Breaking Grain-128 with Dynamic Cube Attacks

Itai Dinur and Adi Shamir

Computer Science department
The Weizmann Institute
Rehovot 76100, Israel

Abstract. We present a new variant of cube attacks called a *dynamic cube attack*. Whereas standard cube attacks [4] find the key by solving a system of linear equations in the key bits, the new attack recovers the secret key by exploiting distinguishers obtained from cube testers. Dynamic cube attacks can create lower degree representations of the given cipher, which makes it possible to attack schemes that resist all previously known attacks. In this paper we concentrate on the well-known stream cipher Grain-128 [6], on which the best known key recovery attack [15] can recover only 2 key bits when the number of initialization rounds is decreased from 256 to 213. Our first attack runs in practical time complexity and recovers the full 128-bit key when the number of initialization rounds in Grain-128 is reduced to 207. Our second attack breaks a Grain-128 variant with 250 initialization rounds and is faster than exhaustive search by a factor of about 2^{28}. Finally, we present an attack on the full version of Grain-128 which can recover the full key but only when it belongs to a large subset of 2^{-10} of the possible keys. This attack is faster than exhaustive search over the 2^{118} possible keys by a factor of about 2^{15}. All of our key recovery attacks are the best known so far, and their correctness was experimentally verified rather than extrapolated from smaller variants of the cipher. This is the first time that a cube attack was shown to be effective against the full version of a well known cipher which resisted all previous attacks.

Keywords: Cryptanalysis, stream ciphers, Grain-128, cube attacks, cube testers, dynamic cube attacks.

1 Introduction

A well designed cipher is expected to resist all known cryptanalytic attacks, including distinguishing attacks and key recovery attacks. These two types of attacks are closely related since in many cases a distinguisher can be extended to a key recovery attack. Examples include many of the key-recovery attacks on iterated block ciphers such as differential cryptanalysis [1] and linear cryptanalysis [2]: First, the attacker constructs a distinguisher for a certain number of rounds of the iterated block cipher (usually one round less than the total number of rounds). Then, the attacker guesses part of the secret key and uses it to partially decrypt several ciphertexts in the final round. The distinguisher

A. Joux (Ed.): FSE 2011, LNCS 6733, pp. 167–187, 2011.

can be easily exploited to verify the guess: Under the correct guess, the partially decrypted ciphertexts are expected to exhibit the non-random property of the distinguisher. On the other hand, an incorrect guess is actually equivalent to adding another encryption round, and hence ciphertexts decrypted with an incorrect guess are expected to behave randomly. This is a very general technique for exploiting a distinguisher to recover the secret key of block ciphers, but it cannot be typically applied to stream ciphers, where partial decryption is not possible. Moreover, even when dealing with iterated block ciphers, more efficient key-recovery techniques often exist. In this paper we focus on the specific case of distinguishers obtained from cube testers (see [3]) and show how to use them in key recovery attacks.

Cube testers [3] are a family of generic distinguishers that can be applied to the black box representation of any cryptosystem. Cube attacks [4] are related to cube testers since both types of attacks sum the output of a cryptographic function over a subset of its input values. However, cube testers use the resultant sums to distinguish the cipher from a random function, whereas cube attacks use the sums to derive linear equations in the secret key bits. The success of cube testers and cube attacks on a given cryptosystem depends on subtle properties of the ANF (algebraic normal form) representation of the output function in the plaintext and key bits over GF(2). Although the explicit ANF representation is usually unknown to the attacker, cube testers and cube attacks can exploit a relatively low degree or sparse ANF representation in terms of some of its variables to distinguish the cipher from a random function and to recover the secret key.

Both cube attacks and cube testers are performed in two phases: The preprocessing phase which is not dependent on the key, and the online phase in which the key has a fixed unknown value. Whereas cube attacks are key recovery attacks and are thus stronger than cube testers, the preprocessing phase of cube attacks is generally more complex and has a lower chance of succeeding than the preprocessing phase of cube testers. The reason for this is that cube attacks require that the sum of the cipher's output function has a very specific property - it needs to be of low degree when represented as a polynomial in the key bits. Cube testers do not require such a specific property, but rather require that the value of the sum exhibits a property which is easily testable. An example of such a property is balance (i.e. whether the sum (modulo 2) is 0 and 1 with equal probabilities). Examples where cube testers succeed, while cube attacks seem to fail include scaled-down variants of the stream cipher Grain-128 (see [5]). Even in the case of scaled-down variants of the stream cipher Trivium, where cube attacks succeed ([4]), the preprocessing phase of cube attacks is much more time consuming than the one of cube testers. The challenge that we deal with in this paper is to extend cube testers to key recovery attacks in a new generic way. This combines the key recovery feature of cube attacks with the relatively low computational complexity of the preprocessing phase of cube testers.

We present a new attack called a dynamic cube attack that recovers the secret key of a cryptosystem by exploiting distinguishers given by cube testers.

The main observation that we use for the new attack is that when the inputs of the cryptosystem are not mixed thoroughly enough, the resistance of such a marginal cipher to cube testers usually depends on very few (or even one) non-linear operations that are performed at the latest stages of the encryption process. These few non-linear operations produce most of the relatively high degree terms in the ANF representation of the output function. If we manage to simplify the ANF representation of the intermediate encryption state bits that are involved in these non-linear operations (e.g. by forcing one of the two inputs of a multiplication operation to be zero), then the degree of the polynomial will be much lower, making it much more vulnerable to cube testers. In dynamic cube attacks, we analyze the cipher, find these crucial state bits and force them to be zero by using dedicated input bits called dynamic variables. Since the values of the state bits typically depend also on some key bits, we have to either guess them, or to assume that they have a particular value in order to apply the attack. For each guess, we use a different cube tester (that assigns the dynamic variables according to the guess) to distinguish the cipher from random. For the correct guess, the ANF representation of the crucial intermediate encryption state bits is simplified to zero, and the cube tester is likely to detect a strong non-random property in the output. On the other hand, for a large portion of wrong guesses, the cube tester is unlikely to detect this property. Thus, we can efficiently eliminate wrong guesses and thus recover parts of the secret key.

We applied the attack to two reduced variants of the stream cipher Grain-128 [6], and obtained the best known key recovery results for these variants. More significantly, we present an attack on the full version of Grain-128 which is faster than exhaustive search by a factor of 2^{15}, for a subset of 2^{-10} of all the possible keys. The attack can probably be optimized to break a larger set of weak keys of Grain-128, but even in its current form, it can break a practically significant fraction of almost one in a thousand keys. This is much better than other weak key attacks, which can typically break only a negligible fraction of keys.

The idea of assigning dynamic constraints (or conditions) to public variables and using them to recover key bits already appeared in previous work. In [15], the dynamic constraints were used to enhance differential and high order differential attacks by limiting the propagation of differences in the internal state of the cipher. This technique was applied to reduced variants of a few ciphers, and in particular was used to recover 2 key bits of Grain-128 when the number of initialization rounds is reduced from 256 to 213. In dynamic cube attacks, the dynamic constraints are used in a completely different way: The first and most crucial step of dynamic cube attacks is the careful analysis of the output function of the cipher. This analysis allows us to select constraints that weaken the resistance of the cipher to cube testers. Our carefully selected constrains, in addition to our novel algebraic key-recovery techniques, allow us to obtain much improved results on the cipher Grain-128: Our attack on a Grain-128 variant that uses 207 initialization rounds recovers the complete key (rather than a few key bits) with feasible complexity. In addition, we break a Grain-128 variant

with 250 initialization rounds and a weak key set (containing a fraction of 2^{-10} of all the possible keys) of the full version of the cipher.

Next, we briefly describe the standard cube testers and cube attacks (for more details, refer to [3] and [4]). We then describe the new attack in detail and present our results on the cipher Grain-128 [6]. Finally, we conclude and list some open problems.

2 Cube Attacks and Cube Testers

2.1 Cube Attacks

In almost any cryptographic scheme, each output bit can be described by a multivariate master polynomial $p(x_1, .., x_n, v_1, .., v_m)$ over $GF(2)$ of secret variables x_i (key bits), and public variables v_j (plaintext bits in block ciphers and MACs, IV bits in stream ciphers). The cryptanalyst is allowed to tweak the master polynomial by assigning chosen values for the public variables, which result in derived polynomials, and his goal is to solve the resultant system of polynomial equations in terms of their common secret variables. The basic cube attack [4] is an algorithm for solving such polynomials, which is closely related to previously known attacks such as high order differential attacks [11] and AIDA [12].

To simplify our notation, we now ignore the distinction between public and private variables. Given a multivariate master polynomial with n variables $p(x_1, .., x_n)$ over $GF(2)$ in algebraic normal form (ANF), and a term t_I containing variables from an index subset I that are multiplied together, the polynomial can be written as the sum of terms which are supersets of I and terms that miss at least one variable from I:

$$p(x_1, .., x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, .., x_n)$$

$p_{S(I)}$ is called the *superpoly* of I in p. Note that the superpoly of I in p is a polynomial that does not contain any common variable with t_I, and each term in $q(x_1, .., x_n)$ does not contain at least one variable from I. Moreover, compared to p, the algebraic degree of the superpoly is reduced by at least the number of variables in t_I.

The basic idea behind cube attacks is that the symbolic sum over $GF(2)$ of all the derived polynomials obtained from the master polynomial by assigning all the possible 0/1 values to the subset of variables in the term t_I is exactly $p_{S(I)}$ which is the superpoly of t_I in $p(x_1, .., x_n)$. A *maxterm* of p is a term t_I such that the superpoly of I in p is a linear polynomial which is not a constant.

The cube attack has two phases: the preprocessing phase, and the online phase. The preprocessing phase is not key-dependant and is performed once per cryptosystem. The main challenge of the attacker in the preprocessing phase is to find sufficiently many maxterms with linearly independent superpolys. Linear superpolys are not guaranteed to exist, and even when they exist, finding them can be a challenging preprocessing task. However, once sufficiently many linearly independent superpolys are found for a particular cryptosystem, we can repeatedly use them to easily find any secret key during the online phase.

2.2 Cube Testers

Similarly to cube attacks, cube testers [3] work by evaluating superpolys of terms of public variables. However, while cube attacks aim to recover the secret key, the goal of cube testers is to distinguish a cryptographic scheme from a random function, or to detect non-randomness by using algebraic property testing on the superpoly. One of the natural algebraic properties that can be tested is balance: A random function is expected to contain as many zeroes as ones in its truth table. A superpoly that has a strongly unbalanced truth table can thus be distinguished from a random polynomial by testing whether it evaluates as often to one as to zero. Other efficiently detectable properties include low degree, the presence of linear variables, and the presence of neutral variables.

In the preprocessing phase of cube testers, the attacker finds terms whose superpolys have some efficiently testable property.

3 A Simple Example of Dynamic Cube Attacks

Both standard (static) cube testers and dynamic cube attacks sum the output of the cipher over a given cube defined by a subset of public variables, which are called cube variables. In static cube testers, the values of all the public variables that are not summed over are fixed to a constant (usually zero), and thus they are called static variables. However, in dynamic cube attacks the values of some of the public variables that are not part of the cube are not fixed. Instead, each one of these variables (called dynamic variables) is assigned a function that depends on some of the cube public variables and some expressions of private variables. Each such function is carefully chosen, usually in order to zero some state bits in order to amplify the bias (or the non-randomness in general) of the cube tester. Dynamic cube attacks are clearly a generalization of standard cube testers, but also allow us to directly derive information on the secret key without solving any algebraic equations. Moreover, choosing the dynamic variables carefully may help to improve the time complexity of distinguishers obtained by using standard cube testers (we will need fewer cube variables to obtain a distinguisher). We note that the drawback of the new attack compared to basic cube attacks and cube testers, is that it requires a more complex analysis of the internal structure of the cipher.

To demonstrate the idea of the attack, we consider a polynomial P which is a function of the three polynomials P_1, P_2, and P_3:

$$P = P_1 P_2 + P_3$$

P_1, P_2, and P_3 are polynomials over five secret variables x_1, x_2, x_3, x_4, x_5 and five public variables v_1, v_2, v_3, v_4, v_5:

$P_1 = v_2 v_3 x_1 x_2 x_3 + v_3 v_4 x_1 x_3 + v_2 x_1 + v_5 x_1 + v_1 + v_2 + x_2 + x_3 + x_4 + x_5 + 1$
$P_2 =$ arbitrary dense polynomial in the 10 variables

$$P_3 = v_1v_4x_3x_4 + v_2x_2x_3 + v_3x_1x_4 + v_4x_2x_4 + v_5x_3x_5 + x_1x_2x_4 + v_1 + x_2 + x_4$$

Since P_2 is unrestricted, P is likely to behave randomly and it seems to be immune to cube testers (or to cube attacks). However, if we can set P_1 to zero, we get $P = P_3$. Since P_3 is a relatively simple function, it can be easily distinguished from random. We set $v_4 = 0$ and exploit the linearity of v_1 in P_1 to set $v_1 = v_2v_3x_1x_2x_3 + v_2x_1 + v_5x_1 + v_2 + x_2 + x_3 + x_4 + x_5 + 1$ which forces P_1 to zero. During the cube summation, the value of the dynamic variable v_1 will change according to its assigned function. This is in contrast to the static variable v_4, whose value will remain 0 during the cube summation. At this point, we assume that we know the values of all the secret expressions that are necessary to calculate the value of v_1: $x_1x_2x_3$, x_1, and $x_2 + x_3 + x_4 + x_5 + 1$. Plugging in the values for v_1 and v_4, we get:

$$\begin{aligned}&P = v_2x_2x_3 + v_3x_1x_4 + v_5x_3x_5 + x_1x_2x_4 + x_2 + x_4 + \\ &(v_2v_3x_1x_2x_3 + v_2x_1 + v_5x_1 + v_2 + x_2 + x_3 + x_4 + x_5 + 1) = \\ &v_2v_3x_1x_2x_3 + v_2x_2x_3 + v_3x_1x_4 + v_5x_3x_5 + x_1x_2x_4 + v_2x_1 + v_5x_1 + v_2 + x_3 + x_5 + 1\end{aligned}$$

After these substitutions, we can see that the simplified P is of degree 2 in the public variables, and there is only one term ($v_2v_3x_1x_2x_3$) of this degree. We have 3 free public variables (v_2, v_3, v_5) that are not assigned. We can now use them as cube variables: The sum over the big cube $v_2v_3v_5$ and two of its subcubes v_2v_5 and v_3v_5 is always zero. Moreover, the superpoly of v_2v_3 is $x_1x_2x_3$, which is zero for most keys. Thus, we can easily distinguish P from a random function using cube testers. However, the values of the expressions $x_1x_2x_3$, x_1, and $x_2 + x_3 + x_4 + x_5 + 1$ are unknown in advance, and it is not possible to calculate the dynamic values for v_1 without them. Thus, we guess the 3 values of the expressions (modulo 2). For each of the 8 possible guesses (there are actually 6 possible guesses since $x_1 = 0$ implies $x_1x_2x_3 = 0$, but this optimization is irrelevant at this point), we run the cube tester, and get 4 0/1 values - a value for each cube sum. The 7 wrong guesses will not zero P_1 throughout the cube summations. Hence the 4 cube sums for each wrong guess are likely to behave randomly, and it is unlikely that more than 1 wrong guess will give 4 zero cube sum values. On the other hand, the 4 cube sums for the correct guess will all equal to 0 with high probability. Hence, for most keys, we expect to remain with at most 2 possible guesses for the 3 expressions and we can recover the values for the expressions that are assigned a common value by these 2 guesses. This gives us a distinguisher for P and allows us to derive information regarding the secret key.

In the general decomposition of a polynomial P as $P = P_1P_2 + P_3$, we call P_1 (according to which we assign the dynamic variable) the *source polynomial*, P_2 the *target polynomial* and P_3 the *remainder polynomial*. There are many ways to express P is such a way, and the choice of source and target polynomials requires careful analysis of the given cipher.

4 Dynamic Cube Attacks on Grain-128

4.1 Description on Grain-128

We give a brief description of Grain-128, for more details one can refer to [6]. The state of Grain-128 consists of a 128-bit LFSR and a 128-bit NFSR. The feedback functions of the LFSR and NFSR are respectively defined to be

$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}$

$b_{i+128} = s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + b_{i+68}b_{i+84}$

The output function is defined as

$z_i = \sum_{j\in\mathcal{A}} b_{i+j} + h(x) + s_{i+93}$, where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$

where the variables x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7 and x_8 correspond to the tap positions b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} and s_{i+95} respectively.

Grain-128 is initialized with a 128-bit key that is loaded into the NFSR, and with a 96-bit IV that is loaded into the LFSR, while the remaining 32 LFSR bits are filled with the value of 1. The state is then clocked through 256 initialization rounds without producing an output, feeding the output back into the input of both registers.

4.2 Previous Attacks

Several attacks have been published on Grain-128 variants: [7] found a distinguisher when the number of initialization rounds is reduced from 256 to 192 rounds, [8] described shortcut key-recovery attacks on a variant with 180 initialization rounds, and [9] exploited a sliding property to speedup exhaustive search by a factor of two. Related-key attacks on the full cipher were presented in [10]. However, the relevance of related-key attacks is disputed and we concentrate on attacks in the single key model. Stankovski [16] presented a distinguishing attack on a variant that uses 246 initialization rounds, which works for less than half of the keys. The most powerful distinguishing attack on most keys of Grain-128 was given in [5], where cube testers were used in order to distinguish the cipher from random for up to 237 initialization rounds. Moreover, the authors claim that by extrapolating their experimentally verified results, one can argue that cube testers may be used in order to attack the full cipher. However, this conjecture has not been verified in practice due to the infeasibility of the attack. Note that [5] only gives a distinguisher, and leaves the problem of exploiting cube testers (or cube attacks) for key recovery open. More recently [15] used conditional differential cryptanalyses to recover 2 key bits of Grain-128 with 213 initialization rounds, which gives the best known key-recovery attack in the single key model up to this point.

4.3 Outline of the New Attacks on Grain-128

We present 3 attacks:

1. A feasible full key recovery attack on a Grain-128 variant that uses 207 initialization rounds, while utilizing output bits $208 - 218$.

2. An experimentally verified full key recovery attack on a Grain-128 variant with 250 initialization rounds.
3. An experimentally verified attack on the full Grain-128, which can recover a large subset of weak keys (containing 2^{-10} of all the possible keys).

We begin by describing the common steps shared by these three attacks. We then elaborate on each attack in more detail.

The preprocessing phase of the attacks consists of 2 initial steps:

Step 1. We first choose the state bits to nullify, and show how to nullify them by setting certain dynamic variables to appropriate values.

This is a complex process that cannot be fully automated and involves manual work to analyze the cipher: When applying the attack to Grain-128, we would like to decompose its output function into a source and target polynomials (representing intermediate state bits multiplied together), and a remainder polynomial which should be more vulnerable to cube testers than the original output. In our small example, this was easy since we could explicitly write down and analyze its ANF in terms of the public and private variables. However, the output function of Grain-128 is too complex to decompose and analyze in such a way. Our approach in this paper is to use the recursive description of the cipher's output function in order to find a good decomposition.

In the case of Grain-128, specific non-linear terms in the cipher's output stand out as being of higher degree than others and are good candidates to be nullified or simplified. The output function of Grain-128 is a multivariate polynomial of degree 3 in the state. The only term of degree 3 is $b_{i+12}b_{i+95}s_{i+95}$, and hence we focus on nullifying it. Since b_{i+12} is the state bit that is calculated at the earliest stage of the initialization steps (compared to b_{i+95} and s_{i+95}), it should be the least complicated to nullify. However, after many initialization steps, the ANF of b_{i+12} becomes very complicated and we were not able to nullify it when more than 230 initialization rounds are used (i.e. for $i > 230$). The compromise we make is to simplify (and not nullify) $b_{i+12}b_{i+95}s_{i+95}$: We write the most significant term of degree 3 that is used in the calculation of these state bits, which for b_{i+12} is $b_{i-128+12+12}b_{i-128+95+12}s_{i-128+95+12} = b_{i-104}b_{i-21}s_{i-21}$. The most significant term for both b_{i+95} and s_{i+95} is $b_{i-128+12+95}b_{i-128+95+95}s_{i-128+95+95} = b_{i-21}b_{i+62}s_{i+62}$. We can see that b_{i-21} participates in all terms, and thus nullifying it is likely to simplify the ANF of $b_{i+12}b_{i+95}s_{i+95}$ significantly.

The ANF of the earlier b_{i-21} is much easier to analyze compared to the one of b_{i+12}, but it is still very complex. Thus, we perform more iterations in which we simplify b_{i-21} further by using its recursive description to nullify previous state bits. When the ANF representation of b_{i-21} is simple enough, we select a linear public variable in its ANF and assign to it an expression which will make the whole expression identically zero. We elaborate on this multistage process for output bit 215 of Grain-128 (used in attack 1): We would like to zero $b_{215-21} = b_{194}$. However, we do not zero it directly. We first zero 4 other state bits in order to simplify its ANF representation. The details of how these bits were chosen are given in Appendix A.

Step 2. We choose a big cube and a set of subcubes to sum over during the online phase. We then determine the secret expressions that need to be guessed in order to calculate the values of the dynamic variables during the cube summations.

Some choices of the big cube give better results than other, and choosing a cube that gives good results is a crucial part of the preprocessing. One can use heuristics in order to find cubes that give better results (an example of a heuristic is given in [5]). However, it is difficult to predict in advance which cubes will give good results without actually executing the attack and calculating the results for many cubes.

The secret expressions that need to be guessed are calculated according to the symbolic expressions of the dynamic variables and the chosen big cube. This is a simple process that can be easily automated:

1. Given the symbolic form of a dynamic variable, look for all the terms which are combinations of variables from the big cube. In our simple example, the symbolic form of the single dynamic variable is $v_2v_3x_1x_2x_3 + v_2x_1 + v_5x_1 + v_2 + x_2 + x_3 + x_4 + x_5 + 1$. Our big cube is $v_2v_3v_5$. The terms which are combinations of variables from the big cube in the symbolic form are v_2v_3, v_2, v_5 and the empty combination.
2. Rewrite the symbolic form as a sum of these terms, each one multiplied by an expression of secret variables. In our example, we write $v_2v_3x_1x_2x_3 + v_2x_1 + v_5x_1 + v_2 + x_2 + x_3 + x_4 + x_5 + 1 = v_2v_3(x_1x_2x_3) + v_2(x_1 + 1) + v_5(x_1) + (x_2 + x_3 + x_4 + x_5 + 1)$,
3. Add the expressions of secret variables to the set of expressions that need to be guessed. In the example, we add $x_1x_2x_3$, x_1 and $x_2 + x_3 + x_4 + 1$ (note that guessing the value of x_1 is the same as guessing the value of $x_1 + 1$, and we do not add it twice). In addition, we do not add expressions whose value can be deduced from the values of the expressions already in the set. For example, if x_1 and x_2 are is the set, we do not add x_1x_2 or $x_1 + x_2$.

In steps $3-4$, the attacker uses the parameters obtained in the first two steps in order to derive information regarding the secret key. These steps constitute the online phase of the attack that is executed by the attacker after the secret key has been set. In addition, steps $3-4$ are simulated by the attacker in the preprocessing phase for several pseudo random keys, in order to verify his choices in steps $1-2$.

Step 3

1. For each possible value (guess) of the secret expressions, sum over the subcubes chosen in the previous step with the dynamic variables set accordingly, and obtain a list of sums (one sum per subcube).
2. Given the list of sums, calculate the guess score (which measures the non-randomness in the subcube summations). The output of this step is a sorted guess score list in which guesses are sorted from the lowest score to the highest.

Given that the dimension of our big cube is d, the complexity of summing over all its subcubes is bounded by $d2^d$ (this can be done using the Moebius transform [13]). Given that we need to guess the values of e expressions, the complexity of this step is bounded by $d2^{d+e}$. However, the data complexity of this step can by significantly lower than the time complexity: Assuming that we have only $y \leq e$ dynamic variables, the data complexity is bounded by 2^{d+y} (an output bit for every possible value of the cube and dynamic variables is sufficient).

After we obtain the summation values for each of the subcubes for a specific guess, we determine its score. The simple score function that we use in this paper measures the percentage of 1 values in the summations. The reason that we consider summation values which are biased towards 0 as non-random (but not summation values which are biased towards 1) is that the superpolys of the cubes in our attacks tend to be extremely sparse, and their ANF contains the constant 1 (or any other term) with very low probability. Such sparse polynomials evaluate to zero for almost all keys.

Step 4 Given the sorted guess score list, we determine the most likely values for the secret expressions, for a subset of the secret expressions, or for the entire key. The straightforward approach to calculate the values for the secret expressions is to simply take the values for the expressions from the guess that has the lowest score (or the least percentage of 1 values in its summations values), in the sorted guess list. However, this approach does not always work. Depending on the setting of the attack, there could be guesses that have a score that is at least as low as the correct guess score: In our small example, the correct guess score is expected to be 0, however there is a reasonable probability that there is another arbitrary guess with the score of 0. Therefore, the details of this step vary according to the attack and are specified separately for each attack.

4.4 Details of the First Attack

The first attack is a full key recovery attack on a Grain-128 variant that uses 207 initialization rounds, while utilizing output bits $208 - 218$. The key bits are recovered in small groups of size $1 - 3$, where each group is recovered using a different set of parameters that was obtained in the preprocessing phase.

One set of parameters for the attack is given in Table 1 in Appendix B. We now specify how the attack is carried out given the parameters of this table: First, we assign to each one of the dynamic variables in the table its symbolic value. Appendix B shows how to do this given the parameters of Table 1, and the assignment algorithm for the other tables in this paper is similar.

After all the dynamic variables are assigned, we determine the secret expressions that need to be guessed in order to fully calculate the values of the dynamic variables during the cube summations (step 2). Altogether, there are 7 expressions that need to be guessed, and since the big cube is of dimension 19, the total complexity of the step 3 of the attack with this specific set of parameters is about $19 \times 2^{19+7} < 2^{31}$. We will use only linear expressions for the full key recovery (step 4), hence we concentrate on retrieving their value from the sorted

guess list. The two linear expressions actually contain only a single key bit and are listed in the "Expressions Retrieved" row. We sum on all subcubes of dimension at least $19 - 3 = 16$ of the big cube (of dimension 19), and the score for each guess is simply the fraction of 1 values among all the subcube sums. In step 4, we retrieve the value of the 2 expressions by taking the corresponding values from the best guess. We simulated the attack with the above parameters with hundreds of random keys. The attack failed to retrieve the correct values for the expressions $x_{127}, x_{122} + 1$ for about 10% of the keys. However for all the failed keys, the score of the best guess was at least 0.44 (i.e. the dynamic cube tester did not give a strong distinguisher), and thus we know when we fail by declaring the expressions as "undetermined" whenever we encounter a key for which the best guess score is at least 0.44 (this occurs for about 15% of the keys). This is important for the full key recovery attack that is described next.

We showed how to retrieve 2 key bits with high probability with one carefully chosen set of parameters. It is not difficult to find more sets of parameters that allow us to retrieve more key bits. Another example of such a set of parameters that uses the same output bit is given in table 2 in Appendix B. Note that we only changed the chosen big cube, which in turn changed the retrieved expressions. A different set of parameters that uses output bit 218 is given in table 3 in Appendix B. Altogether, we have 55 sets of parameters that ideally allow us to recover 86 of the 128 key bits. For each set of parameters, the score calculation method is identical to the one described above, i.e. we compute the percentage of 1 values in the cube sums for all cubes of dimension at least $d - 3$. The key recovery method is identical as well, i.e. we recover the values of the secret linear expressions from the guess with the best score, but only if its score is at least 0.44. We simulated the full attack on hundreds of random keys. On average, we could retrieve about 80 secret key bits per key. The remaining 48 key bits can be recovered with feasible complexity by exhaustive search.

We note that it is possible to retrieve more key bits in a similar way by using more output bits (e.g. output bits 219, 220, etc.), or using the same output bits with different sets of parameters. A more efficient key recovery method can try to determine values of non-linear secret expressions, some of which can be made linear by plugging in values for secret key bits which we already recovered. However, our main goal is to attack much stronger variants of Grain-128, as described next.

4.5 A Partial Simulation Phase

When attacking Grain-128, we perform the preprocessing steps $(1, 2)$ and then simulate the online steps of the attack $(3, 4)$ for several random keys. In this case, steps 3 and 4 are performed in order to estimate the success of the attack and are called the simulation phase. If we are not satisfied with the results, we can repeat steps 1 and 2 by choosing different parameters and performing another simulation phase. This process can be very expensive and its complexity is generally dominated by step 3. We can significantly reduce the complexity of the simulation phase by calculating the cube summations only for the correct

guess and observing whether the correct guess exhibits a significant non-random property for most keys. This is unnecessary for the first attack in which we can run the full simulation phase and recover the secret key. However, in the second and third attacks, we try to attack variants of Grain-128 which are significantly stronger and the simulation phase becomes infeasible even for a single random key. In these cases, the observed non-randomness for the correct guess provides strong evidence that the stronger variants of Grain-128 can also be broken by the full key recovery version of the attack.

Given that we choose a big cube of size d and guess e expressions, the complexity of the cube summations when running the full simulation phase on one key is about $d2^{d+e}$ bit operations. However, the complexity of the simulation phase is actually dominated by the 2^{d+e} executions of the cipher: Assuming that each execution requires about b bit operations, the total complexity is about $b2^{d+e}$ (for Grain-128 $b > 2^{10} >> d$). Similarly, the partial simulation phase on one key requires $b2^d$ bit operations. Since the complexity does not depend on e, we can feasible verify the behavior of dynamic cube attacks even when their total complexity is infeasible when the dimension of the cube d is not too large. This ability to experimentally verify the performance of dynamic cube attacks is a major advantage over static cube attacks and cube testers.

4.6 A Generic Key Recovery Method

In the first attack, we run the full simulation phase and obtain the sorted guess list in step 3. Since we can do this many times and calculate the complexity of the attack, we tailored the key derivation algorithm used in step 4 such that it is very efficient for our chosen parameter sets. On the other hand, in the second and third attacks, we must perform the partial simulation phase as described above and we obtain only the score for the correct guess. Since we do not have the sorted guess list, we cannot calculate the exact complexity of the attack and we cannot customize the algorithm used in step 4 as in the first attack (for example, we cannot verify that the first guess in the sorted guess list assigns correct values for some expressions, as in the first attack). As a result, we use a key recovery method which is more generic in a sense that it is not tailored to a specific cipher, or to a specific set of parameters. The only property of the parameter sets for the attacks that it exploits, is that many guessed key expressions are linear. We now describe the details of this method as performed in real time (not in the simulation phase) and then estimate its complexity.

Assume that we have executed steps $1-3$ for Grain-128 with $n = 128$ secret key bits. Our big cube is of dimension d and we have e expressions to guess, out of which l are linear. Our sorted guess score list is of size 2^e and the correct guess is located at index g in the sorted list.

1. Consider the guesses from the lowest score to the highest: For each guess (that assigns values to all the expressions), perform Gaussian Elimination on the l linear expressions and express l variables as linear combinations of the other $n - l$ variables.

2. Exhaustively search the possible 2^{n-l} values for those $n-l$ variables: For each value, get the remaining part of the key from the linear expressions, execute the cipher with the key, and compare the result to the given data. If there is equality, return the full key.

Overall, we have 2^{n-l} iterations per guess. The evaluation of the linear expressions can be performed efficiently if we iterate over the 2^{n-l} values using Gray Codes. Hence, we assume that the evaluation of the linear expressions takes negligible time compared to the execution of the cipher. The total running time per guess is thus about 2^{n-l} cipher executions and the overall running time of step 4 is $g \times 2^{n-l}$. We can also to try improve the running time by using some of the $e-l$ non linear expressions which can be efficiently evaluated (compared to a single cipher execution): For each key we first check if the key satisfies these non linear equations before executing the cipher.

The complexity of the generic key recovery method is dependent on g which denotes the index of the correct guess in the sorted guess list. The expected value of g can be estimated for a random key by running several simulations of the attack on random keys. However, when the simulation phase is infeasible and we are forced to perform a partial simulation phase, we cannot estimate g this way since we do not have the guess list. A possible solution to this problem is to assume that all the incorrect guesses behave randomly (i.e. the subcube sums are independent uniformly distributed boolean random variables). Under this assumption, we run the partial simulation on an arbitrary key. If the cube sums for the correct guess detect a property that is satisfied by a random cipher with probability p, then we can estimate $g \approx max\{p \times 2^e, 1\}$.

The assumption that incorrect guesses behave randomly is clearly an oversimplification. In the first attack, we retrieve the value of a carefully chosen subset of the expressions by taking the corresponding values from the best guess. However for about half of the keys the best guess is not the correct guess, i.e. it does not assign the correct values for all the expressions, but rather to our chosen set of expressions. In other words, there are specific (non arbitrary) incorrect guesses that are likely to have a low score that can be at least as low as the score of the correct guess. These incorrect guesses usually assign a correct value to a fixed subset of the guessed expressions. In order to understand this, consider the following example: assume that $P = P_1P_2 + P_3$, the source and target polynomials P_1 and P_2 are of degree 3, and the remainder polynomial is of degree 5 (all degrees are in terms of the public variables). We choose a dynamic variable to nullify P_1, and assume for the sake of simplicity that the degrees of P_2 and P_3 do not change after assigning this variable. We choose a cube of dimension 7, and sum on all its subcubes of dimension 6 and 7. Clearly, the correct guess will have a score of 0. However, any other which reduces the degree of P_1 to 1 or 0 will also have a score of 0.

To sum up, our estimation of g (and hence our estimation for the complexity of the attack) may not be completely accurate since incorrect guesses do not behave randomly. However, our simulations on Grain-128 variants on which the simulation phase is feasible, show that the effect of the incorrect guesses biased

towards 0 is usually insignificant, and our estimation of g is reasonable. In addition, even incorrect non-uniform guesses are still likely to be highly correlated with the correct guess, and thus they can actually speed up the attack (this was experimentally verified in our first attack, which has a feasible complexity). Hence, our estimation of the complexity of step 4 of the attack is a reasonable upper bound.

4.7 Details of the Second Attack

In order to attack the almost full version of Grain-128 with 250 initialization rounds (out of 256), we nullify $b_{251-21} = b_{230}$. The parameters of the attack are specified in Table 4 in Appendix B. As in the first attack, most of the dynamic variables are used in order to simplify b_{230}. Note that we need many more dynamic variables compared to the previous attack. This is because it is much more difficult to nullify b_{230} than to nullify b_{194} or b_{197} (for example). In addition, we set v_{82} to the constant value of 1 so that v_{89} can function as a dynamic variable that nullifies b_{197}. Since the big cube is of dimension 37 and we have 24 dynamic variables, the data and memory complexity is $2^{37+24} = 2^{61}$. The number of expressions that need to be guessed seems to be 84. However, after removing many linearly dependent expressions, this number can be reduced to 59. Thus, the total complexity of the cube summations is about $37 \times 2^{37+59} < 2^{101}$, implying that we have to use the partial simulation phase. Out of the 59 expressions that need to be guessed, 29 contain only a single key bit on which we concentrate for generic key recovery.

During the partial simulation phase, we summed on all subcubes of dimension at least 35 of the big cube, calculating the percentage of 1 values separately for all the subcubes of each dimension (35, 36, or 37). We performed the partial simulation phase on dozens of random keys. For the sake of completeness, we also sampled a few random incorrect guesses for several keys and verified that they do not have a significant bias. For about 60% of the keys, the subcube sums for the correct guess contained only 0 values for the subcube of sizes 36 and 37, and less than 200 '1' values among the 666 subcubes of size 35. Assuming that incorrect guesses behave randomly, we expect the correct guess to be among the first guesses in the sorted guess list. The complexity of the unoptimized version of the attack (that ignores the non-linear expressions) is dominated by the exhaustive search for the remaining $128 - 29 = 99$ key bits per guess. Overall the complexity for about 60% of the keys is about 2^{100} cipher evaluations, and can almost surely be optimized further. For another 30% of the keys we tested, the non-randomness in the subcube sums was not as significant as in the first 60%, but still significant enough for the attack to be much faster than exhaustive search. For the remaining 10% of the keys, the non-randomness observed was not significant enough and the attack failed. However, we are certain that most of these problematic keys can still be broken by selecting different parameters for the attack.

4.8 Details of the Third Attack

In order to attack the full version of Grain-128 with 256 initialization rounds, we have to nullify $b_{257-21} = b_{236}$. However, the ANF of b_{236} is too complicated to zero using our techniques, and we had to make assumptions on 10 secret key bits in order nullify it. As a result, we could verify the correctness of our attack on the full version of Grain-128 only for a subset of about 2^{-10} of the possible keys in which 10 key bits are set to zero. Our current attack can thus be viewed as an attack on an unusually large subset of weak keys, but it is reasonable to assume that it can be extended to most keys with further improvements.

The parameters of the attack are specified in Table 5 in Appendix B. Since the big cube is of dimension 46 and we have 13 dynamic variables, the data and memory complexity is $2^{46+13} = 2^{59}$. After removing many linearly dependent expressions, the number of guessed expression is 61. Thus, the total complexity of the cube summations is about $46 \times 2^{46+61} < 2^{113}$ bit operations. Out of the 61 expressions that need to be guessed, 30 contain only a single key bit. Moreover, we can fix the values of 35 more variables such that 30 out of the remaining $61 - 30 = 31$ expression become linear. In order to recover the key efficiently, we use an extension of the generic key recovery method: Let the key be n, and denote the dimension of the big cube by d. Assume that given the values of c variables we can plug them into l (linear or non-linear) expressions such that they become linear, and perform Gaussian Elimination which makes it possible to express l variables as linear combinations of the remaining (unspecified) $n-l-c$ variables.

1. Consider the guesses from the lowest score to the highest: For each guess, iterate the $n-l$ variables using Gray Coding such that the c variables function as most significant bits (i.e their value changes every 2^{n-l-c} iterations of the remaining $n - l - c$ variables).
2. For each value of the c variables, perform Gaussian Elimination and express l variables as linear combinations of the remaining $n - l - c$ variables.
3. For each value of the remaining $n - l - c$ variables, compute the values of the l linear variables, execute the cipher with this derived key and compare the result to the given data. If there is equality, return the full key.

In our case, we have $n = 118$ (after fixing 10 key bits), $c = 35$, and $l = 60$. We call the second sub-step in which we perform Gaussian Elimination a big iteration and the third sub-step in which we do not change any value among the $c = 35$ variables, a small iteration. Note that big iterations are performed only every $2^{n-l-c} = 2^{23}$ small iterations. It is clear that computing the linear equations and performing Gaussian Elimination with a small number of variables in a big iteration takes negligible time compared to executing the cipher 2^{23} times in small iterations. Hence the complexity of the exhaustive search per guess is dominated by the small iterations and is about $2^{n-l} = 2^{58}$ cipher evaluations (as in the original generic key recovery method) . In order to complete the analysis of the attack, we need to describe the score calculation method and the estimate the index g of the correct guess.

During the partial simulation phase, we summed on all subcubes of dimension at least 44 of the big cube, calculating the percentage of 1 values separately for all the subcubes of each dimension. We performed simulations for 5 random keys (note that each simulation requires 2^{46} cipher executions, which stretched our computational resources): For 3 out of the 5 keys, we observed a significant bias towards 0 (which is expected to occur with probability less than 2^{-20} for a random cipher) in the subcubes of dimension 45 and 46. This implies that $g \approx 2^{61} \times 2^{-20} = 2^{41}$ and the total complexity of step 4 is about $2^{41} \times 2^{118-60} = 2^{99}$ cipher evaluations. Assuming that each cipher evaluation requires about 2^{10} bit operations, the total complexity of the attack remains dominated by step 3, and is about 2^{113} bit operations. This is better than exhaustive search by a factor of about 2^{15} even when we take into account the fact that our set of weak keys contains only 2^{-10} of the 2^{128} possible keys. For another key, the bias towards 0 in the subcubes of dimension 45 and 46 was not as strong and we also need to use the bias towards 0 in the subcubes of dimension 44. For this key, we were able to improve exhaustive search by a factor of about 2^{10}. For the fifth key, we also observed a bias towards 0, but it was not strong enough for a significant improvement compared to exhaustive search. As in the previous attack, we stress that it should be possible to choose parameters such that the attack will be significantly better than exhaustive search for almost all keys in the weak key set.

4.9 Discussion

Any attack which can break a fraction of 2^{-10} of the keys is sufficiently significant, but in addition we believe that our third attack can be improved to work on a larger set of weak keys of Grain-128. This can be done by making fewer assumptions on the key and optimizing the process of nullification of b_{236}. However, we do not believe that nullifying b_{236} will suffice to attack most keys of Grain-128. For such an attack, the most reasonable approach would be to choose a larger big cube to sum over, while nullifying fewer state bits at earlier stages of the cipher initialization process. The question whether a key recovery attack on most keys of Grain-128 can be feasibly simulated to yield an experimentally verified attack remains open.

5 Generalizing the Attack

In the previous section, we described in detail the dynamic cube attack on Grain-128. However, most of our techniques can naturally extend to other cryptosytems. In this section, we describe the attack in a more generic setting, emphasizing some important observations.

Step 1. As specified in the attack on Grain-128, choosing appropriate state bits to nullify and actually nullifying them is a complex process. In the case of Grain-128, specific non-linear terms in the cipher's output stand out as being of

higher degree and enable us to decompose the output function to a source and target polynomials relatively easily. It is also possible to find good decompositions experimentally: We can tweak the cipher by removing terms in the output function. We then select various cubes and observe whether the tweaked cipher is more vulnerable to cube testers than the original cipher. If the tweaked cipher is indeed more vulnerable, then the removed terms are good candidates to nullify or simplify.

As in the case of Grain-128, there are several complications that may arise during the execution of this step and hence it needs to be executed carefully and repeatedly through a process of trial and error. One complication is that zeroing a certain group of state bits may be impossible due to their complex interdependencies. On the other hand, there may be several options to select dynamic variables and to zero a group of state bits. Some of these options may give better results than others. Another complication is that using numerous dynamic variables may overdeplete the public variables that we can use for the cube summations.

Step 2. The choice of a big cube, can have a major impact on the complexity of the attack. Unfortunately, as specified in the attack on Grain-128, in order to find a cube that gives good results we usually have to execute the attack and calculate the results for many cubes. After the big cube is chosen, the secret expressions that need to be guessed are calculated according to the simple generic process that is used for Grain-128.

Step 3. The only part of this step that is not automated is the score calculation technique for each guess from the subcube sums. We can use the simple method of assigning the guess its percentage of 1 values, or more complicated algorithms that give certain subcubes more weight in the score calculation (e.g. the sum of high dimensional subcubes can get more weight than the sum of lower dimensional ones, which tend to be less biased towards 0).

Step 4. Techniques for recovering information about the key differ according to the attack. It is always best to adapt the technique in order to optimize the attack as in the first attack on Grain-128. In this attack, we determined the values of some carefully chosen key expression from the guess with the best score. It is possible to generalize this technique by determining the value of a key expression (or several key expressions) according to a majority vote taken over several guesses with the highest score. We can also try to run the attack with different sets of parameters, but with some common guessed expressions. The values for those common guessed expressions can then be deduced using more data from several guess score lists.

When the simulation phase (steps 3 and 4) is not feasible we must use the partial simulation phase. The generic key recovery method and its extension in the third attack on Grain-128 can be used in case many of the guessed key expressions are linear, or can be made linear by fixing the values of some key bits.

6 Conclusions and Open Issues

Dynamic cube attacks provide new key recovery techniques that exploit in a novel way distinguishers obtained from cube testers. Our results on Grain-128 demonstrate that dynamic cube attacks can break schemes which seem to resist all the previously known attacks. Unlike cube attacks and cube testers, the success of dynamic cube attacks can be convincingly demonstrated beyond the feasible region by trying sufficiently many random values for the expressions we have to guess during the attack.

An important future work item that was discussed in section 4.9 is how to break most keys of Grain-128. In addition, the new techniques should be applied to other schemes. Preliminary analysis of the stream cipher Trivium [14] suggests that dynamic cube attacks can improve the best known attack on this cipher, but the improvement factor we got so far is not very significant.

References

1. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
2. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
3. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
4. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
5. Aumasson, J.-P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. In: SHARCS - Special-purpose Hardware for Attacking Cryptographic Systems (2009)
6. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: IEEE International Symposium on Information Theory, ISIT 2006 (2006)
7. Englund, H., Johansson, T., Sönmez Turan, M.: A framework for chosen IV statistical analysis of stream ciphers. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 268–281. Springer, Heidelberg (2007)
8. Fischer, S., Khazaei, S., Meier, W.: Chosen IV statistical analysis for key recovery attacks on stream ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 236–245. Springer, Heidelberg (2008)
9. De Cannière, C., Kücük, Ö., Preneel, B.: Analysis of Grain's initialization algorithm. In: SASC 2008 (2008)
10. Lee, Y., Jeong, K., Sung, J., Hong, S.H.: Related-key chosen IV attacks on grain-v1 and grain-128. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 321–335. Springer, Heidelberg (2008)

11. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Symposium on Communication, Coding and Cryptography, in Honor of James L. Massey on the Occasion of His 60'th Birthday, pp. 227–233 (1994)
12. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413 (2007)
13. Joux, A.: Algorithmic Cryptanalysis, pp. 285–286. Chapman & Hall, Boca Raton
14. De Cannière, C., Preneel, B.: Trivium - a stream cipher construction inspired by block cipher design principles. estream, ecrypt stream cipher. Technical report of Lecture Notes in Computer Science
15. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
16. Stankovski, P.: Greedy Distinguishers and Nonrandomness Detectors. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 210–226. Springer, Heidelberg (2010)

A Appendix: Zeroing State Bits of Grain-128

To demonstrate the process that we use to zero state bits of Grain-128, consider the problem of zeroing b_{194}. The ANF representation of b_{194} is a relatively small polynomial of degree 9 in the 128 secret variables and 96 public variables which contains 9813 terms. It is calculated by assigning all the key and IV bits a distinct symbolic variable, and calculating the symbolic value of the feedback to the NFSR after 67 rounds. It may be possible to choose a dynamic public variable and zero b_{194} directly. However, since the ANF representation of b_{194} is difficult to visualize, this has a few disadvantages: After we choose a cube to sum over, we need to guess all the secret expressions that are multiplied by terms of cube variables, and the complex ANF representation of b_{194} will force us to guess many expressions, which will unnecessarily increase the complexity of the attack. Moreover, since the ANF representation of b_{194} is of degree 9, many of the guessed expressions are expected to be non-linear, while ideally we would like to collect linear equations in order to be able to solve for the key bits efficiently. The process that we use to zero b_{194} is given below.

1. Use the description of Grain-128 to simplify the ANF representation of b_{194} by writing $b_{194} = b_{161}(b_{78}s_{161}) + P_{r1}$. In this form, b_{161} is the source polynomial, $b_{78}s_{161}$ is the target polynomial, and P_{r1} is some remainder polynomial with a simpler ANF representation compared to b_{194}.
2. The ANF representation of b_{161} is a simpler polynomial of degree 6 which contains 333 terms. Again, do not zero it directly, but write: $b_{161} = b_{128}(b_{45}s_{128}) + P_{r2}$, with b_{128} as the source polynomial with degree 3 and 26 terms. Choose v_0 as the dynamic variable and set it accordingly.
3. Now, the ANF representation of b_{161}, with v_0 set to its dynamic value is a polynomial of degree 2 which contains 47 terms. b_{161} can be zeroed by choosing v_{33} as a dynamic variable.

4. Recalculate the ANF of b_{194} with v_0 and v_{33} set to their dynamic values. It is now a polynomial of degree 5 which contains 1093 terms. Write $b_{194} = b_{134}b_{150} + P_{r3}$, and choose v_6 as the dynamic variable to zero b_{134}.
5. Write $b_{194} = b_{162} + P_{r4} = b_{129}(b_{46}s_{129}) + P_{r5}$ and choose v_1 as the dynamic variable to zero b_{129}.
6. Now, the symbolic form of b_{194} with v_0, v_{33}, v_6 and v_1 all set to their dynamic values, is a polynomial of degree 3 with 167 terms. Finally we choose v_{29} as the dynamic variable which can zero b_{194}.

B Appendix: Parameters for Our Attacks on Grain-128

The parameter sets for the different attacks are given in the tables below. As an example, we demonstrate the process of assigning values to the dynamic variables in Table 1. The process for the other tables is similar.

The first index of the "Dynamic Variables" list in Table 1 is 0 (i.e v_0). It is used to nullify the first state bit in the "State Bits Nullified" list (b_{128}). The symbolic form of v_0 is calculated as follows:

1. Initialize the state of Grain-128 with all the key bits assigned a distinct symbolic variable and all the IV bits set to 0, except the IV bits in the "Cube Indexes" row and v_0 which are assigned a distinct symbolic variable.
2. Clock the state once and obtain the symbolic value of the bit fed back into the NFSR (note that v_0 is a linear variable of the polynomial).
3. Delete the term v_0 from the symbolic form of this polynomial and assign v_0 the symbolic sum of the remaining terms, i.e. set $v_0 = x_3x_{67} + x_{11}x_{13} + x_{17}x_{18} + x_{27}x_{59} + x_{40}x_{48} + x_{61}x_{65} + x_{68}x_{84} + x_0 + x_2 + x_{15} + x_{26} + x_{36} + x_{45} + x_{56} + x_{64} + x_{73} + x_{89} + x_{91} + x_{96}$.

Next, we determine the symbolic value of v_1 (second in the "Dynamic Variables" list), according to the second state bit in the "State Bits Nullified" list (b_{129}). It is calculated in a similar way to v_0, except that we set v_0 to the dynamic value calculated in the previous step and set v_1 to a distinct symbolic variable. Finally we assign v_1 the symbolic value that is fed back to the NFSR after 2 initialization rounds (again, removing the linear term of v_1 from the symbolic form). We iteratively continue assigning v_6, v_{33} and v_{29} according to the symbolic values fed back to the NFSR after $7, 34$ and 67 clocks respectively, each time setting the previously determined dynamic variables to their dynamic values.

Table 1. Parameter set No.1 for the attack on Grain-128, given output bit 215

Cube Indexes	{3,28,31,34,40,50,51,52,54,62,63,64,65,66,67,68,69,80,92}
Dynamic Variables	{0,1,6,33,29}
State Bits Nullified	$\{b_{128}, b_{129}, b_{134}, b_{161}, b_{194}\}$
Expressions Retrieved	$\{x_{127}, x_{122} + 1\}$

Table 2. Parameter set No.2 for the attack on Grain-128, given output bit 215

Cube Indexes	{5,19,28,31,34,40,50,51,52,54,62,63,64,65,66,67,68,80,92}
Dynamic Variables	{0,1,6,33,29}
State Bits Nullified	$\{b_{128}, b_{129}, b_{134}, b_{161}, b_{194}\}$
Expressions Retrieved	$\{x_{69}, x_{23}\}$

Table 3. A Parameter set for the attack on Grain-128, given output bit 218

Cube Indexes	{19,20,28,29,30,31,41,45,53,54,55,63,64,65,66,67,68,69,89,92}
Dynamic Variables	{2,3,4,9,1,36,7,32}
State Bits Nullified	$\{b_{130}, b_{131}, b_{132}, b_{137}, s_{129}, b_{164}, b_{170}, b_{197}\}$
Expressions Retrieved	$\{x_{98}, x_{49}\}$

Table 4. Parameter set for the attack on Grain-128, given output bit 251

Cube Indexes	{11,12,13,15,17,21,24,26,27,29,32,35,38,40,43,46,49,51,52, 53,55,57,58,63,64,65,66,74,75,77,78,79,81,84, 86,87,95}
Dynamic Variables	{8,9,10,14,0,1,39,2,72,3,4,5,80,25,90,92,41,7,36,37,88,23,89,54}
Public Variables Set to 1	{82}
State Bits Nullified	$\{b_{136}, b_{137}, b_{138}, b_{142}, b_{128}, b_{129}, s_{129}, b_{130}, s_{130}, b_{131}, b_{132}, b_{133}, b_{148}, b_{153}, b_{158}, b_{160}, s_{162}, b_{163}, b_{164}, b_{165}, b_{174}, b_{186}, b_{197}, b_{230}\}$

Table 5. Parameter set for the attack on a weak key subset of the full Grain-128, given output bit 257

Cube Indexes	{0,3,5,10,11,13,14,15,17,19,21,23,26,31,34,35,37,39,40,43,45,48,49,51, 53,54,55,56,57,59,63,65,66,67,68,71,77,78,79,81,85,91,92,93,94,95}
Dynamic Variables	{9,1,12,4,7,6,8,89,2,29,83,25,69}
State Bits Nullified	$\{b_{137}, b_{129}, s_{133}, b_{132}, b_{135}, b_{134}, b_{136}, s_{168}, b_{169}, s_{150}, b_{176}, b_{192}, b_{236}\}$
Key Bits Set to 0	$\{x_{48}, x_{55}, x_{60}, x_{76}, x_{81}, x_{83}, x_{88}, x_{111}, x_{112}, x_{122}\}$