Improved Greedy Nonrandomness Detectors for Stream Ciphers

Linus Karlsson, Martin Hell and Paul Stankovski
Department of Electrical and Information Technology, Lund University, P.O. Box 118, 221 00, Lund, Sweden
{linus.karlsson, martin.hell, paul.stankovski}@eit.lth.se

Keywords: Maximum Degree Monomial, Distinguisher, Nonrandomness Detector, Grain-128a, Grain-128.

Abstract: We consider the problem of designing distinguishers and nonrandomness detectors for stream ciphers using the maximum degree monomial test. We construct an improved algorithm to determine the subset of key and IV-bits used in the test. The algorithm is generic, and can be applied to any stream cipher. In addition to this, the algorithm is highly tweakable, and can be adapted depending on the desired computational complexity. We test the algorithm on the stream ciphers Grain-128a and Grain-128, and achieve significantly better results compared to an earlier greedy approach.

1 INTRODUCTION

Modern stream ciphers take as input a secret key and a public IV. These values are typically loaded into the state as part of the initialization. Before generating keystream, the key and IV are mixed such that each state bit depends on a large number of key and IV bits. This is accomplished using a set of initialization rounds. During the initialization, the cipher does not produce any output. These initialization rounds are used to prevent attacks in which an attacker can use knowledge of the IV in order to deduce information about the keystream.

When designing a stream cipher, care must be taken when choosing the number of initialization rounds. Too many, and the cipher will have poor initialization performance, too few and an attacker may be able to perform an attack, e.g., a chosen-IV attack.

A part of the field of cryptanalysis concerns the construction of distinguishers and nonrandomness detectors. The goal of both devices is to determine if the input comes from a cipher, or is just random data. For a good cipher, such a device should not be possible to construct, since the output from a cipher should appear random.

The difference between distinguishers and nonrandomness detectors is the amount of control the attacker has. In a distinguisher, the key is fixed and unknown to the attacker, thus only the IV bits are available to modify. In a nonrandomness detector, both key and IV bits are modifiable by the attacker.

Previous work such as (Englund et al., 2007) has considered the design of distinguishers and nonrandomness detectors by using a test called the Maximum Degree Monomial (MDM) test. This test looks at statistical properties of a cipher, and tries to find weaknesses.

The MDM test requires choosing a suitable subset of the cipher's IV bits. The choice of this subset is crucial, and a greedy algorithm for finding a good subset was proposed in (Stankovski, 2010).

We build upon the previous work and propose an improved, generalized, algorithm which outperforms the greedy algorithm in finding suitable subsets. We also implement and test our algorithm and present new results on the stream ciphers Grain-128 and Grain-128a.

The paper is organized as follows. Section 2 presents the necessary background, while Section 3 describes our improved algorithm. In Section 4 the results are presented, followed by a discussion of related work in Section 5. Section 6 concludes the paper.

2 BACKGROUND

The maximum degree monomial test was presented in (Englund et al., 2007) and described a clean way to detect nonrandomness by looking at the cipher output.

Considering an arbitrary stream cipher, we can consider it as a black box with two inputs, and one output. The input is the key K and the initialization vector (IV) V respectively. We consider the concatenation of the key K and the IV V as a boolean space B of dimension $b = |K| + |V|$.

We are interested in finding out whether or not the

Karlsson, L., Hell, M. and Stankovski, P.
Improved Greedy Nonrandomness Detectors for Stream Ciphers.
DOI: 10.5220/0006268202250232
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 225-232
ISBN: 978-989-758-209-7

maximum degree monomial exists in the ANF of the boolean function of the first keystream bit. According to the Reed-Muller transform, the coefficient of the maximum degree monomial can be found simply by XORing all the entries in the truth table of a boolean function as

$$\bigoplus_{\boldsymbol{x}\in\{0,1\}^b} z_0(\boldsymbol{x})$$

where $z_0(\boldsymbol{x})$ is the first output bit of the stream cipher initialized with the values in $\boldsymbol{x}$. Thus we generate all possible values for the input set, and for each value perform an initialization of the cipher to get the first keystream bit.

An interesting use case for this test is in analyzing the required amount of initialization rounds of a stream cipher. Typically, a stream cipher designer has to choose a suitable count of these rounds: too few, and we may have attacks on the cipher, too many, and the performance hit will be large.

The idea above can be extended such that we consider a modified version of the cipher, where we also look at the cipher output during its initialization rounds (normally this output is suppressed). Assuming a cipher with l initialization rounds, we denote the ith initialization round output function as $f_i(\boldsymbol{x})$, thus giving us a vector

$$f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_l(\boldsymbol{x})\,.$$

Thus, instead of only looking at the ANF and finding the maximum degree of a single function (z_0 before), we now look at l different boolean functions, and for each of the functions, we find the coefficient of the maximum degree monomial. We call this sequence of coefficients the *maximum degree monomial signature*, or MDM signature, following the terminology in (Stankovski, 2010).

The keystream produced by an ideal stream cipher should be indistinguishable from a random stream of bits to an observer. This means that if we look at each output bit function $f_i(\boldsymbol{x})$, it should appear to be a random function $f_i : B \to \{0,1\}$. Consequentially, for a random function, we expect the maximum degree monomial to exist with probability $\frac{1}{2}$. Therefore, we expect the coefficients 0 and 1 to appear with equal probability, and we expect to see a random-looking MDM signature for an ideal cipher.

However, if the input space B is large, clearly the construction of a MDM signature will result in too many initialization of the cipher to be practically usable. Therefore, we can only consider a subset S of the input space B. The remaining part, $B \setminus S$, is set to some constant value, typically either all zero or all one.

2.1 Finding the Subset S

Choosing different subsets S will give different MDM signatures. For example, looking at the stream cipher Grain-128a (Ågren et al., 2011) and the subset S consisting of key bit 23, and IV bits 47, 53, 58, 64, we get the following MDM signature:

$$\underbrace{000\ldots000}_{\text{187 zeros}}111\ldots$$

Clearly, the start of this MDM signature, with its 187 consequent zeros, does not appear to be random. However, after the initial 187 zeros, the sequence start to appear more random-like. We can interpret the result above as that it is not enough with 187 initialization rounds. However, Grain-128a has 256 rounds in total, and thus we state that *we find nonrandomness in 187 out of 256 initialization rounds*. We note that we get a nonrandomness result, since we include both key and IV bit in the input subset S.

It should now be clear that we wish to maximize the number of zeros we can find, with the ultimate goal being to find a nonrandomness in *all* initialization rounds, thus spilling over to the actual keystream of an unmodified cipher.

The subset S plays a crucial role here. Its composition affects the resulting MDM signature to a great extent. Consider the two examples for Grain-128a found in Table 1.

Table 1: The number of initial zeros in the MDM signature for two different subsets S for Grain-128a.

K	IV	rounds out of 256
$\{\}$	$\{1,2,3,4,5\}$	107
$\{23\}$	$\{47,53,58,64\}$	187

As we see, the choice of the input set S is significant. In the above case, where $|S| = 5$, i.e. we have chosen five key and/or IV bits, the second row is actually the optimal result. However, calculating the optimal result is infeasible as S grows to a larger set. For Grain-128a, which has 96 IV bits and 128 key bits, we will have to test $\binom{224}{|S|}$ combinations.

2.2 Greedy Approach

In (Stankovski, 2010) the author proposed a greedy algorithm to find such a subset. The suggested approach can concisely be described with the following steps:

1. Find an optimal starting bitset of a small size (possibly also zero, making this step optional)

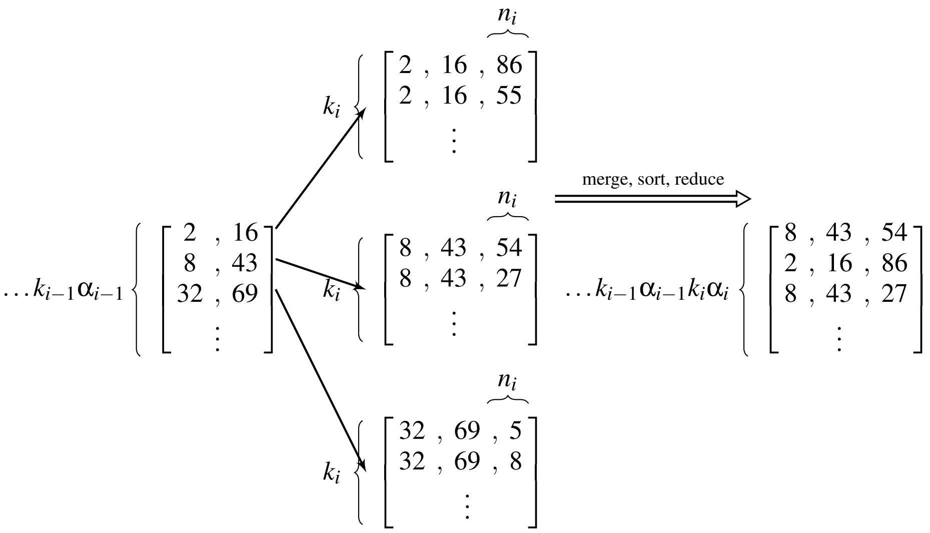

Figure 1: One step of our improved algorithm.

2. Add the n bits which together produce the highest number of zero rounds to the current bitset.
3. Repeat step 2 until a bitset of the wanted size m is found.

Consider an example where we start with the optimal bitset described earlier. A few steps of the greedy algorithm, with $n = 1$, would then look like this:

i_0 :	$K = \{23\}$	$IV = \{47,53,58,64\}$
i_1 :	$K = \{23\}$	$IV = \{47,53,58,64,12\}$
i_2 :	$K = \{23,72\}$	$IV = \{47,53,58,64,12\}$
i_3 :	$K = \{23,72,31\}$	$IV = \{47,53,58,64,12\}$
i_4 :	$K = \{23,72,31,107\}$	$IV = \{47,53,58,64,12\}$

As we can see, the algorithm will, for each iteration, add the best bit, i.e. the one that gives the largest amount of zeros in the MDM signature.

The issue with this greedy algorithm is the same as for greedy algorithms in general—they may fail to find the global optimum, and instead get stuck in local optima.

3 IMPROVED ALGORITHM

We build upon the greedy algorithm presented in Section 2.2, but instead present a more general solution which can achieve better results.

One efficiency problem with a purely greedy algorithm is the risk that it gets stuck in a local optimum. This optimum may or may not be close to the global optimum. Our core idea is to extend the naïve greedy algorithm by examining more possible paths. Instead of only looking at the single best candidate in each step, we store and explore several different candidates. The idea is that the second best candidate in one step may very well be the better one at the next stage of the algorithm.

Examining more possible bit sets will, of course, increase the computational complexity of the algorithm. However, the computational complexity can be calculated, and we can derive an expression for the total computational complexity, which can be used to estimate the time required.

Our algorithm is described in Algorithm 1 and Algorithm 2. The algorithm is parametrized by three different parameter vectors: $\boldsymbol{\alpha}$, $\boldsymbol{k}$, and $\boldsymbol{n}$. A graphical presentation of one iteration of the algorithm is given in Figure 1, which serve as the basis for the description of the algorithm below:

1. Consider a set of candidates from a previous iteration, or from an optimal starting set.
2. For each candidate in the list, we add the k_i best n_i new bits and store them in a new list. Note that we now have one such new list for each candidate in the original list.
3. Merge all lists, sorting by the number of zero rounds. This gives a list of $\ldots k_{i-1}\alpha_{i-1}k_i$ items.
4. Finally, reduce the size of this list with the factor α_i ($0 < \alpha_i \leq 1.0$), limiting the size of the combined list to $\ldots k_{i-1}\alpha_{i-1}k_i\alpha_i$ items.
5. Repeat from step 1 until a bitset of the wanted size has been found.

We note that the previous greedy algorithm from Section 2.2 corresponds to specific values of these parameter vectors, namely $\boldsymbol{\alpha} = [1.0, 1.0, \ldots]$), $\boldsymbol{k} =$

$[1,1,\ldots]$, and $\boldsymbol{n} = [n,n,\ldots]$. Thus our improved algorithm is a generalization of the previous algorithm.

Algorithm 1: SlightlyGreedy.

```
Input: key $K$, IV $V$, bit space $B$, maximum bitset size $m$, vector $\boldsymbol{k}$, vector $\boldsymbol{n}$, vector $\boldsymbol{\alpha}$
Output: bitset $S$ of size $m$.

$S_0 = \{\emptyset\}$
/* The set $S_0$ contains a single empty bitset */
for (each $i \in \{0,\ldots,m-1\}$) {
    for (each $c \in S_i$) {
        $L_c$ = FindBest($K,V,B,c,k_i,n_i$);
    }
    $S_{i+1}$ = concatenate(all $L_c$ from above);
    sort $S_{i+1}$;
    reduce # of elements in $S_{i+1}$ by a factor $\alpha_i$;
}
return $S_m$;
```

Algorithm 2: FindBest.

```
Input: key $K$, IV $V$, bit space $B$, current bitset $c$, number of best bitsets to retain $k$, bits to add $n$
Output: $k$ bitsets each of size $|c|+n$.

/* let $\binom{S}{k}$ denote the set of all $k$-combinations of a set $S$. */

$S = \emptyset$;
for (each $n$-tuple $\{b_1,\ldots,b_n\} \in \binom{B\setminus c}{n}$) {
    $z$ = number of initial zeros using bit set
        $c \cup \{b_1,\ldots,b_n\}$;
    if ($z$ is among the $k$ highest values) {
        add $c \cup \{b_1,\ldots,b_n\}$ to $S$;
        reduce $S$ to $k$ elements by removing
            element with lowest $z$;
    }
}
return $S$;
```

3.1 Computational Cost

The computational cost C depends on the input parameter vectors according to the following function, where c is the number of iterations required ($c = |\boldsymbol{k}| = |\boldsymbol{n}| = |\boldsymbol{\alpha}|$), b is the bit space size $b = |B|$.

$$C(b,c,\boldsymbol{k},\boldsymbol{n},\boldsymbol{\alpha}) = \sum_{i=0}^{c-1}\left[2^{\sum_{j=0}^{i} n_j}\binom{b-\sum_{j=0}^{i-1} n_j}{n_i}\prod_{j=0}^{i-1} k_j\alpha_j\right] \qquad (1)$$

The formula can be derived using combinatorics. In the formula, the power of two is related to the size of the bitset—a large bitset requires more calculations. The binomial coefficient is the number of possible bitsets we can form given the current iteration's n_i. Finally, the final product is needed because we reduce the number of candidates in each iteration using the factors in $\boldsymbol{\alpha}$. Clearly, in practice, the actual running time is also dependent on other factors, such as the cipher we are running the algorithm on.

Note that for the naïve greedy algorithm, where we have a constant n, and $\boldsymbol{k}$ and $\boldsymbol{\alpha}$ are both all ones, we get the following simplified expression for the complexity:

$$C(b,c,n) = \sum_{i=0}^{c-1}\left[2^{n(i+1)}\binom{b-n\cdot i}{n}\right] \qquad (2)$$

4 RESULTS

Finding the correct vectors $\boldsymbol{k}$, $\boldsymbol{n}$, and $\boldsymbol{\alpha}$ is crucial to achieving good performance of our algorithm. While the original greedy approach only had one degree of freedom (the constant n), our generalized algorithm has many more. We have performed a significant amount of simulations to present results on how the choice of parameters affect the final result. The tests have been performed mainly on the cipher Grain-128a, but also on Grain-128 (Hell et al., 2006). The exact parameters used for each result presented below are available in the Appendix.

4.1 Tuning the Greediness

We start by varying $\boldsymbol{k}$ and $\boldsymbol{\alpha}$. We keep an identical amount of candidates in each iteration, but vary $\boldsymbol{k}$ and $\boldsymbol{\alpha}$. Recall that k_i govern how many new candidates we generate from a previous iteration's bitset. A high k_i and low α_i means that we may end up with several candidates that have the same "stem", i.e. they have the same origin list. If we lower k_i and instead increase α_i we will get a greater mix of different stems, while still maintaining the same amount of candidates for the iteration—in a sense we lower the greediness of the algorithm. Thus, we want to test some different tradeoffs between the two parameters. In the results below, we name the different test cases as a percentage value of the total amount of candidates for each round. As an example, if the total number of candidates in a given round is 1000, we could select a k_i of 200, and a corresponding α_i of 0.005, which gives us 1000 candidates for the next round as well. We call this particular case *20 %-k* since k_i is 20 % of the candidates for the round.

We have tried several combinations of $\boldsymbol{k}$ and $\boldsymbol{\alpha}$ as can be seen in the plot in Figure 2. We also include the greedy algorithm as a reference. Note that the greedy algorithm will due to its simplistic nature have a lower computational complexity since it only keeps one candidate in each iteration. To be able to compare the results based on computational complexity, we have plotted the graph based on logarithmic complexity rather than bit set size. The complexity is calculated using Equation 1, and we then take the (natural) logarithm of the this value, so that we can produce a reasonably scaled plot. This graph can be seen in Figure 3. The maximum rounds for each case is also available in Table 2.

From the results we note that a too low k seems to lower the efficiency of the algorithm. The reason for this is probably that the too low k forces the algorithm to choose candidates from lists with lower amounts of rounds, which are then not useful in the following iterations.

Table 2: Results when varying $\boldsymbol{k}$ and $\boldsymbol{\alpha}$.

Test case	Maximum rounds
Greedy	187
20 %-k	203
0.5 %-k	198
0.2 %-k	192
min-k %-k	190

4.2 Varying the Number of Bits Added in Each Iteration

We have also tried modifying n in a number of different combinations. The rationale behind this is that we expect that a higher n will yield better results, since that also reduces the risk to get stuck in a local optima. However, having a large, constant, n throughout all iteration as in (Stankovski, 2010) means that the later iterations will be very computationally demanding. Therefore, we test three different variants where the $\boldsymbol{n}$-vector contains decreasing values of n_i. As a base line, we present the naive greedy approach with a constant n of 1, 2, and 3 respectively.

We note that the computational complexity will vary significantly depending on the choice of the vector $\boldsymbol{n}$. We first plot the graph with the x-axis as the bit set size, as can be seen i Figure 4. We can also look at the graph plotted with the computational complexity, calculated as in Equation 1. This allows us to compare the results achieved for a specific computational complexity. This is done in Figure 5. The maximum rounds for each case are also available in Table 3.

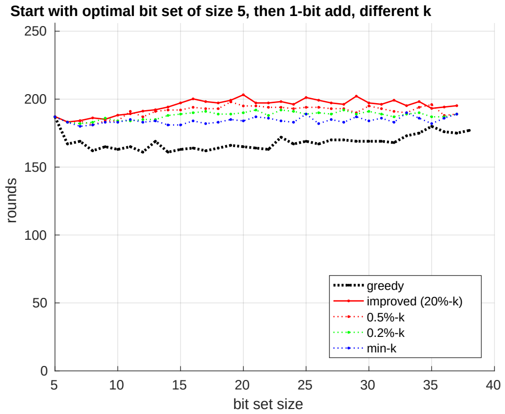

Figure 2: Varying k and α, with $n_i = 1$. Thick dotted black line is the greedy baseline.

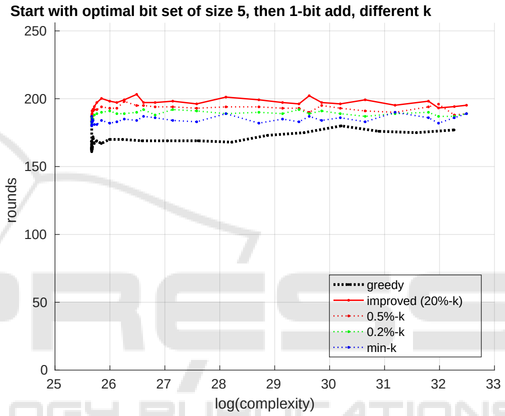

Figure 3: Varying k and α, with $n_i = 1$. Thick dotted black line is the greedy baseline. The x-axis scaled according to logarithmic computational complexity.

From the results we note that regardless of our choice of $\boldsymbol{n}$, our algorithm outperforms the greedy variants. We also see that a higher n_i in the initial iterations seem to lead to better results which remain as the algorithm proceeds towards larger bitsets.

Table 3: Results when varying $\boldsymbol{n}$.

Test case	Maximum rounds
Greedy 1-bit	187
Greedy 2-bit	187
Greedy 3-bit	187
2-2-2-2-2-2-2-2-1-...	203
2-2-2-2-1-...	199
1-...	195

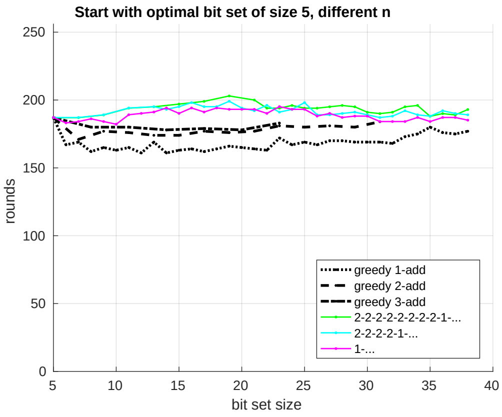

Figure 4: Varying n. Thick black lines are the greedy baselines for n equal to 1, 2, and 3.

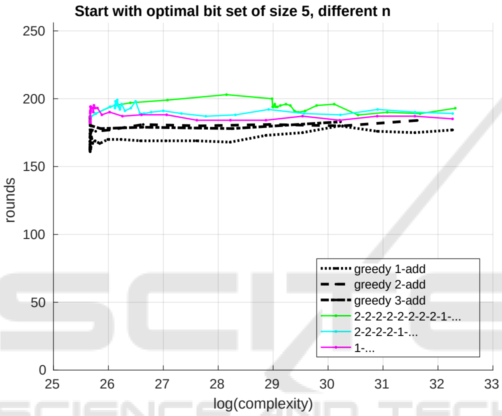

Figure 5: Varying n. Thick black lines are the greedy baselines for n equal to 1, 2, and 3. The x-axis scaled according to logarithmic computational complexity.

4.3 Results on Grain-128

In addition to the results above on Grain-128a, we also tested our algorithm on Grain-128. In (Stankovski, 2010), a full-round (256 out of 256 initialization rounds) result was presented using a bit-set of size 40, using only IV-bits, with an optimal starting bitset of size 6. This was found using a constant $n = 2$. In our generalized algorithm, this corresponds to a parameter set of $\boldsymbol{\alpha} = [1.0, 1.0, 1.0, \ldots])$, $\boldsymbol{k} = [1, 1, 1, \ldots]$, and $\boldsymbol{n} = [6, 2, 2, \ldots]$.

We construct a new set of parameters utilizing our improved algorithm. We utilize the possibility to keep multiple candidates in each step, especially in the beginning where we still have small bitsets. We find a smaller bitset, of size 25, which still gives us a full-round result of 256 out of 256 initialization rounds. Using the complexity formula in Equation 1, we can compare the computational complexity between the two results, and find that our improved algorithm has a complexity which is a factor about 2^{12} lower than the earlier result, while still finding an equal amount of zeros in the MDM signature.

5 RELATED WORK

In (Saarinen, 2006) the author discussed the d-Monomial test, and how it can be applied in chosen-IV attacks against stream ciphers. In contrast to our work, Saarinen considers monomials of various degrees (up to degree d, therefore the name d-Monomial test). Another difference is the choice of bit set, where Saarinen only considers n-consecutive bits either in the beginning or in the end of the IV, rather than choosing freely from all parts of the IV.

In (Englund et al., 2007) the Maximum Degree Monomial (MDM) test was introduced. Instead of looking at different degrees of monomials, the MDM test only focus on the maximum degree. The rationale behind this is that the maximum degree monomial is likely to occur only if all IV bits have been properly mixed. In addition, the existence of the maximum degree monomial is easy to find. The coefficient of the monomial can be found by simply XORing all entries in the truth table.

In the previously mentioned work, a subset of the IV space was used in the tests. In (Stankovski, 2010), a greedy heuristic to find these bitsets was discussed. The algorithm started with an optimal bitset of a small size, and then added n bits in each step in a greedy fashion. In addition, both IV and key bits were suggested for getting distinguisher and nonrandomness results, respectively.

In (Liu et al., 2015), the authors concentrate on small cubes, and instead look at unions of these cubes. Another difference is that they look at sub-maximal degree monomial tests.

Also partly based on Stankovski's work is the work in (Sarkar et al., 2016), where the authors propose two new, alternative heuristics. In the first, called "maximum last zero", the authors not only maximize the initial sequence of zeros, but also ensure that the position of the current iteration in the MDM signature is a zero as well. In their second heuristic, called "maximum frequency of zero", they instead look at the total amount of zeros in the MDM signature. Their heuristics are only applied to Trivium (De Cannière, 2006) and Trivia-SC (Chakraborti et al., 2015). Similar to this paper, they also mention the use of a non-constant n, i.e. a $\boldsymbol{n}$-vector, although the authors do not discuss the reasons for this choice.

In (Vielhaber, 2007) an attack called AIDA on Trivium (with half the amount of initialization rounds) was presented. Related to this attack are the cube attacks (Dinur and Shamir, 2009), and especially the dynamic cube attacks (Dinur and Shamir, 2011) which was used to attack Grain-128.

Attacks on the newer Grain-128a has been discussed in some recent papers as well. In (Banik et al., 2013) the authors present a related-key key attack requiring $> 2^{32}$ related keys, and $> 2^{64}$ chosen IVs, while in (Sarkar et al., 2015) the authors present a differential fault attack against all the three ciphers in the Grain-family.

6 CONCLUSIONS

In this paper we have discussed the use of the maximum degree monomial test to analyze the required amount of initialization rounds of stream ciphers. We have proposed a new algorithm to find a suitable subset of key and IV-bits used in the MDM test. Our algorithm has its root in a greedy approach, but by generalizing, we have designed an algorithm less susceptible to local optima. The algorithm is very flexible, and the parameters can be tuned based on the desired computational complexity. Testing our algorithm on the cipher Grain-128a, with different parameters, we achieve new significantly better results compared to the previous greedy algorithm.

ACKNOWLEDGMENTS

The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at Lunarc.

REFERENCES

Ågren, M., Hell, M., Johansson, T., and Meier, W. (2011). Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59.

Banik, S., Maitra, S., Sarkar, S., and Meltem Sönmez, T. (2013). A chosen IV related key attack on Grain-128a. In *Information Security and Privacy: 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings*, pages 13–26. Springer.

Chakraborti, A., Chattopadhyay, A., Hassan, M., and Nandi, M. (2015). Trivia: A fast and secure authenticated encryption scheme. In *Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 330–353. Springer.

De Cannière, C. (2006). Trivium: A stream cipher construction inspired by block cipher design principles. In *Information Security: 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006. Proceedings*, pages 171–186. Springer.

Dinur, I. and Shamir, A. (2009). Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 278–299. Springer.

Dinur, I. and Shamir, A. (2011). Breaking Grain-128 with dynamic cube attacks. In *Fast Software Encryption: 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 167–187. Springer.

Englund, H., Johansson, T., and Sönmez Turan, M. (2007). A framework for chosen IV statistical analysis of stream ciphers. In *Progress in Cryptology – INDOCRYPT 2007: 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007. Proceedings*, pages 268–281. Springer.

Hell, M., Johansson, T., Maximov, A., and Meier, W. (2006). A stream cipher proposal: Grain-128. In *2006 IEEE International Symposium on Information Theory*, pages 1614–1618.

Liu, M., Lin, D., and Wang, W. (2015). Searching cubes for testing boolean functions and its application to Trivium. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 496–500.

Saarinen, M.-J. O. (2006). Chosen-IV statistical attacks on eSTREAM stream ciphers. http://www.ecrypt.eu.org/stream/papersdir/2006/013.pdf.

Sarkar, S., Banik, S., and Maitra, S. (2015). Differential fault attack against Grain family with very few faults and minimal assumptions. *IEEE Transactions on Computers*, 64(6):1647–1657.

Sarkar, S., Maitra, S., and Baksi, A. (2016). Observing biases in the state: case studies with Trivium and Trivia-SC. *Designs, Codes and Cryptography*.

Stankovski, P. (2010). Greedy distinguishers and nonrandomness detectors. In *INDOCRYPT 2010*, pages 210–226. Springer.

Vielhaber, M. (2007). Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413. http://eprint.iacr.org/2007/413.

APPENDIX

This appendix contains the exact vectors used for the different results discussed in Section 4. The vectors used for the results for varying k and α are given in Table 4. In the same fashion, the vectors used for the results for varying n are presented in Table 5. Finally, the vectors for the results on Grain-128 are given in Table 6.

Table 4: Varying $\boldsymbol{k}$ and $\boldsymbol{\alpha}$.

	Greedy
$\boldsymbol{k}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
	Improved (20 %-k)
$\boldsymbol{k}$	{ 1000, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 100, 60, 60, 20, 20, 20, 20, 20, 20, 12, 6, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.01, $\frac{1}{60}$, $\frac{1}{60}$, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, $\frac{1}{12}$, $\frac{2}{15}$, 0.375, 0.5, 0.5, 0.5, 0.5, $\frac{2}{9}$, 1.0, 0.5, 1.0 }
	0.5 %-k
$\boldsymbol{k}$	{ 1000, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, $\frac{1}{6}$, 0.3, 0.5, $\frac{1}{3}$, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6, 0.5, 0.4, 0.75, 1.0, 1.0, 1.0, 1.0, $\frac{2}{9}$, 1.0, 0.5, 1.0 }
	0.2 %-k
$\boldsymbol{k}$	{ 1000, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 1.0, $\frac{1}{3}$, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6, 0.5, 0.4, 0.75, 1.0, 1.0, 1.0, 1.0, $\frac{2}{9}$, 1.0, 0.5, 1.0 }
	min-k
$\boldsymbol{k}$	{ 1000, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 0.6, 1.0, $\frac{1}{3}$, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6, 0.5, 0.4, 0.75, 1.0, 1.0, 1.0, 1.0, $\frac{2}{9}$, 1.0, 0.5, 1.0 }

Table 5: Varying $\boldsymbol{n}$.

	Greedy 1-add
$\boldsymbol{k}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
	Greedy 2-add
$\boldsymbol{k}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 }
$\boldsymbol{\alpha}$	{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
	Greedy 3-add
$\boldsymbol{k}$	{ 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 3, 3, 3, 3, 3, 3 }
$\boldsymbol{\alpha}$	{ 1, 1, 1, 1, 1, 1, 1 }
	2-2-2-2-2-2-2-2-1-...
$\boldsymbol{k}$	{ 1000, 200, 200, 200, 200, 150, 50, 50, 50, 30, 15, 6, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.005, 0.005, 0.0005, 0.005, $\frac{1}{150}$, 0.02, 0.01, 0.02, 0.02, $\frac{1}{15}$, $\frac{1}{15}$, 0.15, 0.2, 0.2, $\frac{4}{45}$, 0.1, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 }
	2-2-2-2-1-...
$\boldsymbol{k}$	{ 1000, 200, 200, 200, 200, 150, 50, 50, 50, 30, 15, 6, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.005, 0.005, 0.0005, 0.005, $\frac{1}{150}$, 0.02, 0.01, 0.02, 0.02, $\frac{1}{15}$, $\frac{1}{15}$, 0.15, 0.2, 0.2, $\frac{4}{45}$, 0.1, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 }
	1-...
$\boldsymbol{k}$	{ 1000, 200, 200, 200, 200, 150, 50, 50, 50, 30, 15, 6, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{n}$	{ 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.005, 0.005, 0.0005, 0.005, $\frac{1}{150}$, 0.02, 0.01, 0.02, 0.02, $\frac{1}{15}$, $\frac{1}{15}$, 0.15, 0.2, 0.2, $\frac{4}{45}$, 0.1, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 }

Table 6: Results on Grain-128.

	Greedy 1-add
$\boldsymbol{k}$	{ 1000, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 60, 60, 20, 20, 20, 20, 20 }
$\boldsymbol{n}$	{ 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
$\boldsymbol{\alpha}$	{ 1.0, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.0125, 0.00625, 0.01, $\frac{1}{60}$, $\frac{1}{60}$, 0.05, 0.05, 0.05, 0.05 }