values are considered as bytes. The internal memory consists of 32 bytes, together forming what we refer to as the *buffer*,

$$\mathbf{b} = (b_0, b_1, \ldots, b_{30}, b_{31}),$$

and two more bytes, here called the *state*,

$$\mathbf{a} = (a_0, a_1).$$

The buffer and state will have a value for every time instance, so we can introduce the notation $a_i(t)$ and $b_i(t)$ as the value of a_i and b_i at time t, respectively.

Next, we need to specify the update function and the output function. The output function is very simple. At each time t the generator outputs one byte z_t by

$$z_t = a_1(t).$$

The update function is more complicated. The buffer $\mathbf{b}$ has a simple update as

$$b_i(t+1) = b_{i-1}(t), \forall i, i \neq \{0, 3, 8, 17\},$$

and

$$\begin{aligned} b_0(t+1) &= b_{31}(t) + a_0(t), \\ b_3(t+1) &= b_2(t) + b_6(t), \\ b_8(t+1) &= b_7(t) + b_{15}(t), \\ b_{17}(t+1) &= b_{16}(t) + b_{28}(t), \end{aligned}$$

where + means bitwise XOR. But the update of the state **a** is more advanced. From $(a_0(t), a_1(t))$, we first compute the intermediate values

$$\begin{aligned} u_0(t) &= a_0(t) + S[b_2(t)], \\ u_1(t) &= a_1(t) + S[b_7(t)]. \end{aligned}$$

Then compute intermediate values

$$\begin{pmatrix} v_0(t) \\ v_1(t) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & d \end{pmatrix} \begin{pmatrix} u_0(t) \\ u_1(t) \end{pmatrix}.$$

Finally, the next state is given as

$$\begin{aligned} a_0(t+1) &= v_0(t) + S[b_{16}(t)], \\ a_1(t+1) &= v_1(t) + S[b_{29}(t)]. \end{aligned}$$

Here $S[x]$ is a byte-oriented S-box (8 bits to 8 bits), providing the nonlinearity in the construction. The S-box $S[x]$ is built from using a 4 bit to 4 bit S-box, called S_4, several times. For the details on how $S[x]$ is constructed from S_4, we refer to the design document. Also, the linear transformation given by the matrix above is denoted by L, or $(v_0(t), v_1(t))^T = L(u_0(t), u_1(t))$ It is a linear transformation over $\mathbb{F}_2^8$ where bytes are associated with values in $\mathbb{F}_2^8$ through a specific generating polynomial, also given in the design document. The value d in the matrix is a specific element in $\mathbb{F}_2^8$. An overview of the keystream generation part, as described, is given in Figure 1.

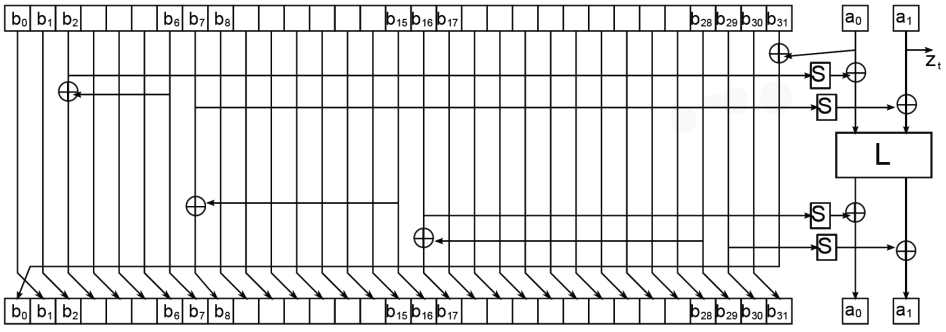

Fig. 1. Keystream generation in Enocoro-128v2.

Finally, we need to shortly introduce the initialization function. Given a 128 bit (16 byte) key $\mathbf{k}$, written in bytes as

$$\mathbf{k} = (k_0, k_1, \ldots k_{15}),$$

and a 64 bit (8 byte) IV value (public value) $\mathbf{i}$, written in bytes as

$$\mathbf{i} = (i_0, i_1, \ldots, i_7),$$

the initialization function should give us the starting value of the buffer $\mathbf{b}$ and the state $\mathbf{a}$. This is done by setting $b_0, b_1, \ldots, b_{15}$ to the $\mathbf{k} = (k_0, k_1, \ldots k_{15})$ value, setting $b_{16}, b_{17}, \ldots, b_{23}$ to the $\mathbf{i} = (i_0, i_1, \ldots, i_7)$ value and finally setting $b_{24}, b_{25}, \ldots, b_{31}$ to some predetermined value (given in the specification document). The state $\mathbf{a}$ is set to zero. Then the state update function, as described in the keystream generation part, is applied 96 times, but with a small exception. The exception is in the update of b_0, which is now given by

$$b_0(t+1) = b_{31}(t) + a_0(t) + ctr(t).$$

The new part, $ctr(t)$ is a known counter value that is added, a different value in different time instances in the initialization. The reason is to protect against some related key attacks on the previous version, using sliding techniques [37].

We have very shortly described the Enocoro-128v2 PRNG, taking a secret 128 bit key, a public 64 bit IV value and for each such pair it produces a keystream sequence of bytes. If we want to use it for encryption, we typically add bitwise the keystream to the plaintext, getting the ciphertext.

3 Statistical Tests on the Keystream

Statistical tests checking the randomness of the keystream has been performed. The test suite used was the one provided by NIST [29]. This is a set of 15 statistical tests that can be applied to a bit sequence. In the context of pseudo random generators for cryptographic purposes, it is very important to understand the applicability of these types of tests. If the generator fails one on more tests, this gives a clear indication that the keystream is not random and susceptible to a distinguishing attack (and perhaps also a key recovery attack). If, on the other hand, the generator passes all tests, this does not say much at all about the security of the generator. The reason is that the tests do not take actual design into account.

In the tests, 200 keystream sequences generated by a random key/IV pair have been tested. Each sequence is $1,000,000$ bits long. The results of the tests are given in Table 1. The P-value can be interpreted as the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested. Each row in Table 1 gives the name of the test, the number of tests that were passed out of the 200, the P-value, and the distribution of the 200 P-values for the individual tests. As can be seen, Enocoro-128v2 did not pass all 200 tests in any of the cases. However, as $\alpha = 0.01$, this is as expected. For an in-depth description of the tests we refer to [29].

Table 1. Results from running the NIST statistical test suite on Enocoro-128v2 keystreams. Note that some tests are run with different parameters. Only one result for each test has been included in the table. Also note that the RandomExcursions and RandomExcursionsVariant tests have been aborted in some cases (the number of zero crossings are too few to give a good test.)

Statistical test	Proportion	P-value	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Frequency	198/200	0.196920	23	12	18	24	17	19	17	17	22	31
BlockFrequency	198/200	0.678686	20	18	23	26	20	19	25	13	18	18
CumulativeSums	198/200	0.428095	23	18	17	23	14	13	20	21	24	27
Runs	199/200	0.064822	24	18	31	18	14	15	23	27	17	13
LongestRun	197/200	0.689019	19	25	19	20	21	24	11	20	22	19
Rank	195/200	0.096578	32	17	18	28	17	14	17	22	16	19
FFT	197/200	0.080519	17	31	18	18	16	24	19	26	10	21
NonOverlappingTemplate	197/200	0.419021	20	18	29	15	24	21	18	20	13	22
OverlappingTemplate	199/200	0.202268	31	23	21	14	15	21	21	23	16	15
Universal	196/200	0.334538	18	20	18	24	24	17	21	29	12	17
ApproximateEntropy	199/200	0.534146	22	29	24	20	17	16	19	16	21	16
RandomExcursions	128/129	0.973388	12	13	14	16	11	13	11	15	12	12
RandomExcursionsVariant	128/129	0.182384	11	8	12	12	10	21	12	12	12	19
Serial	194/200	0.162606	22	28	16	25	22	20	23	19	16	9
LinearComplexity	196/200	0.437274	18	19	30	20	22	19	14	23	20	15

3.1 Conclusion

We conclude that Enocoro-128v2 keystreams pass all statistical tests in the NIST test suite, but also emphasize that this does not say much about the security of the stream cipher.

4 Time-Memory-Data Tradeoff Attacks

In this section we consider the generic time-memory-data tradeoff attacks on Enocoro-128v2. Two main types of tradeoffs for stream ciphers will be considered, namely the Babbage-Golić tradeoff and the Biryukov-Shamir tradeoff. Throughout the section N denotes the search space, T is the time complexity in the online phase of the attack, D is the amount of data needed in the attack, P is the time complexity in the offline (or precomputation) phase and M is the amount of memory (storage) needed in the attack. We consider both attacks on the internal state and on the secret key. In the general discussion, we denote the size of the key and IV by k and iv respectively.

4.1 Attack on Internal State

We first consider attacks on the internal state of Enocoro-128v2. The one-way function f considered in these attacks is the mapping from the $\log N$-bit internal

state s to a $\log N$-bit keystream,

$$f : \{0,1\}^{\log N} \rightarrow \{0,1\}^{\log N}.$$

It is easy to generate keystream when the internal state is known, but recovering the state from known keystream is infeasible for any secure stream cipher.

Babbage-Golić. Babbage [3] and Golić [18] independently described a simple but powerful time-memory-data tradeoff attack on the internal state of a stream cipher. In the precomputation phase the attacker chooses M random states and stores the corresponding keystream for each state, i.e., the pair $(s, f(s))$. By observing D $\log N$-bit keystream blocks from the stream cipher, the table is searched for the corresponding $f(s)$. The table covers a fraction M/N of the total search space and the number of keystream blocks needed in order to find a match is geometrically distributed with an expected value of N/M. Thus we get the tradeoff curve $N = M \cdot D$ with $T = D$. (This is often referred to as the birthday paradox.) The precomputation time is $P = M$. One typical point on the curve is $T = M = D = N^{1/2}$, which implies that the size of the state $\log N$ should be at least twice the key size (or security claim).

Biryukov-Shamir. Another variant of this attack was described by Biryukov and Shamir [4]. The idea was similar to the original attack by Hellman [20]. Chains using two functions f, as given above, and h, called reduction function, are computed. The function f maps an input (state) to an output (keystream block) and the function h maps the output to a new input. These chains are t long and one table consists of m chains. To avoid merging chains no more rows are computed when $N = mt^2$. Instead t different tables are built, each using a different reduction function h. The online time is $T = t^2$ and the memory required is $M = mt$. This gives the tradeoff curve $N^2 = M^2T$.

The Biryukov-Shamir approach takes advantage of the fact that the attacker has access to many keystream blocks. Thus, instead of covering the complete search space N by the tables as in the case with Hellman's attack, only a fraction N/D is covered by the tables and the attack is repeated D times, one time for each observed $\log N$-bit keystream block. Instead of building t tables as in the Hellman case (one data point), $t/D(\geq 1)$ tables are built instead. Searching one data point costs time $t \cdot t/D$. Searching all D data points costs $T = t^2$. Since only start and endpoints are saved the amount of memory needed is $M = m \cdot t/D$. This gives the tradeoff curve $N^2 = D^2M^2T$, with the restriction $D^2 \leq T$ (since $D \leq t$). The precomputation time is $P = N/D$.

One point on this curve is $T = M = N^{1/2}$, $D = N^{1/4}$ and $P = N^{3/4}$. The time T and memory M is the same as that in the Babbage-Golić attack, but the data complexity D is reduced while the precomputation time P is increased. Another point on the curve is $P = T = N^{2/3}$ and $D = M = M^{1/3}$. Compared to the Babbage-Golić attack, this tradeoff has smaller data and memory complexity, but an increased precomputation and online time complexity.

Since Enocoro-128v2 has a 272-bit internal state, and a 128-bit key, we conclude that Enocoro-128v2 resists time-memory-data tradeoff attacks on the internal state.

Table 2. Attack complexities for a time-memory-data tradeoff attack on the internal state. Complexities for the Babbage-Golić (BG) attack and two choices of parameters for the Biryukov-Shamir attack (BS1 and BS2) are given. Other tradeoffs are possible.

Time-Memory-Tradeoff Attacks on the Internal State						
	General			Enocoro-128v2 ($N = 272$)		
	BG	BS1	BS2	BG	BS1	BS2
P	$N^{1/2}$	$N^{3/4}$	$N^{2/3}$	2^{136}	2^{204}	2^{181}
T	$N^{1/2}$	$N^{1/2}$	$N^{2/3}$	2^{136}	2^{136}	2^{181}
M	$N^{1/2}$	$N^{1/2}$	$N^{1/3}$	2^{136}	2^{136}	2^{91}
D	$N^{1/2}$	$N^{1/4}$	$N^{1/3}$	2^{136}	2^{68}	2^{91}

4.2 Attack on the Initialization Function

The attacks given in the previous section were applied to the initialization function in [21,22] by Hong and Sarkar. The one-way function f considered in these attacks is the mapping from the $(k + iv)$-bit key and initialization vector to the first $k + iv$ bits of the keystream,

$$f : \{0,1\}^{k+iv} \to \{0,1\}^{k+iv}.$$

In this case $N = 2^{k+iv}$. In the case of Enocoro-128v2, the key is 128 bits and the IV is 64 bits.

Babbage-Golić. In the Babbage-Golić attack, D different key/IV pairs are used to initialize the cipher and the first $k + iv$ bits of the keystream are stored in a table together with the key/IV. The table is sorted according to the keystream. By observing $2^{k+iv}/D$ keystream blocks from 2^{k+iv} key/IV pairs, one of these pairs is expected to be found in the table. As an example, it is possible to mount a time-memory-data tradeoff attack of this kind with 2^{96} memory words and observing keystream from 2^{96} key/IV pairs.

Biryukov-Shamir. In the Biryukov-Shamir attack, a reduction function is used to map a keystream block to a new key/IV pair. A new reduction function is used for each table. Tables covering in total N/D key/IV pairs are computed

and, for each data point, the tables are searched. As an example, with precomputation time 2^{144}, memory 2^{96}, online time 2^{96} and 192-bit keystream blocks corresponding to 2^{48} key/IV pairs, it is expected to recover one of these key/IV pairs.

Some typical attack complexities for these attacks are summarized in Table 3.

Table 3. Attack complexities for a time-memory-data tradeoff attack on the initialization function when 0%, 50% and 100% of the IV can be controlled by the attacker. Complexities for the Babbage-Golić (BG) attack and two choices of parameters for the Biryukov-Shamir attack (BS1 and BS2) are given. Other tradeoffs are possible.

Time-Memory-Tradeoff Attacks on the Initialization Function												
	General			Enocoro-128v2 ($N = 192$)								
				No IV Control			50% IV Control			Full IV Control		
	BG	BS1	BS2	BG	BS1	BS2	BG	BS1	BS2	BG	BS1	BS2
P	$N^{1/2}$	$N^{3/4}$	$N^{2/3}$	2^{96}	2^{144}	2^{128}	2^{80}	2^{120}	2^{107}	2^{64}	2^{96}	2^{85}
T	$N^{1/2}$	$N^{1/2}$	$N^{2/3}$	2^{96}	2^{96}	2^{128}	2^{80}	2^{80}	2^{107}	2^{64}	2^{64}	2^{85}
M	$N^{1/2}$	$N^{1/2}$	$N^{1/3}$	2^{96}	2^{96}	2^{64}	2^{80}	2^{80}	2^{53}	2^{64}	2^{64}	2^{43}
D	$N^{1/2}$	$N^{1/4}$	$N^{1/3}$	2^{96}	2^{48}	2^{64}	2^{80}	2^{40}	2^{53}	2^{64}	2^{32}	2^{43}

4.3 Other Notes On the Time-Memory-Data Tradeoff Attacks

In a chosen IV scenario, the IV can be fixed when building the tables for these attacks. In that case, $N = 2^k$. Even if the IV can not be completely chosen, it might be the case that some part of the IV is fixed and/or known to the attacker beforehand. This will significantly improve the attack numbers above. As a comparison, in Table 3, we also give the corresponding figures for P, M, T and D when both half the IV and the full IV can be chosen or predicted by the attacker. The applicability of these figures is of course dependent on the usage scenario and how IVs are chosen.

Note that there are important fundamental differences between the attacks on the internal state and the attacks on the initialization function. The attack on the internal state can be used to break a stream cipher that uses one particular key and IV. It is keystream from this key and IV that is used in the attack. If the state update function and the initialization function are invertible, then this will also recover the key and IV. The attack on the initialization function recovers the key and IV for the particular keystream that is used. Only one keystream

is used for each key/IV pair. Thus, this attack requires keystreams from many different key/IV pairs and only recovers one such pair.

The fact that the IV is assumed to be publicly known was taken advantage of by Dunkelman and Keller in [11]. They create separate tables for different selected IVs. When a keystream with a selected IV is observed, only the corresponding table(s) are searched. This gives the same tradeoff as in the Biryukov-Shamir attack, but does not have the restriction $D^2 \leq T$.

4.4 Conclusions

The internal state of Enocoro-128v2 is 272 bits. Thus, there is no time-memory-data tradeoff attack on the internal state that has time, memory and data less that 2^{128}. An exhaustive key search will always be more efficient than an online phase in a TMDTO attack.

The relatively short IV (64 bits) makes Enocoro-128v2 more susceptible to time-memory-data tradeoff attacks on the initialization function. There are several possibilities to mount these attacks with time, memory and data less that 2^{128}. Still, the precomputation time is equal to or larger than exhaustive key search if data and/or memory is to be decreased. Additionally, only one out of several key/IV pairs can be recovered. If an attacker can choose the IV, the complexities for recovering one out of 2^{64} keys are rather moderate. This has nothing to do with the IV length, but is a result of the 128-bit key.

5 Chosen IV Attacks

By chosen IV cryptanalysis we mean attacks that targets the initialization of the cipher. These attacks typically use a number of keystream sequences generated from the same key but using initialization with different IV values [9]. We can also consider a number of keystream sequences generated from the different but related keys and possibly using initialization with different IV values. This gives related key attacks. In this section, let $i_0, i_1, \ldots, i_{m-1}$ be the IV bits, $k_0, k_1, \ldots, k_{n-1}$ be the key bits and let z_i be the ith keystream bit. Note that we consider bits instead of bytes in the notation in this section! Then we can write z_i as a function of key and IV bits,

$$z_i = f_i(i_0, i_1, \ldots, i_{m-1}, k_0, k_1, \ldots, k_{n-1}) \qquad i_j \in \mathcal{I} \qquad k_j \in \mathcal{K}.$$

The basic assumption is the knowledge of $\mathbf{z} = z_0, z_1, \ldots$ for a number of different IV values $(i_0, i_1, \ldots, i_{m-1}) \in I$, for some set I.

We consider a few possible attacks under this assumption. One is a differential attack to be described next. Another one is pure statistical testing, described in Subsection 5.2. A final one is another statistical testing tool based on the max degree test to be described in Section 6.

5.1 Differential Chosen IV Attacks

A differential attack on the initialization treats Enocoro-128v2 as a 96 round function and studies how differences in the input, in this case IV bits and possibly key bits, propagates through the 96 rounds and eventually appears in the keystream bits.

Recalling the initialization of Enocoro-128v2, we see that key bits set the state byte b_0 up to b_{15} and IV bits set the state byte b_{16} up to b_{23}. Let us introduce the notation

$$Y(t) = (b_0(t), b_1(t), \ldots, b_{31}(t)|a_0(t), a_1(t))$$

for the full state after t round (clockings). Furthermore let us consider the difference between the generation of any two sequences. We have a sequence of state values $Y(t)$ when generating the first keystream and another sequence of state values $Y'(t)$ when generating the second keystream. Furthermore, the XOR difference between them is denoted $\delta Y(t)$, i.e.,

$$\delta Y(t) = Y(t) + Y'(t).$$

A promising differential attack could be if we start with a single differential in the second IV byte, i.e., in the b_{17} state variable. Then we will have a propagation of the differential as

$$\delta Y(0) = (0000000000000000a00000000000000|00),$$

going to

$$\delta Y(12) = (000000000000000000000000000a000|00),$$
$$\delta Y(13) = (000000000000000a000000000000a00|00).$$

Next, the difference in b_{29} goes through an S-box to a_1 as well as shifting one step. The difference after the S-box is now a new nonzero value b. We reach

$$\delta Y(14) = (0000000000000000a000000000000a0|0b).$$

Next, the difference in a_0, a_1, $\delta(a_0, a_1) = (0, b)$ is transformed to $\delta(a_0, a_1) = (b, c = d \cdot b)$,

$$\delta Y(15) = (00000000000000000a000000000000a|bc).$$

This approach would lead to a successful attack if we could cancel out the differences in b_{31}, a_0, a_1 in the next step. However, since the update of a_0, a_1 is linear, it is not possible to reach $\delta(a_0, a_1) = (0, 0)$ in the next step, and the probability to reach

$$\delta Y(16) = (00000000000000000a000000000000|00)$$

is zero. This shows that the linearity in the (a_0, a_1) update is crusial. If the update would be nonlinear there would likely be some positive probability for reaching $Y(16)$ as above. The procedure could be repeated to reach a large number of rounds, perhaps all 96.

Note that this kind of analysis is strongly related to the search for collisions for hash functions. We have tried to examine more complicated differential patterns but not been able to find any promising such patterns.

5.2 Statistical Testing in Chosen IV Attacks

Basic statistical testing on the keystream sequence is a very simple and usually not very succesful approach, if the initialization is designed carefully.

Again, we generate a few keystream bits $\mathbf{z} = z_0, z_1, \ldots$ for a number of different IV values $(\mathbf{i}_0, \mathbf{i}_1, \ldots, \mathbf{i}_{m-1}) \in I$, for some set I. Let us enumerate the n bit strings of keystream generated from different IVs as

$$\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1}.$$

This can be considered as m samples drawn from a distribution D on bit strings of length n. If we can find a statistical test that with a large probability shows that D is not the uniform distribution, we have a distinguisher.

We run several simulations generating m different output strings of length n for a fixed key and then apply the set of statistical tests provided by NIST [29] on the output.

Table 4 gives a summary of the results when applying one such test. In this test we generated 1 byte ($n = 8$) from $1,000,000$ initializations with fixed key and random IV. This was done for 200 random keys. The table should be interpreted similar to Table 1.

Table 4. Results from running the NIST statistical test suite on initializations with Enocoro-128v2. Note that some tests are run with different parameters. Only one result for each test has been included in the table. Also note that the RandomExcursions and RandomExcursionsVariant tests have been aborted in some cases (the number of zero crossings are too few to give a good test.)

Statistical test	Proportion	P-value	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Frequency	197/200	0.118812	21	29	26	12	12	19	16	23	23	19
BlockFrequency	196/200	0.296834	19	15	13	27	23	23	14	18	24	24
CumulativeSums	195/200	0.036352	24	26	24	16	11	16	18	15	18	32
Runs	198/200	0.118812	15	22	15	19	25	30	18	21	24	11
LongestRun	199/200	0.242986	16	22	26	26	14	26	20	21	16	13
Rank	199/200	0.112047	13	21	33	18	16	21	19	16	18	25
FFT	197/200	0.249284	21	28	19	20	13	28	16	18	22	15
NonOverlappingTemplate	198/200	0.960198	21	22	19	21	17	21	25	16	20	18
OverlappingTemplate	195/200	0.057146	24	30	20	28	17	12	19	22	14	14
Universal	200/200	0.055361	20	14	24	20	15	9	29	26	24	19
ApproximateEntropy	197/200	0.202268	18	24	28	19	25	18	16	15	12	25
RandomExcursions	173/174	0.679903	19	20	21	19	17	14	18	21	10	15
RandomExcursionsVariant	174/174	0.716158	14	13	15	14	19	22	16	19	20	22
Serial	197/200	0.788728	20	18	22	13	21	26	22	18	22	18
LinearComplexity	199/200	0.875539	22	22	18	21	13	21	24	17	21	21

5.3 Conclusion

Neither differential chosen IV attacks, nor pure statistical testing in a chosen IV setting give attacks that have potential. A differential attack would be powerful if a differential in the (a_0, a_1) update could be cancelled, but this is not possible due to the linearity of the (a_0, a_1) update. Generic statistical tests using the NIST test suite did not show any deviations from random when considering a few keystream bits from initialization with a random IV and fixed key.

6 Maximum Degree Monomial Test

The maximum degree monomial test can be used to check if all Key/IV bits are properly mixed into the state in the initialization round. The initialization procedure is seen as a Boolean function. The input to the Boolean function is a set of key/IV bits and the output is the first keystream bit. It is possible to consider also other keystram bits but as more keystream is generated, key/IV bits are mixed into the state even further. Let $\mathcal{I}$ be the set of all IV bits, $\mathcal{K}$ be the set of all key bits and let z_0 be the first keystream bit. Then we look at the function

$$z_0 = f(i_0, i_1, \ldots, i_{m-1}, k_0, k_1, \ldots, k_{n-1}) \qquad i_j \in \mathcal{I} \qquad k_j \in \mathcal{K}.$$

The idea of the test is to look at the coefficient of the monomial of highest degree in the Boolean function., i.e., the monomial $i_0 i_1 \ldots i_{m-1} k_0 k_1 \ldots k_{n-1}$. In a random Boolean function, this coefficient should be 1 with probability 0.5. The idea is to find a subset of key/IV bits, for which this coefficient is 0 with very high probability. If only IV bits are used, this will give a chosen IV distinguishing attack. If key bits are used as well, we can not relate this to any attack, since an attacker is in general not allowed to choose the value of specific key bits. However, it will still indicate a nonrandomness property in the stream cipher, which might potentially be exploited in another attack setting.

Looking at the statistics of monomials of a certain degree d was first proposed by Filiol in [16]. It was used by Saarinen to attack eSTREAM ciphers in [32]. Looking at the maximum degree polynomial was proposed by Englund, Turan and Johansson in [13]. Finding the coefficient for the maximum degree monomial of a Boolean function can be very easily done by just XORing all entries of the truth table, i.e., the coefficient is given by

$$\sum_{i_0, i_1, \ldots, i_{m-1}, k_0, k_1, \ldots, k_{n-1} \in \{0,1\}^{n+m}} f(i_0, i_1, \ldots, i_{m-1}, k_0, k_1, \ldots, k_{n-1}),$$

summing over $\mathbb{F}_2$. The complexity of finding the coefficient of the maximum degree monomial is exponential in the number of chosen key/IV bits. Finding the best subset of $n+m$ key/IV bits is an open problem. One method was given in [33]. For a very small number of bits an exhaustive search is possible. The idea was to use a greedy algorithm. The best subset of a bits is used as a starting

point when determining the subset of $a+b$ bits. The complexity of this algorithm for the chosen IV distinguishing attack is given by

$$\binom{|\mathcal{I}|}{i} + \sum_{j=0}^{m+n} \binom{|\mathcal{I}| - i - j \cdot b}{b}$$

and for a nonrandomness detector the complexity is given by

$$\binom{|\mathcal{I}| + |\mathcal{K}|}{i} + \sum_{j=0}^{m+n} \binom{|\mathcal{I}| + |\mathcal{K}| - i - j \cdot b}{b}.$$

The greedy bit determining algorithm has been shown to give very good results on e.g., Grain-128 and Trivium. This algorithm has been applied to Enocoro-128v2. We have put $i = 1$ and performed the maximum degree monomial test using the greedy bit set algorithm for $b = 1$, $b = 2$ and $b = 3$. We have also performed the test for both $n = 0$, corresponding to a chosen IV distinguishing attack and for $n = 128$, corresponding to a nonrandomness test.

The test is performed as follows. The initialization procedure of Enocoro-128v2 has been modified such that it will output 8 bits in each initialization round. These bits are saved in a vector x. Thus, the first 8 in the vector correspond to the first keystream byte if 0 initialization rounds were used, the next 8 bits in x correspond to the first output byte if 1 initialization round was used and so on. All vectors x, each corresponding to one choice of the $a+b$ bits in the key/IV are XORed and the number of starting zeroes in the resulting vector is counted. The full initialization of Enocoro-128v2 is 96 rounds, i.e., $96 \cdot 8 = 768$ bits are suppressed in the initialization. The key/IV bits that are not used in the test are fixed to 0. Results are given in Table 5. Figures 2 and 3 gives a graphical overview of the performance of the test.

6.1 Conclusion

The initialization function of Enocoro-128v2 has high resistance against the maximum degree monomial test. A chosen IV distinguishing attack against 21 initialization rounds has been identified. It should be noted that the key bits and the remaining IV bits were fixed to 0 in this test. Previous results on Grain-128 and Trivium suggests that this is the most advantegous choice from the attacker's point of view. For a random key, the attack should be slightly worse.

It is possible to consider several other variants of this attack. Instead of looking at the full 8-bit word we could consider only one bit in the word. Also any linear combination of the bits in a byte can be targeted. Depending on the strategy, it is likely that the attack could be improved by a few rounds. However, the full initialization of 96 rounds make a comfortable security margin against this attack.

7 Cube Attacks

Cube attacks have been proposed in [10] and has been used in e.g., [1,2]. A very similar attack is the AIDA attack proposed by Vielhaber in [34].

Table 5. Results for the maximum degree monomial test. The maximum number of leading zeroes in the XOR of all vectors x, the size of the bit set that gave the maximum number of leading zeroes, the bits in the bit-set and the maximum size of the tested bit sets are given.

		Leading Zeroes (%)	Bit-set size	Bit-set	Max Size
Dist	$b = 1$	178/768 (23.2%)	20	12, 7, 23, 18, 16, 30, 19, 38, 31, 39, 20, 32, 37, 17, 29, 22, 21, 57, 41, 14	30
	$b = 2$	180/768 (23.4%)	19	12, 7, 23, 28, 46, 17, 22, 36, 38, 16, 18, 19, 43, 20, 21, 5, 15, 29, 55	23
	$b = 3$	174/768 (22.7%)	19	12, 18, 34, 46, 9, 17, 30, 20, 23, 31, 11, 14, 25, 21, 26, 27, 16, 19, 22	19
NR	$b = 1$	161/768 (21.0%),	26	**Key bits:** 2, 69, 77, 44, 31, 126, 62, 61, 5, 122 **IV bits:** 12, 32, 28, 25, 16, 21, 10, 8, 26, 9, 23, 30, 42, 38, 39, 11	26
	$b = 2$	162/768 (21.09%)	3	**Key bits:** 61, 115 **IV bits:** 12	23
	$b = 3$	166/178 (21.61%)	4	**Key bits:** 35, 122 **IV bits:** 12, 1	16

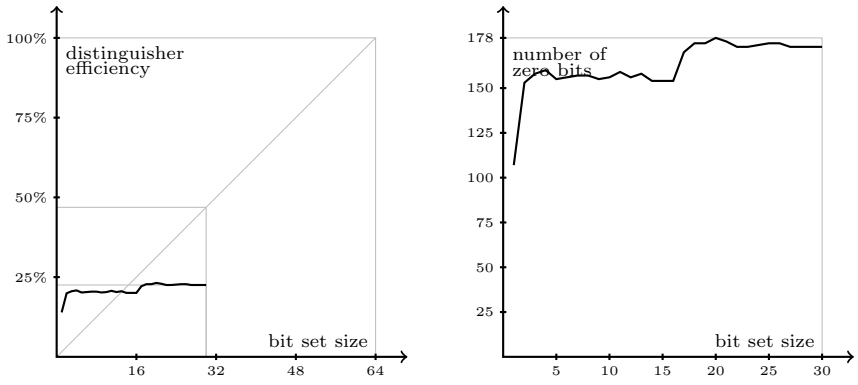

Fig. 2. Plots showing the number of initialization rounds that fails the maximum degree monomial test for a given bit-set size. Only IV bits are allowed in the greedy algorthm (corresponding to a chosen IV distinguishing attack). 178 leading zero bits correspond to a possible distinguishing attack on 21 initialization rounds of Enocoro-128v2, since after 21 rounds, bits $169 - 176$ are output.

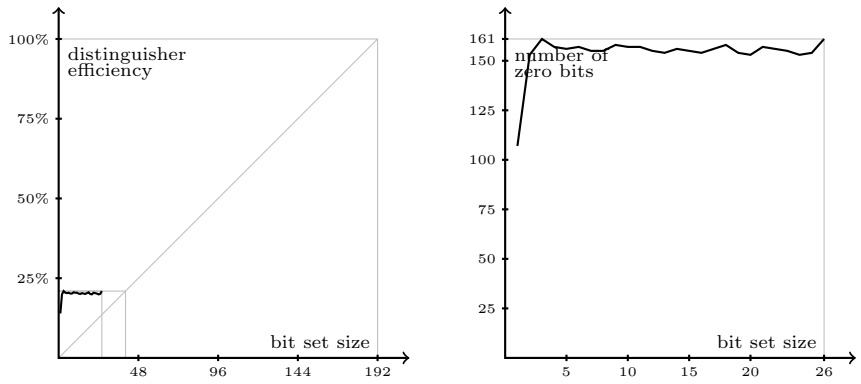

Fig. 3. Plots showing the number of initialization rounds that fails the maximum degree monomial test for a given bit-set size. Both key and IV bits are allowed in the greedy algorithm. Note that this plot is worse than the case when only IV bits are used. This is an artifact of the greedy algorithm. If all bit selection choices were tested the result would have been at least as good as in the case when only IV bits were chosen.

The idea of cube attacks shares several properties of the maximum degree monomial test, or more generally, a d-monomial test. Instead of iterating over all bits in a set in order to find the coefficient, which is zero or one depending on if the monomial is present in the Boolean function, we only iterate over a subset of the bits. In that case the coefficient can be an expression in key/IV bits instead. In particular, if this expression is a linear function of key bits, knowing the coefficient can help recovering the key by solving a set of equations. The resistance against the maximum degree monomial test, indicate that Enocoro-128v2 has strong resistance also against the cube attacks.

7.1 Conclusion

We have not found any way of mounting cube attacks on Enocoro-128v2. Moreover, the resistance against these attacks seems strong due to the large number of initialization rounds in the stream cipher.

8 Linear Distinguishing Attacks

8.1 Background

The generator should efficiently produce random-looking keystreams that are as "indistinguishable" as possible from true randomness. We always assume a known-plaintext attack (or chosen-plaintext or chosen-ciphertext), which is equivalent to having access to the keystream $\mathbf{z} = z_0, z_1, z_2, \ldots$. So we assume that an output sequence $\mathbf{z}$ is known. We can then try to perform different types of attacks. One is a *key recovery attack*, trying to recover the value of the secret

key K. Here we consider instead a *distinguishing attack:*, when we try to determine whether a given sequence $\mathbf{z} = z_0, z_1, z_2, \ldots$ is likely to have been generated from the considered stream cipher or whether it is just a truly random sequence. A *distinguisher* is an algorithm (or a black box) that takes (at least) the given sequence as an input and with high probability answers the above question correctly. We refer to standard literature for more details on why distinguishing attacks are a weakness for a cipher.

Let $D(\mathbf{z})$ be an algorithm that takes input $\mathbf{z}$ and as output gives one out of two possible answers, either "X" or "RANDOM". Assume the sequence $\mathbf{z}$ is either produced by the generator or that it is a purely random sequence (both occuring with the same probability). The probability that $D(\mathbf{z})$ correctly determines the origin of $\mathbf{z}$ is written $(1 + \varepsilon)/2$. If ε is not very close to zero we say that $D(\mathbf{z})$ is a distinguisher and we have a distinguishing attack on the generator. It is standard to introduce the advantage of a distinguisher D, Adv_D, as

$$\mathrm{Adv}_D = |P(D(\mathbf{z}) = \mathrm{X}|\mathbf{z} \text{ generated by X}) - P(D(\mathbf{z}) = \mathrm{X}|\mathbf{z} \text{ truly random})|.$$

In terms of ε as above, the advantage is simply written $\mathrm{Adv}_D = |\varepsilon|$.

This basic scenario can be generalized to include several keystreams produced from different known or possibly chosen IVs.

A distinguisher should detect some behavior that appears to be nonrandom. A first and basic approach is to apply various statistical tests on the received keystream $\mathbf{z}$. This has been done in previous sections where special statistical packages have been used to analyze the output of a generator and check if there is some statistical property that looks suspicious. These approaches may detect statistical weaknesses in some weak generators but they are not too powerful in general. The problem is that the tests are generic and not targeted towards the considered cipher. Stronger attacks may be achieved if we take the internal structure of the cipher into account, when we design a distinguisher.

In essence, we would try to detect a statistical deviation in the keystream $\mathbf{z}$ based on some internal relationship. However, symbols in $\mathbf{z}$ (or even small blocks of symbols) will often be very close to the uniform distribution. Instead, the internal relationship often gives dependence among different z_i symbols that can be far apart in time. So it is natural that we transform our keystream $\mathbf{z}$ into a new sequence of symbols, called *samples*, denoted by $\mathbf{x} = x_0, x_1, x_2, \ldots$. In general, this can be done in almost any way,

$$x_i = F(i, \mathbf{z}), i = 0, 1, 2, \ldots$$

where F is some function. With a given sample sequence, we would finally try to distinguish if $\mathbf{x}$ behaves as if generated from a truly random $\mathbf{z}$ or not.

A common type of distinguishers, *linear distinguishers*, select F as a linear function, i.e., the samples are selected as linear combinations of keystream bits. Usually, the samples are regarded as independent and the distinguisher examines whether the sample values are consistent with a uniform distribution or not.

Continuing, we need to find a suitable way to transform the keystreams to a sample sequence $\mathbf{x}$. Once the sample sequence is given, we apply statistical tools

to analyze which distribution the sample sequence follows, and hence the answer is given by the distinguisher.

We do not give the details about the statistical tools we use. These can be found in many places, e.g. [19].

8.2 Linear Approximations in Enocoro-128v2

We adopt the following strategy. We linearize the cipher by replacing nonlinear blocks with linear ones. We try of course to use the linear approximations that give the best attack. The approximation of a nonlinear block can be modeled as replacing it with some linear block together with some added noise. As a second step we find a linear relationship among keystream symbols, where the relationship involves as few approximated blocks as possible.

If the linear relationship is written as

$$x_t = \sum_{j=0}^{k} c_j z_{t+j}$$

then $x_t = 0$ if no approximation was done. However, since linear approximations were introduced we can instead write x_t as a sum of noise variables. The noise is usually nonuniformly distributed, and so is then often the sum of them. If we denote this distribution by P_0, then the samples are distributed according to P_0 in the cipher case. In the other case, samples are uniformly distributed (P_1). Hypothesis testing is used to distinguish between the two distributions.

For Enocoro-128v2 we note that the cipher is byte-oriented, so it makes sense to initially assume that symbols are bytes, or belonging to the field of 2^8 elements $\mathbb{F}_2^8$. We also note that the only nonlinear operation in the cipher is the S-box. The L transformation is linear in $\mathbb{F}_2^8$. The linear approximation of s_8 is then written as

$$S[x] = \alpha_{i,t} x + N_{i,t},$$

where $0 \leq i \leq 3$ and $t \geq 0$. Here $\alpha_{i,t}$ is some arbitrary constant that fixes a certain linear approximation for the S-box considered and $N_{i,t}$ is the noise introduced by the linear approximation. The more nonrandom noise, the better approximation in general. Also, there are four S-boxes in every time instance, so i determines which S-box we consider. Finally, approximations can be done in any time t.

In order to proceed with the linear distinguishing attack we write the keystream bytes z_{t+i} as a function of the state bytes at time t, denoted $B_0, B_1, B_2, \ldots, B_{31}, A_0$, as follows.

$$
\begin{aligned}
z_{t+1} &= A_0 + S[B_2] + d * z_t + d * S[B_7] + S[B_{29}],\\
z_{t+2} &= A_0 + S[B_2] + z_t + S[B_7] + S[B_{16}] + S[B_1] + d * z_{t+1} + d * S[B_6] +\\
&\quad + S[B_{28}],\\
z_{t+3} &= A_0 + S[B_2] + z_t + S[B_7] + S[B_{16}] + S[B_1] + z_{t+1} + S[B_6] +\\
&\quad + S[B_{15}] + S[B_0] + d * z_{t+2} + d * S[B_{05}] + S[B_{27}],\\
z_{t+4} &= A_0 + S[B_2] + z_t + S[B_7] + S[B_{16}] + S[B_1] + z_{t+1} + S[B_6] +\\
&\quad + S[B_{15}] + S[B_0] + z_{t+2} + S[B_5] + S[B_{14}] + S[B_{31} + A_0] +\\
&\quad + d * z_{t+3} + d * S[B_4] + S[B_{26}],\\
z_{t+5} &= A_0 + S[B_2] + z_t + S[B_7] + S[B_{16}] + S[B_1] + z_{t+1} + S[B_6] +\\
&\quad + S[B_{15}] + S[B_0] + z_{t+2} + S[B_5] + S[B_{14}] + S[B_{31} + A_0] + z_{t+3} +\\
&\quad + S[B_4] + S[B_{13}] + S[B_{30} + A_0 + S[B_2] + z_t + S[B_7] + S[B_{16}]] +\\
&\quad + d * z_{t+4} + d * S[B_3] + S[B_{25}],\\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots
\end{aligned}
$$

Continuing in this way we can write any keystream byte in a similar fashion. The strategy in our linear distinguisher is now to approximate an S-box as a linear function with some added noise, as described above. Since there are in total 33 unknown variables, writing 34 such equations will always allow us to write a sum of keystream bytes as a sum of only noise variables. This will result in a distinguisher for the keystream. The sum of keystream bytes gives us one (biased) sample x_i. Using a sequence $\mathbf{x}$ of samples, a hypothesis test can be performed. The strength of the distinguisher is then measured as the number of samples required in order to make a reliable decision in the hypothesis test, i.e., the number of samples such that the error probability in the test is lower than 0.5. The number is samples is approximately $1/\epsilon^2$, where ϵ is the bias of the samples. The factors that determine the bias is the bias of the approximation of the S-box and the number of S-boxes that has to be approximated.

The bias of the noise introduced by a linear approximation of an S-box will depend on the which approximation is used. Since we are in the field $\mathbb{F}_2^8$ there are $2^8 - 1$ nonzero approximations of the S-box (the noise from the zero approximation does not have a bias since the S-box is a bijection). Tables 6 and 7, given in the Appendix, gives the bias for each approximation. We compute the bias as

$$\epsilon = \sum_{i=0}^{256} |\Pr[i] - \frac{1}{256}| \tag{1}$$

where $\Pr[i]$ is the probability that the noise takes the value $i \in \mathbb{F}_2^8$. This equation for the bias is accurate if $\Pr[i]$ is close to $1/256$ for each i. In order to get an accurate value we give the bias for the XOR sum of 16 distributions in the tables. Studying these values we can conclude that

1. There is no one approximation that is *significantly* better than other approximations.
2. The best approximation is given by the polynomial 0x78.

We take a somewhat simplified approach, namely that an S-box multiplied by the factor 1 is approximated by the identity function and that an S-box multiplied by the factor $d = \alpha = 0x02$ is approximated by the factor $d^{-1} = \alpha^{254} = 0x8e$. Then each state bit in the equations, after approximations has been done, will have the constant factor 1. In the resulting expression we will have three possibilities for the noise variables.

1. A noise variable occurs once, stemming from an approximation by 1.
2. A noise variable occurs once, stemming from an approximation by d^{-1}.
3. A noise variable occurs twice, once from each approximation. This corresponds to an approximation by $(1+d)^{-1} = \alpha^{230} = 0xf4$.

Following the approach described above, we tested different possibilities for the attack. One possible distinguisher is given by the biased sum

$$\begin{aligned} 0 = {} & z_t + z_{t+1} + z_{t+4} + z_{t+6} + z_{t+8} + z_{t+12} + z_{t+13} + z_{18} + z_{t+21} + z_{t+22} + \\ & z_{t+24} + z_{t+25} + z_{t+28} + z_{t+32} + z_{33} + z_{34} + d \cdot z_t + d \cdot z_{t+1} + d \cdot z_{t+3} + \\ & d \cdot z_{t+5} + d \cdot z_{t+8} + d \cdot z_{t+12} + d \cdot z_{t+15} + d \cdot z_{t+16} + d \cdot z_{t+17} + d \cdot z_{t+21} + \\ & d \cdot z_{t+24} + d \cdot z_{t+28} + d \cdot z_{t+32} + d \cdot z_{t+33} \end{aligned}$$

The bias for this distinguisher is $\epsilon = 2^{-180}$ which means that the number of samples, and the amount of keystream bytes, required in order to distinguish the biased sequence from a completely random sequence is in the order of 2^{360}. All other biased sums of keystream bytes that we found required more samples (and keystream) than this.

8.3 Conclusion

The number of keystream bytes needed in the most efficient distinguishing attack found by our method is about 2^{360}. Even though it is likely possible to find better attacks by looking at other equations and thus, other combinations of keystream bytes, this number indicates that Enocoro-128v2 has a comfortable security margin for this attack. Looking at other approximations of the S-boxes than the ones we used could also improve the attack, but probably only slightly. We have noted that if we XOR different distributions, the bias of this sum has a significantly smaller bias than if we XOR the same distribution. As an example, taking the XOR of the 16 distributions resulting from approximating 16 S-boxes by 0x01, 0x02, ..., 0x16 respectively would have the bias $\epsilon = 2^{-62.48}$, as compared to values around 2^{-35} (see Tables 6 and 7) when taking the XOR of the same distribution.

We can note that there might be better ways to construct biased sums of keystream bits, different from our method. It might be possible to find biased relations involving keystream bits that are very far apart, exploiting some internal property of the generator. However, we have not found any way of doing this. According to our analysis, Enocoro-128v2 resists linear distinguishing attacks.

9 Guess-and-Determine Attacks

In a guess-and-determine attack, the contents of some state variables are guessed during the process. Based on this guess, the contents of other state variables are determined. The basic idea is quite trivial, but finding the best procedure of guessing and determining state variables in order to achieve minimum computational complexity can be quite complicated.

A first attempt to attack Enocoro-128v2 in this fashion is described as follows. We assume we know the (a_0, a_1) variables all the time. In order to keep this true, we need to guess the contributions from the S-boxes. Assume that the initial state is $(B_0, B_1, \ldots, B_{31})$ in the buffer and (A_0, A_1) for the state a. By $\hat{c}$ we simply mean a known value is added to c. We proceed a few steps. We start with $a_0(0) = A_0, a_1(0) = z_0$ and a buffer

$$[B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15},$$
$$B_{16}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, B_{26}, B_{27}, B_{28}, B_{29}, B_{30}, B_{31}]$$

As we have $a_1(0) = z_0$ we know A_1 and guess only A_0. In order to be able to compute the next state a we need to guess values from four S-boxes, i.e., we guess B_2, B_7, B_{16}, B_{29}. As $a_1(1) = z_1$ this gives us one equation in guessed variables that must hold, so in total we guess A_0 and three buffer bytes, or 4 bytes. This gives us a known $a(1)$ with $a_1(1) = z_1$ and an updated buffer containing

$$[\hat{B_{31}}, B_0, B_1, \hat{B_6}, B_3, B_4, B_5, B_6, \hat{B_{15}}, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14},$$
$$B_{15}, \hat{B_{28}}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, B_{26}, B_{27}, B_{28}, \hat{0}, B_{30}]$$

Again, to be able to derive $a(2)$, we need to guess the S-boxes $S[B_1]$, $S[B_6]$, $S[B_{15}]$, $S[B_{28}]$, i.e., B_1, B_6, B_{15}, B_{28}. Then $a_1(2) = z_2$ gives us one equation, so the total number of guessed variables is now 7 and the buffer state is

$$[\hat{B_{30}}, \hat{B_{31}}, B_0, \hat{B_5}, \hat{0}, B_3, B_4, B_5, \hat{B_{14}}, \hat{0}, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13},$$
$$B_{14}, \hat{B_{27}}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, B_{26}, B_{27}, \hat{0}, \hat{0}].$$

In the next step, we proceed as before and guess B_0, B_5, B_{14}, B_{27}. With one equation we reach 10 guessed bytes, known $a(3)$ and a buffer at time 3 with content

$$[\hat{0}, \hat{B_{30}}, \hat{B_{31}}, \hat{B_4}, \hat{0}, \hat{0}, B_3, B_4, \hat{B_{13}}, \hat{0}, \hat{0}, B_8, B_9, B_{10}, B_{11}, B_{12},$$
$$B_{13}, \hat{B_{26}}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, B_{26}, \hat{0}, \hat{0}].$$

One more step gives guesses on $B_{31}, B_4, B_{13}, B_{26}$, a total of 13 guessed bytes and at time 4 a known $a(4)$ and buffer

$$[\hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{0}, \hat{0}, B_3, \hat{B_{12}}, \hat{0}, \hat{0}, \hat{0}, B_8, B_9, B_{10}, B_{11},$$
$$B_{12}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, \hat{0}, \hat{0}].$$

In the next step, one possibility is to skip guessing buffer values and just guess the next a_0 value, the reason being that known values are used as input to two S-boxes in the next step. So we would assume a given value on $a_0(5)$, which is a function of B_{30}, B_3, B_{12}. An equation connects the output $a_1(5) = z_5$ to B_{30}, B_3, B_{25}. Notice that we here have two known equations that are not explored at this point!

This gives 14 guessed bytes and a known $a(5)$ and the buffer is

$$[\hat{0}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{0}, \hat{0}, B_3 + B_{11}, \hat{B_{12}}, \hat{0}, \hat{0}, \hat{0}, B_8, B_9, B_{10},$$
$$B_{11}, B_{12} + B_{24}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, \hat{0}].$$

Now the inputs to S-boxes taken from position 2 and 7 are known. So only two values need to be guessed, B_{11}, B_{24}. But one equation from the output z_6 makes this only one byte guess and the total number of guessed bytes is 15. The buffer is

$$[\hat{0}, \hat{0}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{0}, \hat{B_{10}}, \hat{B_3}, \hat{B_{12}}, \hat{0}, \hat{0}, \hat{0}, B_8, B_9,$$
$$B_{10}, \hat{B_{23}}, \hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, \hat{0}, B_{25}].$$

We can now continue guessing values. Obviously, we now get a complexity that is higher than exhaustive key search. But it is still of interest to see how good such an approach can be. So continuing, we see again that guessing B_{10}, B_{23} together with the output equation adds one more guessed byte, 16 bytes in total, a known value of $a(7)$ and buffer

$$[\hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{B_9}, \hat{0}, \hat{B_3}, \hat{B_{12}}, \hat{0}, \hat{0}, \hat{0}, B_8,$$
$$B_9, \hat{B_{22}}, \hat{0}, \hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, \hat{0}, \hat{0}].$$

Next, B_9, B_{22} are guessed, giving 17 guessed bytes, known $a(8)$ and buffer at time 8 being

$$[\hat{0}, \hat{B_{25}}, \hat{0}, \hat{B_3}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{B_8}, \hat{0}, \hat{0}, \hat{B_3}, \hat{B_{12}}, \hat{0}, \hat{0}, \hat{0},$$
$$B_8, \hat{B_{21}}, \hat{0}, \hat{0}, \hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, \hat{0}, \hat{0}].$$

At some point we may stop guessing variables, and collect a number of equations to process. At this point we could express the new a_0 as $a_0(9) = \ell(S[\hat{B_3}], S[B_8])$ and $a_1(9) = z_9 = \ell(S[\hat{B_3}], S[B_{21}])$, where $\ell()$ denotes an affine function of some variables. Then the buffer at time 9 can be written

$$[\hat{0}, \hat{0}, \hat{B_{25}}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{B_8}, \hat{0}, \hat{0}, \hat{B_3}, \hat{B_{12}}, \hat{0}, \hat{0},$$
$$\hat{0}, B_8 + B_{20}, \hat{B_{21}}, \hat{0}, \hat{0}, \hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, \hat{0}].$$

Note that we have one unexplored equation from $z_9 = \ell(S[\hat{B_3}], S[B_{21}])$.

In the next time instance we can write $a_0(10) = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}])$ and $a_1(10) = z_{10} = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{20}])$. The buffer at time 10 is

$$[a_0(9), \hat{0}, \hat{0}, \hat{B_{25}}, \hat{B_{30}}, \hat{B_3}, \hat{0}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{B_8}, \hat{0}, \hat{0}, \hat{B_3}, \hat{B_{12}}, \hat{0},$$
$$\hat{0}, \hat{B_{19}}, B_8 + B_{20}, \hat{B_{21}}, \hat{0}, \hat{0}, \hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}].$$

In the next step, all S-box contributions but B_{19} are known, so $a_0(11)$ is as $a_0(10)$, i.e., $a_0(11) = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}])$ and $a_1(11) = z_{11} = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{19}])$. The buffer at time 11 is

$$[B_{21} + a_0(10), a_0(9), \hat{0}, \hat{0}, \hat{B_{25}}, \hat{B_{30}}, \hat{B_3}, \hat{0},$$
$$\hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{B_8}, \hat{0}, \hat{0}, \hat{B_3}, \hat{B_{12}},$$
$$\hat{0}, \hat{B_{18}}, \hat{B_{19}}, B_8 + B_{20}, \hat{B_{21}}, \hat{0}, \hat{0}, \hat{B_{12}},$$
$$\hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}, B_{20}].$$

In the next step, the first two inputs to the S-boxes are given, so $a_0(12)$ is given from $a_0(11)$ or $a_0(12) = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{12}}], S[\hat{B_{25}}], S[\hat{B_{30}}])$ and $a_1(12) = z_{12} = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{18}])$. The buffer at time 12 is

$$[B_{20} + a_0(11), B_{21} + a_0(10), a_0(9), \hat{B_3}, \hat{0}, \hat{B_{25}}, \hat{B_{30}}, \hat{B_3},$$
$$\hat{B_{12}}, \hat{0}, \hat{B_{30}}, \hat{B_3}, \hat{B_8}, \hat{0}, \hat{0}, \hat{B_3},$$
$$\hat{B_{12}}, \hat{B_{17}}, \hat{B_{18}}, \hat{B_{19}}, B_8 + B_{20}, \hat{B_{21}}, \hat{0}, \hat{0},$$
$$\hat{B_{12}}, \hat{B_{25}}, \hat{0}, \hat{0}, \hat{0}, B_{17}, B_{18}, B_{19}].$$

In the next step, two inputs to the S-boxes are given, so we can write $a_0(13) = \ell(a_0(12), S[\hat{a_0(9)}])$ and $a_1(13) = z_{13} = \ell(S[\hat{a_0(9)}], S[\hat{B_3}], S[B_8], S[\hat{B_{12}}], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{17}])$.

Now the expressions in the next output byte z_{14} will include a more complicated expression. Without processing this to a detailed end, it looks like a few more guessed bytes are needed to reach a unique state value. This would probably allow a quick recovering of the remaining part of the buffer but result in 20-22 guessed bytes in total, or a complexity of at least 2^{160}. A better attack has been published in [23]. This paper was not available to us as it is in Japanese.

9.1 More Equations From Clocking Backwards

Recall that we previously started with $a_0(0) = A_0, a_1(0) = z_0$ and a buffer

$$[B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15},$$
$$B_{16}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, B_{26}, B_{27}, B_{28}, B_{29}, B_{30}, B_{31}].$$

There are some additional information that can be used to improve the attack. From time 0 we looked ahead and tried to determine buffer values by guessing and using output words from time 0 and onwards. But it is clear that time 0 is only a starting point that can be any actual time. If we take any such time there might be output words before time 0 that can be used. One way to model this is to try to clock the entire generator backwards, i.e. move from time 0 to -1 etc.

First we recall that the L transformation used to get the next a value is described by the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & d \end{pmatrix}.$$

Going in the opposite direction would mean going through the inverse linear transformation, given as

$$(1+d)^{-1}\begin{pmatrix} d & 1 \\ 1 & 1 \end{pmatrix}.$$

¿From the starting values of the buffer given above, we clock once backwards. Then we obtain

$$a_1(-1) = z_{-1} = \ell(A_0, S[B_{30}], S[B_{17}+B_{29}], S[B_8+B_{16}]),$$

and

$$a_0(-1) = \ell(A_0, S[B_{30}], S[B_{17}+B_{29}], S[B_3+B_7]).$$

The buffer at time -1 is then

$$\begin{aligned}[B_1, B_2, B_3+B_7, B_4, B_5, B_6, B_7, B_8+B_{15},\\ B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15}, B_{16},\\ B_{17}+B_{29}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24},\\ B_{25}, B_{26}, B_{27}, B_{28}, B_{29}, B_{30}, B_{31}, B_0+a_0(-1)]\end{aligned}$$

One more step gives

$$a_1(-2) = z_{-2} = \ell(a_0(-1), S[B_{31}], S[B_{18}+B_{30}], S[B_9+B_{17}+B_{29}]),$$

and

$$a_0(-2) = \ell(a_0(-1), S[B_{31}], S[B_{18}+B_{30}], S[B_4+B_8+B_{15}]).$$

The buffer at time -1 is then

$$\begin{aligned}[B_2, B_3+B_7, B_4+B_8+B_{15}, B_5, B_6, B_7, B_8+B_{15}, B_9+B_{17}+B_{29},\\ B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15}, B_{16}, B_{17}+B_{28}\\ , B_{18}+B_{30}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25},\\ B_{26}, B_{27}, B_{28}, B_{29}, B_{30}, B_{31}, B_0+a_0(-1), B_1+a_0(-2)]\end{aligned}$$

Let us write out one more equation,

$$a_1(-3) = z_{-3} = \ell(a_0(-2), S[B_0+a_0(-1)], S[B_{19}+B_{31}], S[B_{10}+B_{18}+B_{30}]).$$

We now try to combine these observations with our previous attempt to guess variables. Let us return to the situation when 17 bytes had been guessed. Looking carefully at this situation gives that we know all bytes but B_3, B_8, B_{12}, B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{25}, B_{30}.

We have unexplored equations from time 5,

$$\begin{aligned} c_0 &= \ell(S[\hat{B_{30}}], S[B_3], S[B_{12}]),\\ z_5 &= \ell(S[\hat{B_{30}}], S[B_3], S[B_{25}]),\end{aligned}$$

where c_0 is the value of the guessed $a_0(5)$. We have two equations from backward clocking,

$$z_{-1} = \ell(S[B_{30}], S[\hat{B_{17}}], S[\hat{B_8}]),$$
$$z_{-2} = \ell(S[B_{30}], S[\hat{B_{17}}], S[\hat{B_3}], S[B_{18} + B_{30}], \hat{B_{17}}),$$

and also

$$z_{12} = \ell(S[\hat{B_3}], S[B_8], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{18}])$$
$$z_{13} = \ell(S[\ell(S[\hat{B_3}], S[B_8])], S[\hat{B_3}], S[B_8], S[\hat{B_{12}}], S[\hat{B_{25}}], S[\hat{B_{30}}], S[B_{17}])$$

With one more guessed byte, we can reach a solution for the 7 bytes involved in these 6 equations. If by no other means, we can simply tabulate all possible such systems of equations that can occur and in the table write down the solution. Using the equations for z_9 to z_{11} we can then derive the complete state and check its correctness. Using a huge table this would give an attack with complexity around 2^{152}. Time constraints limited us in further investigations and providing detailed explanations.

9.2 Conclusions

We have not been able to find an attack with complexity lower than 2^{152}. There seems to be a potential for investigating this type of attack further and look at the situation when fewer bytes are guessed, and then writing up the system of equations. Then finding the solutions by pure table lookup methods or by some more advances method of solving nonlinear equations. This is a direction we recommend is studied further.

10 Algebraic Attacks

The idea of algebraic attacks is to attack the stream cipher by solving a system of equations. These equations tend to be nonlinear. A common scenario in the case of stream ciphers is when there is a part of the state being linearly updated. Assume one can obtain a nonlinear equation f that depends only on the linearly updated part of the form

$$z_0 = f(s_0, s_1, \ldots, s_{m-1}),$$

where $(s_0, s_1, \ldots, s_{m-1})$ is the linearly updated part, updated by T. Then one can obtain an arbitrary number of equations of the same degree as f by

$$\begin{aligned} z_0 &= f(s_0, s_1, \ldots, s_{m-1}) \\ z_1 &= f(T(s_0, s_1, \ldots, s_{m-1})) \\ z_2 &= f(T^2(s_0, s_1, \ldots, s_{m-1})) \\ &\vdots \quad \vdots \end{aligned}$$

A linearization of the above system means that we replace every monomial up to the degree of f by a new variable. We receive a linear system of equations and solve it using Gaussian elimination or some other method. Several ways to improve this basic approach exists, e.g., [6, 7, 15].

Regarding Enocoro-128v2, this basic approach for algebraic attacks on stream ciphers does not work, as it does not have a part of the state that is linearly updated. Because of the feedback from a_0 into the buffer, nonlinear expressions are inserted into the buffer and through the S-boxes and feedback, the degrees of these expressions grow.

To apply an algebraic attack, we would need to perform the attack more in style of algebraic attacks on block ciphers. Here we typically set up a system of nonlinear equations by introducing new intermediate variables. It is well known that the general problem of solving a system of nonlinear equations in NP-complete, so in order to be successful we need to come up with a system of equations that has some deviating property. Such properties could for example be an overdefined system or a very sparse system.

Regarding Enocoro-128v2, the most obvious approach would be to introduce a new intermediate variable for $a_0(t)$ in each time instance. We would then get a system of equations consisting of two equations from each time instance. With variables $B_0, B_1, \ldots B_{31}$ and A_0 as starting values we then get

$$\begin{aligned}
D_1 &= A_0 + S[B_2] + S[B_8] + z_0 + S[B_{16}],\\
z_1 &= A_0 + S[B_2] + d \cdot (S[B_8] + z_0) + S[B_{29}],\\
D_2 &= D_1 + S[B_1] + S[B_7] + z_1 + S[B_{15}],\\
z_2 &= D_1 + S[B_1] + d \cdot (S[B_7] + z_1) + S[B_{28}],\\
D_3 &= D_2 + S[B_0] + S[B_6] + z_2 + S[B_{14}],\\
z_3 &= D_2 + S[B_0] + d \cdot (S[B_6] + z_2) + S[B_{27}],\\
D_4 &= D_3 + S[B_{31} + A_0] + S[B_5] + z_3 + S[B_{13}],\\
z_4 &= D_3 + S[B_{31} + A_0] + d \cdot (S[B_5] + z_3) + S[B_{26}],\\
D_5 &= D_4 + S[B_{30} + D_1] + S[B_4] + z_4 + S[B_{12}],\\
z_5 &= D_4 + S[B_{30} + D_1] + d \cdot (S[B_4] + z_4) + S[B_{25}],\\
&\vdots \quad \vdots
\end{aligned}$$

After 33 time instances the system of equations will have one or a few solutions. Using a few more time instances would give us an overdefined system at the expense of increasing the number of variables.

We have not been able to find any particular weakness in the system of equations that could potentially be explored. It is difficult to judge the strength against methods trying to solve such a system of equations. Typical methods use the XL algorithm [6], or techniques based on Gröbner bases [5], like the F4 or F5 algorithms [14, 15], Methods using SAT-solvers, e.g., [12] or the related Agreeing-Glueing algorithm [31] are also possible approaches.

Unfortunately, the usefulness of all these methods is very hard to predict on full size systems, as the complexity is not easy to model and it is in general very difficult to predict the complexity from simulating smaller instances of systems.

10.1 Conclusion

The nonlinear update of the buffer makes standard algebraic attacks on stream ciphers not applicable. A system of 66 quite simple nonlinear equations in 66 unknown variables over $\mathbb{F}_2^8$ can be given. We have not identified any weaknesses that would allow an efficient way of solving the system of equations using known methods related to Gröbner bases techniques or other similar approaches.

11 Conclusions

After considerable work, going through various kinds of techniques of cryptanalysis we come to the final conclusions regarding the stream cipher Enocoro-128v2.

We believe that Enocoro-128v2 is a very solid construction. It appears to have a very good resistance against all types of cryptanalytic attacks. Still, the implementation cost is quite low.

Regarding the various attack techniques we have examined, the different resynchronization attacks (differential chosen IV, max degree tests and cube attacks) do not come close to the full number of initialization rounds. Attacks on the previous version, using related keys and sliding techniques are not possible, due to the introduced counter.

The TMD tradeoff attacks do not reveal any weaknesses. In the linear distinguishing attack scenario, we could not find a linear relationship among output bytes with low weight, or equivalently, high bias. Our numerical values here can clearly be improved, but the feeling is that we would still be very far from an attack better than exhaustive key search.

The perhaps most promising attack is the guess-and-determine type of attack. Even though we could not find such an attack better than exhaustive key search, we feel that here we are not too far away from this boundary. We would recommend to further study this type of attack, possibly in combination with some algebraic method of solving systems of equations.

References

1. J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, and A. Shamir. Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128. In *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems (SHARCS'09)*, 2009.
2. J.-P. Aumasson, I. Dinur, W. Meier, and A. Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In O. Dunkelman, editor, *Fast Software Encryption 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2009.

3. S. H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *ECOS95 (European Convention on Security and Detection)*, volume 408. IEE Conference Publication, 1995.
4. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000.
5. Bruno Buchberger. Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal". *Journal of Symbolic Computation"*, 41(3-4):475 – 511, 2006. Logic, Mathematics and Computer Science: Interactions in honor of Bruno Buchberger (60th birthday).
6. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.
7. N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Y. Zheng, editor, *Advances in Cryptology—ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.
8. J. Daemen and C. Clapp. Fast hashing and stream encryption with PANAMA. In *Fast Software Encryption'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 1998.
9. J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronization weaknesses in synchronous stream ciphers. In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 159–169. Springer-Verlag, 1994.
10. I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In A. Joux, editor, *Advances in Cryptology—EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer-Verlag, 2009.
11. O. Dunkelman and N. Keller. Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. *Information Processing Letters*, 107(5):133–137, 2008.
12. N. Eén and N. Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.
13. H. Englund, T. Johansson, and M. S. Turan. A framework for chosen IV statistical analysis of stream ciphers. In K. Srinathan, C. Pandu Rangan, and M. Yung, editors, *Progress in Cryptology - INDOCRYPT 2007*, volume 4859/2007 of *Lecture Notes in Computer Science*, pages 268–281. Springer-Verlag, 2007.
14. J. C. Faugére. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1-3):61 – 88, 1999.
15. J. C. Faugére. A new efficient algorithm for computing Gröbner bases without reduction to zero. In *Workshop of Applications of Commutative Algebra*. ACM Press, 2002.
16. E. Filiol. A new statistical testing for symmetric ciphers and hash functions. In R. Deng, F. Bao, J. Zhou, and S. Qing, editors, *ICICS—2002*, volume 2513/2002 of *Lecture Notes in Computer Science*, pages 342–353. Springer-Verlag, 2002.
17. H. Furuichi, K. Muto, D. Watanabe, and T. Kaneko. Security evaluation of enocoro-80 against differential resynchronization attack. In *The 2008 Symposium on Cryptography and Information Security, SCIS 2008, 4A1-3*, January 2008. In Japanese.

18. J.D. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.
19. M. Hell, T. Johansson, and L. Brynielsson. An overview of distinguishing attacks on stream ciphers. *Cryptography and Communications*, 1(1):71–94, 2008.
20. M.E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, IT-26(4):401–406, July 1980.
21. J. Hong and P. Sarkar. New applications of time memory data tradeoffs. In B. Roy, editor, *Advances in Cryptology—ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 353–372. Springer-Verlag, 2005.
22. J. Hong and P. Sarkar. Rediscovery of time memory tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. *http://eprint.iacr.org/2005/090*.
23. K. Ideguchi and D. Watanabe. Method of security evaluation of guess and determine attacks. In *The 2008 Symposium on Cryptography and Information Security, SCIS 2008, 3A1-4*, January 2008. In Japanese.
24. J. Kitahara and D. Watanabe. An electrical power evaluation of stream cipher enocoro which is implemented in hardware. In *The 2008 Symposium on Cryptography and Information Security, SCIS 2008, 2C2-3*, January 2008. In Japanese.
25. K. Konosu, K. Muto, H. Furuichi, D. Watanabe, and T. Kaneko. Evaluation of enocoro-128 ver.1.1 against resynchronization attack. IEICE Technical Report ISEC2007-147, 2008. In Japanese.
26. K. Muto, D. Watanabe, and T. Kaneko. A study on strength of pseudorandom number generator enocoro-80 against linear distinguish attack. In *SITA2007*, 2007. In Japanese.
27. K. Muto, D. Watanabe, and T. Kaneko. Security evaluation of enocoro-80 against linear resynchronization attack. In *The 2008 Symposium on Cryptography and Information Security, SCIS 2008, 4A1-2*, January 2008. In Japanese.
28. K. Muto, D. Watanabe, and T. Kaneko. Strength evaluation of enocoro-128 against lda and its improvement. In *The 2008 Symposium on Cryptography and Information Security, SCIS 2008, 4A1-1*, January 2008. In Japanese.
29. NIST. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22b, 2010.
30. K. Okamoto, K. Muto, and T. Kaneko. Security evaluation of pseudorandom number generator enocoro-80 against differential/linear cryptanalysis (ii). In *The 2009 Symposium on Cryptography and Information Security, SCIS 2009, 4B2-3*, 2009. In Japanese.
31. H. Raddum and I. Semaev. Solving multiple right hand sides linear equations. *Designs, Codes and Cryptography*, 49:147–160, 2008.
32. M.-J. O. Saarinen. Chosen-IV statistical attacks on eSTREAM stream ciphers. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/013, 2006. *http://www.ecrypt.eu.org/stream*.
33. P. Stankovski. Greedy distinguishers and nonrandomness detectors. In G. Gong and K. C. Gupta, editors, *Progress in Cryptology—INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 210–226. Springer-Verlag, 2010.
34. M. Vielhaber. Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Available at *http://eprint.iacr.org/2007/413*, 2007.
35. D. Watanabe and T. Kaneko. A construction of light weight panamalike keystream generator. IEICE Technical report ISEC2007-78, 2007. In Japanese.
36. D. Watanabe, K. Okamoto, and T. Kaneko. A hardware-oriented light weight pseudo-random number generator enocoro-128v2). In *The 2010 Symposium on Cryptography and Information Security, SCIS 2010, 3D1-3*, 2010. In Japanese.

37. D. Watanabe, T. Owada, K. Okamoto, Y. Igarashi, and T. Kaneko. Update on enocoro stream cipher. In *International Symposium on Information Theory and its Applications (ISITA), 2010*, pages 778–783, 2010.

A Tables

Table 6. The bias for the noise N from each approximation of the Sbox. Bias corresponding to an approxiamtion with the polynomial 0x00 - 0x7F is given here. In order to get a fair value for the bias, and allow for easy comparison between the distributions, the bias of N^{16} is shown.

0x00	No bias	0x20	$2^{-40.65}$	0x40	$2^{-35.08}$	0x60	$2^{-35.92}$
0x01	$2^{-36.79}$	0x21	$2^{-40.65}$	0x41	$2^{-37.70}$	0x61	$2^{-40.64}$
0x02	$2^{-39.61}$	0x22	$2^{-38.64}$	0x42	$2^{-39.57}$	0x62	$2^{-39.59}$
0x03	$2^{-40.65}$	0x23	$2^{-35.08}$	0x43	$2^{-41.43}$	0x63	$2^{-35.08}$
0x04	$2^{-40.58}$	0x24	$2^{-38.45}$	0x44	$2^{-39.62}$	0x64	$2^{-41.72}$
0x05	$2^{-39.62}$	0x25	$2^{-37.70}$	0x45	$2^{-40.65}$	0x65	$2^{-41.72}$
0x06	$2^{-40.65}$	0x26	$2^{-38.59}$	0x46	$2^{-38.64}$	0x66	$2^{-40.65}$
0x07	$2^{-39.62}$	0x27	$2^{-31.29}$	0x47	$2^{-38.64}$	0x67	$2^{-38.32}$
0x08	$2^{-35.92}$	0x28	$2^{-40.06}$	0x48	$2^{-39.62}$	0x68	$2^{-39.62}$
0x09	$2^{-35.08}$	0x29	$2^{-39.62}$	0x49	$2^{-36.79}$	0x69	$2^{-38.64}$
0x0A	$2^{-39.62}$	0x2A	$2^{-35.08}$	0x4A	$2^{-32.73}$	0x6A	$2^{-39.62}$
0x0B	$2^{-40.65}$	0x2B	$2^{-39.25}$	0x4B	$2^{-38.64}$	0x6B	$2^{-41.72}$
0x0C	$2^{-38.64}$	0x2C	$2^{-38.64}$	0x4C	$2^{-38.64}$	0x6C	$2^{-31.29}$
0x0D	$2^{-37.70}$	0x2D	$2^{-41.72}$	0x4D	$2^{-40.65}$	0x6D	$2^{-35.08}$
0x0E	$2^{-39.62}$	0x2E	$2^{-39.59}$	0x4E	$2^{-40.06}$	0x6E	$2^{-41.30}$
0x0F	$2^{-37.70}$	0x2F	$2^{-36.79}$	0x4F	$2^{-36.79}$	0x6F	$2^{-37.68}$
0x10	$2^{-36.74}$	0x30	$2^{-37.70}$	0x50	$2^{-39.62}$	0x70	$2^{-37.70}$
0x11	$2^{-32.73}$	0x31	$2^{-35.92}$	0x51	$2^{-38.64}$	0x71	$2^{-36.79}$
0x12	$2^{-39.62}$	0x32	$2^{-39.62}$	0x52	$2^{-34.27}$	0x72	$2^{-39.96}$
0x13	$2^{-39.62}$	0x33	$2^{-35.08}$	0x53	$2^{-41.40}$	0x73	$2^{-37.70}$
0x14	$2^{-37.70}$	0x34	$2^{-37.70}$	0x54	$2^{-42.85}$	0x74	$2^{-38.39}$
0x15	$2^{-44.03}$	0x35	$2^{-36.79}$	0x55	$2^{-38.64}$	0x75	$2^{-37.70}$
0x16	$2^{-38.64}$	0x36	$2^{-38.64}$	0x56	$2^{-41.72}$	0x76	$2^{-41.72}$
0x17	$2^{-38.64}$	0x37	$2^{-33.49}$	0x57	$2^{-36.79}$	0x77	$2^{-35.92}$
0x18	$2^{-37.70}$	0x38	$2^{-35.08}$	0x58	$2^{-40.65}$	0x78	$2^{-30.60}$
0x19	$2^{-41.72}$	0x39	$2^{-40.65}$	0x59	$2^{-35.08}$	0x79	$2^{-35.08}$
0x1A	$2^{-37.70}$	0x3A	$2^{-37.70}$	0x5A	$2^{-40.43}$	0x7A	$2^{-37.70}$
0x1B	$2^{-35.92}$	0x3B	$2^{-39.62}$	0x5B	$2^{-38.64}$	0x7B	$2^{-36.79}$
0x1C	$2^{-38.64}$	0x3C	$2^{-39.62}$	0x5C	$2^{-37.70}$	0x7C	$2^{-35.92}$
0x1D	$2^{-41.24}$	0x3D	$2^{-38.64}$	0x5D	$2^{-40.65}$	0x7D	$2^{-38.64}$
0x1E	$2^{-40.65}$	0x3E	$2^{-40.50}$	0x5E	$2^{-38.47}$	0x7E	$2^{-36.79}$
0x1F	$2^{-40.35}$	0x3F	$2^{-36.79}$	0x5F	$2^{-36.79}$	0x7F	$2^{-40.63}$

Table 7. The bias for the noise N from each approximation of the Sbox. Bias corresponding to an approxiamtion with the polynomial 0x80 - 0xFF is given here. In order to get a fair value for the bias, and allow for easy comparison between the distributions, the bias of N^{16} is shown.

0x80	$2^{-39.62}$	0xA0	$2^{-42.85}$	0xC0	$2^{-36.79}$	0xE0	$2^{-40.65}$
0x81	$2^{-40.65}$	0xA1	$2^{-39.62}$	0xC1	$2^{-37.70}$	0xE1	$2^{-41.14}$
0x82	$2^{-39.62}$	0xA2	$2^{-36.79}$	0xC2	$2^{-37.70}$	0xE2	$2^{-40.50}$
0x83	$2^{-39.04}$	0xA3	$2^{-41.64}$	0xC3	$2^{-37.70}$	0xE3	$2^{-38.55}$
0x84	$2^{-35.08}$	0xA4	$2^{-38.32}$	0xC4	$2^{-40.65}$	0xE4	$2^{-37.59}$
0x85	$2^{-36.79}$	0xA5	$2^{-35.08}$	0xC5	$2^{-34.27}$	0xE5	$2^{-39.55}$
0x86	$2^{-39.55}$	0xA6	$2^{-34.27}$	0xC6	$2^{-36.79}$	0xE6	$2^{-38.64}$
0x87	$2^{-41.62}$	0xA7	$2^{-37.70}$	0xC7	$2^{-35.92}$	0xE7	$2^{-39.62}$
0x88	$2^{-37.70}$	0xA8	$2^{-37.70}$	0xC8	$2^{-41.59}$	0xE8	$2^{-37.70}$
0x89	$2^{-33.49}$	0xA9	$2^{-38.56}$	0xC9	$2^{-36.79}$	0xE9	$2^{-35.08}$
0x8A	$2^{-36.79}$	0xAA	$2^{-33.49}$	0xCA	$2^{-39.62}$	0xEA	$2^{-35.92}$
0x8B	$2^{-34.27}$	0xAB	$2^{-35.08}$	0xCB	$2^{-33.49}$	0xEB	$2^{-39.31}$
0x8C	$2^{-38.55}$	0xAC	$2^{-35.92}$	0xCC	$2^{-37.70}$	0xEC	$2^{-40.63}$
0x8D	$2^{-40.65}$	0xAD	$2^{-40.65}$	0xCD	$2^{-39.62}$	0xED	$2^{-37.70}$
0x8E	$2^{-40.65}$	0xAE	$2^{-38.64}$	0xCE	$2^{-36.69}$	0xEE	$2^{-40.65}$
0x8F	$2^{-37.70}$	0xAF	$2^{-40.64}$	0xCF	$2^{-39.62}$	0xEF	$2^{-36.79}$
0x90	$2^{-37.70}$	0xB0	$2^{-33.49}$	0xD0	$2^{-36.79}$	0xF0	$2^{-40.63}$
0x91	$2^{-38.64}$	0xB1	$2^{-39.46}$	0xD1	$2^{-38.64}$	0xF1	$2^{-42.32}$
0x92	$2^{-38.64}$	0xB2	$2^{-35.92}$	0xD2	$2^{-35.08}$	0xF2	$2^{-38.64}$
0x93	$2^{-38.64}$	0xB3	$2^{-37.70}$	0xD3	$2^{-38.64}$	0xF3	$2^{-40.61}$
0x94	$2^{-35.92}$	0xB4	$2^{-36.79}$	0xD4	$2^{-42.71}$	0xF4	$2^{-39.51}$
0x95	$2^{-41.14}$	0xB5	$2^{-37.70}$	0xD5	$2^{-41.58}$	0xF5	$2^{-41.71}$
0x96	$2^{-41.72}$	0xB6	$2^{-39.62}$	0xD6	$2^{-41.72}$	0xF6	$2^{-35.92}$
0x97	$2^{-35.08}$	0xB7	$2^{-42.71}$	0xD7	$2^{-37.70}$	0xF7	$2^{-37.68}$
0x98	$2^{-39.62}$	0xB8	$2^{-38.64}$	0xD8	$2^{-40.65}$	0xF8	$2^{-39.59}$
0x99	$2^{-36.79}$	0xB9	$2^{-37.70}$	0xD9	$2^{-41.72}$	0xF9	$2^{-39.31}$
0x9A	$2^{-38.64}$	0xBA	$2^{-36.79}$	0xDA	$2^{-39.62}$	0xFA	$2^{-40.65}$
0x9B	$2^{-35.92}$	0xBB	$2^{-36.79}$	0xDB	$2^{-34.27}$	0xFB	$2^{-41.67}$
0x9C	$2^{-39.46}$	0xBC	$2^{-39.62}$	0xDC	$2^{-40.57}$	0xFC	$2^{-42.67}$
0x9D	$2^{-42.85}$	0xBD	$2^{-42.60}$	0xDD	$2^{-35.92}$	0xFD	$2^{-39.45}$
0x9E	$2^{-34.88}$	0xBE	$2^{-37.70}$	0xDE	$2^{-42.80}$	0xFE	$2^{-35.08}$
0x9F	$2^{-35.92}$	0xBF	$2^{-35.08}$	0xDF	$2^{-35.92}$	0xFF	$2^{-35.92}$