An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware

Itai Dinur[1], Tim Güneysu[2], Christof Paar[2],
Adi Shamir[1], and Ralf Zimmermann[2]

[1] Computer Science Department, The Weizmann Institute, Rehovot, Israel
[2] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany

Abstract. In this paper we describe the first single-key attack which can recover the full key of the full version of Grain-128 for arbitrary keys by an algorithm which is significantly faster than exhaustive search (by a factor of about 2^{38}). It is based on a new version of a cube tester, which uses an improved choice of dynamic variables to eliminate the previously made assumption that ten particular key bits are zero. In addition, the new attack is much faster than the previous weak-key attack, and has a simpler key recovery process. Since it is extremely difficult to mathematically analyze the expected behavior of such attacks, we implemented it on RIVYERA, which is a new massively parallel reconfigurable hardware, and tested its main components for dozens of random keys. These tests experimentally verified the correctness and expected complexity of the attack, by finding a very significant bias in our new cube tester for about 7.5% of the keys we tested. This is the first time that the main components of a complex analytical attack are successfully realized against a full-size cipher with a special-purpose machine. Moreover, it is also the first attack that truly exploits the configurable nature of an FPGA-based cryptanalytical hardware.

Keywords: Grain-128, stream cipher, cryptanalysis, cube attacks, cube testers, RIVYERA, experimental verification.

1 Introduction

Grain-128 [3] is a 128-bit variant of the Grain scheme which was selected by the eSTREAM project in 2008 as one of the three recommended hardware-efficient stream ciphers. The only single-key attacks published so far on this scheme which were substantially faster than exhaustive search were either on a reduced number of rounds or on a specific class of weak keys which contains about one in a thousand keys. In this paper we describe the first attack which can be applied to the full scheme with arbitrary keys. It uses an improved cube distinguisher with new dynamic variables, which makes it possible to attack Grain-128 with no restriction on the key. Its main components were experimentally verified by running a 50-dimensional cube tester for 107 random keys and discovering a very

D.H. Lee and X. Wang (Eds.): ASIACRYPT 2011, LNCS 7073, pp. 327–343, 2011.

strong bias (of 50 zeroes out of 51 bits) in about 7.5% of these keys. For these keys, we expect the running time of our new attack to be about 2^{38} times faster than exhaustive search, using 2^{63} bits of memory. Our attack is thus both faster and more general than the best previous attack on Grain-128 [1], which was a weak-key attack on one in a thousand keys which was only 2^{15} times faster than exhaustive search. However, our attack does not seem to threaten the security of the original 80-bit Grain scheme.

In order to develop and experimentally verify the main components of the attack, we had to run thousands of summations over cubes of dimension 49 and 50 for dozens of randomly chosen keys, where each summation required the evaluation of 2^{49} or 2^{50} output bits of Grain-128 (running the time-consuming initialization phase of Grain-128 for about 2^{56} different key and IV values). This process is hardware-oriented, highly parallelizable, and well beyond the capabilities of a standard cluster of PC's. We thus decided to implement the attack on a new type of special purpose hardware consisting of 128 Spartan-3 FPGAs.

Special-purpose hardware, i. e., computing machines dedicated to cryptanalytical problems, have a long tradition in code-breaking, including attacks against the Enigma cipher during WWII [15]. Their use is promising if two conditions are fulfilled. First, the complexity of the cryptanalytical problem must be in the range of approximately $2^{50} \dots 2^{64}$ operations. For problems with a lower complexity conventional computer clusters are typically sufficient, such as the linear cryptanalysis attack against DES [17] (which required 2^{43} DES evaluations), and more than 2^{64} operations are difficult to achieve with today's technology unless extremely large budgets are available. The second condition is that the computations involved are suited for customized hardware architectures, which is often the case in symmetric cryptanalysis. Both conditions are fulfilled for the building blocks of the Grain-128 attack described in this paper.

Even though it is widely speculated that government organizations have been using special-purpose hardware for a long time, there are only two confirmed reports about cryptanalytical machines in the open literature. In 1998, Deep Crack, an ASIC-based machine dedicated to brute-forcing DES, was introduced [16]. In 2006, COPACOBANA also allowed exhaustive key searches of DES, and in addition cryptanalysis of other ciphers [13]. However, in the latter case often only very small-scale versions of the ciphers are vulnerable. The paper at hand extends the previous work with respect to cryptanalysis with dedicated hardware in several ways. Our work is the first time that the main components of a complex analytical attack, i. e., not merely an exhaustive search, are successfully realized in a public way against a full-size cipher by using a special-purpose machine (previous attacks were either a simple exhaustive search sped up by a special-purpose hardware, or advanced attacks such as linear cryptanalysis which were realized in software on multiple workstations). Also, this is the first attack which makes use of the reconfigurable nature of the hardware. Our RIVYERA computer, consisting of 128 large FPGAs, is the most powerful cryptanalytical machine available outside government agencies (possessing more than four times as many logic resources as the COPACOBANA machine). This makes our attack an interesting case study about what type of cryptanalysis can be done with “university budgets” (as opposed to government budgets). As a final remark, it is worth noting that the same attack implemented on GPU clusters would require an extremely large number of graphic cards, which would not only require a very high budget but would consume considerably more electric energy to perform the same computations.

In the first part of this paper, we give the necessary background regarding Grain-128 and dynamic cube attacks and describe our new attack on Grain-128. In the second part of the paper, we present our FPGA implementation in detail.

2 Preliminaries

In this section we give a short description of Grain-128 [3], of cube testers (which were introduced in [2]), and of dynamic cube attacks (developed in [1]).

2.1 Description on Grain-128

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 1’s. The state is then clocked through 256 initialization rounds without producing an output, feeding the output back into the input of both registers.

2.2 Previous Results on Grain-128

All the previously published single-key attacks ([2], [5], [6], [7] and [8]) on Grain-128 which are substantially better than exhaustive search can only deal with simplified versions of the cryptosystem. In [9] a sliding property was used to speed-up 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 in this paper we concentrate on attacks in the single key model. The only significant known attack on the full version of Grain-128 in the single key model is given in [1], where dynamic cube attacks are used to break a particular subset of weak keys, which contains the 2^{-10} fraction of keys in which ten specific key bits are all zero. The attack is faster than exhaustive search in this weak key set by a factor of about 2^{15}. For the remaining 0.999 fraction of keys, there is no known attack which is significantly faster than exhaustive search.

2.3 Cube Testers

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). This polynomial is usually too large to write down or to manipulate in an explicit way, but its values can be evaluated by running the cryptographic algorithm as a black box. The cryptanalyst is allowed to tweak this master polynomial by assigning chosen values to the public variables (which result in multiple derived polynomials), but in single-key attacks he cannot modify the secret variables.

To simplify our notation, we ignore in the rest of this subsection 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. Compared to p, the algebraic degree of the superpoly is reduced by at least the number of variables in t_I, and its number of terms is smaller.

Cube testers [2] are related to high order differential attacks [11]. The basic idea behind them 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 $p(x_1, .., x_n)$. This simplified polynomial is more likely to exhibit non-random properties than the original polynomial P.

Cube testers work by evaluating superpolys of carefully selected terms t_I which are products of public variables, and trying to distinguish them from a random function. One of the natural 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 used to distinguish the cryptosystem from a random polynomial by testing whether the sum of output values over an appropriate boolean cube evaluates as often to one as to zero (as a function of the public bits which are not summed over).

2.4 Dynamic Cube Attacks

Dynamic Cube Attacks exploit distinguishers obtained from cube testers to recover some secret key bits. This is reminiscent of the way that distinguishers are used in differential attacks to recover the last subkey in an iterated cryptosystem. In static cube testers (and other related attacks such as the original cube attack [18], and AIDA [19]), 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 on some private variables. Each such function is carefully chosen in order to simplify the resultant superpoly and thus to amplify the expected bias (or the non-randomness in general) of the cube tester.

The basic steps of the attack are briefly summarized below (for more details refer to [1], where the notion of dynamic cube attacks was introduced).

A preprocessing stage: We first choose some polynomials that we want to set to zero at all the vertices of the cube, and show how to nullify them by setting certain dynamic variables to appropriate expressions in terms of the other public and secret variables. To minimize the number of evaluations of the cryptosystem, we choose a big cube of dimension d and a set of subcubes to sum over during the online phase. We usually choose the subcubes of the highest dimension (namely d and $d-1$), which are the most likely to give a biased sum. We then determine a set of e expressions in the private variables that need to be guessed by the attacker in order to calculate the values of the dynamic variables during the cube summations.

Note that these steps have to be done only once for each cryptosystem, and the chosen parameters determine the running time and success probabilities of the actual attack, in the same way that finding a good differential property can improve the complexity of differential attacks on a cryptosystem.

The online phase of the attack has two parts:

Online Step 1

1. For each possible vector of values for the e secret expressions, sum modulo 2 the output bits over the subcubes chosen during preprocessing with the dynamic variables set accordingly, and obtain a list of sums (one bit per subcube).
2. Given the list of sums, calculate its score by measuring the non-randomness in the subcube sums. The output of this step is a sequence of lists sorted from the lowest score to the highest (in our notation the list with the lowest score has the largest bias, and is thus the most likely to be correct in our attack).

Given that the dimension of our big cube is d, the complexity of summing over all its subcubes is bounded by $d2^d$ (using the Moebius transform [12]). Assuming that we have to guess the values of e secret expressions in order to determine the values of the dynamic variables, the complexity of this step is bounded by $d2^{d+e}$ bit operations. Assuming that we have y dynamic variables, both the data and memory complexities are bounded by 2^{d+y} (since it is sufficient to obtain an output bit for every possible vertex of the cube and for every possible value of the dynamic variables).

Online Step 2. 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 specific details of this step vary according to the attack.

2.5 A Partial Simulation Phase

The complexity of executing online step 1 of the attack for a single key is $d2^{d+e}$ bit operations and 2^{d+y} cipher executions. In the case of Grain-128, these complexities are too high and thus we have to experimentally verify our attack with a simpler procedure. Our solution is to calculate the cube summations in online step 1 only for the correct guess of the e secret expressions. We then calculate the score of the correct guess and estimate its expected position g in the sorted list of score values by assuming that incorrect guesses will make the scheme behave as a random function. Consequently, if the cube sums for the correct guess detect a property that is satisfied by a random cipher with probability p, we estimate that the location of the correct guess in the sorted list will be $g \approx \max\{p \times 2^e, 1\}$ (as justified in [1]).

3 A New Approach for Attacking Grain-128

The starting point of our new attack on Grain-128 is the weak-key attack described in [1] and we repeat it here for the sake of completeness. Both our new attack and the attack described in [1] use only the first output bit of Grain-128 (with index $i = 257$). The output function of the cipher is a multivariate polynomial of degree 3 in the state, and its only term of degree 3 is $b_{i+12}b_{i+95}s_{i+95}$. Since this term is likely to contribute the most to the high degree terms in the output polynomial, we try to nullify 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 it does not seem possible to nullify it in a direct way. Instead, the idea in [1] is to simplify (and not nullify) $b_{i+12}b_{i+95}s_{i+95}$, by nullifying b_{i-21} (which participated in the most significant terms of b_{i+12}, b_{i+95} and s_{i+95}). 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. The solution adopted in [1] was to assume that 10 specific key bits are set to 0. This leads to a weak-key attack on Grain-128 which can only attack a particular fraction of 0.001 of the keys.

In order to attack a significant portion of all the possible keys, we use a different approach which nullifies state bits that are produced at an earlier stage of the encryption process. This approach weakens the resistance of the output of Grain-128 to cube testers, but in a more indirect way. In fact, the output function is a higher degree polynomial which can be more resistant to cube testers compared to [1]. This forces us to slightly increase the dimension d from 46 to 50. On the other hand, since we choose to nullify state bits that are produced at an earlier stage of the encryption process, their ANF is relatively simple and thus the number of secret expressions e that we need to guess is reduced from 61 to 39. Since the complexity of the attack is proportional to $d2^{d+e}$, the smaller value of e more than compensates for the slightly larger value of d. Our new strategy thus yields not only an attack which has a significant probability of success for all the keys rather than an attack on a particular subset of weak keys, but also a better improvement factor over exhaustive search (details are given at the end of this section).

In the new attack we decided to nullify b_{i-54}. This simplifies the ANF of the output function in two ways: It nullifies the ANF of the most significant term of b_{i-21} (the only term of degree 3), which has a large influence on the ANF of the output. In addition, setting b_{i-54} to zero nullifies the most significant terms of b_{i+62} and s_{i+62}, simplifying their ANF. This simplifies the ANF of the most significant terms of b_{i+95} and s_{i+95}, both participating in the most significant term of the output function. In addition to nullifying b_{i-54}, we nullify the most significant term of b_{i+12} (which has a large influence on the ANF of the output, as described in the first paragraph of this section), $b_{i-104}b_{i-21}s_{i-21}$, by nullifying b_{i-104}.

The parameter set we used for the new attack is given in table 1. Most of the dynamic variables are used in order to simplify the ANF of $b_{i-54} = b_{203}$ so that we can nullify it using one more dynamic variable with acceptable complexity. We now describe in detail how to perform the online phase of the attack, given this parameter set. Before executing these steps, one should take the following preparation steps in order to determine the list of e secret expressions in the key variables we have to guess during the actual attack.

1. Assign values to the dynamic variables given in table 1. This is a very simple process which is described in Appendix B of [1] (since the symbolic values of the dynamic variables contain hundreds of terms, we do not list them here, but rather refer to the process that calculates their values).
2. Given the symbolic form of a dynamic variable, look for all the terms which are combinations of variables from the big cube.
3. Rewrite the symbolic form as a sum of these terms, each one multiplied by an expression containing only secret variables.
4. Add the expressions of secret variables to the set of expressions that need to be guessed. Do not add expressions whose value can be deduced from the values of the expressions which are already in the set.

When we prepare the attack, we initially get 50 secret expressions. However, after removing 11 expressions which are dependent on the rest, the number of expressions that need to be guessed is reduced to 39. We are now ready to execute the online phase of the attack:

1. Obtain the first output bit produced by Grain-128 (after the full 256 initialization steps) with the fixed secret key and all the possible values of the variables of the big cube and the dynamic variables given in table 1 (the remaining public variables are set to zero). The dimension of the big cube is 50 and we have 13 dynamic variables and thus the total amount of data and memory required is $2^{50+13} = 2^{63}$ bits.
2. We have 2^{39} possible guesses for the secret expressions. Allocate a guess score array of 2^{39} entries (an entry per guess). For each possible value (guess) of the secret expressions:
   (a) Plug the values of these expressions into the dynamic variables (which thus become a function of the cube variables, but not the secret variables).
   (b) Our big cube in table 1 is of dimension 50. Allocate an array of 2^{50} bit entries. For each possible assignment to the cube variables:
      i. Calculate the values of the dynamic variables and obtain the corresponding output bit of Grain-128 from the data.
      ii. Copy the value of the output bit to the array entry whose index corresponds to the assignment of the cube variables.
   (c) Given the 2^{50}-bit array, sum over all the entry values that correspond to the 51 subcubes of the big cube which are of dimension 49 and 50. When summing over 49-dimensional cubes, keep the cube variable that is not summed over to zero. This step gives a list of 51 bits (subcube sums).
   (d) Given the 51 sums, calculate the score of the guess by measuring the fraction of bits which are equal to 1. Copy the score to the appropriate entry in the guess score array and continue to the next guess (item 2). If no more guesses remain go to the next step.
3. Sort the 2^{39} guess scores from the lowest score to the highest.

To justify item 2.c, we note that the largest biases are likely to be created by the largest cubes, and thus we only use cubes of dimension 50 and 49. To justify item 2.d, we note that the cube summations tend to yield sparse superpolys, which are all biased towards 0, and thus we can use the number of zeroes as a measure of non-randomness. The big cube in the parameter set is of dimension 50, which has 16 times more vertices than the cube used in [1] to attack the weak key set. The total complexity of algorithm above is about $50 \times 2^{50+39} < 2^{95}$ bit operations (it is dominated by item 2.c, which is performed once for each of the 2^{39} possible secret expression guesses).

Given the sorted guess array which is the output of online step 1, we are now ready to perform online step 2 of the attack (which recovers the secret key without going through the difficult step of solving the large system of polynomial equations). In order to optimize this step, we analyze the symbolic form of the secret expressions: Out of the 39 expressions (denoted by $s_1, s_2, ..., s_{39}$), 20 contain only a single key bit (denoted by $s_1, s_2, ..., s_{20}$). Moreover, 18 out of the remaining $39 - 20 = 19$ expressions (denoted by $s_{21}, s_{22}, ..., s_{38}$) are linear combinations of key bits, or can be made linear by fixing the values of 45 more key bits. Thus, we define the following few sets of linear expressions: Set 1 contains the 20 secret key bits $s_1, s_2, ..., s_{20}$. Set 2 contains the 45 key bits whose choice simplifies $s_{21}, s_{22}, ..., s_{38}$ into linear expressions. Set 3 contains the 18 linear expressions of $s_{21}, s_{22}, ..., s_{38}$ after plugging in the values of the $20+45=65$ key bits of the first two sets (note that the set itself depends on the values of the key bits in the first two sets). Altogether, the first three sets contain $20+45+18=85$ singletons or linear expressions. Set 4 contains $128-85=45$ linearly independent expressions which form a basis to the complementary subspace spanned by the first three sets. Note that given the 128 values of all the expressions contained in the 4 sets, it is easy to calculate the 128-bit key.

Our attack exploits the relatively simple form of 38 out of the 39 secret expressions in order to recover the key using basic linear algebra:

1. Consider the guesses from the lowest score to the highest. For each guess:
   (a) Obtain the value of the key bits of set 1, $s_1, s_2, ..., s_{20}$.
   (b) For each possible possible values of the 45 key bits of set 2:
      i. Plug in the (current) values of the key bits from sets 1 and 2 to the expressions of $s_{21}, s_{22}, ..., s_{38}$ and obtain set 3.
      ii. Obtain the values of the linear expressions of set 3 from the guess.
      iii. From the first 3 sets, obtain the 45 linear expressions of set 4 using Gaussian Elimination.
      iv. For all possible values of the 45 linear expressions of set 4 (iterated using Gray Coding to simplify the transitions between values):
         A. Given the values of the expressions of the 4 sets, derive the secret key.
         B. Run Grain-128 with the derived key and compare the result to a given (known) key stream. If there is equality, return the full key.

This algorithm contains 3 nested loops. The loop of item 1 is performed g times, where g is the expected position of the correct guess in the sorted guess array. The loop of item 1.b is performed 2^{45} times per guess. The loop of item 1.b.iv is performed 2^{45} per iteration of the previous loop. The loop of item 1.b contains linear algebra in item 1.b.iii whose complexity is clearly negligible compared to the inner loop of item 1.b.iv, which contains 2^{45} cipher evaluations. In the inner loop of step 1.b.iv (in item 1.b.iv.A) we need to derive the 128-bit key. In general, this is done by multiplying a 128×128 matrix with a 128-bit vector that corresponds to the values of the linear expressions. However, note that 65 key bits (of sets 1 and 2) are already known. Moreover, since we iterate the values of set 4 using Gray Coding (i. e., we flip the value of a single expression per iteration), we only need to perform the multiplication once and then calculate the difference from the previous iteration by adding a single vector to the previous value of the key. This optimization requires a few dozen bit operations, which is negligible compared to running Grain-128 in item 1.b.iv.B (which requires at least 1000 bit operation). Thus, the complexity of the exhaustive search per guess is about $2^{45+45} = 2^{90}$ cipher executions, which implies that the total complexity the algorithm is about $g \times 2^{90}$.

The attack is worse than exhaustive search if we have to try all the 2^{39} possible values of g, and thus it is crucial to provide strong experimental evidence that g is relatively small for a large fraction of keys. In order to estimate g, we executed the online part of the attack by calculating the score for the correct guess of the 39 expression values, and estimating how likely it is to get such a bias for incorrect guesses if we assume that they behave as random functions. We performed this simulation for 107 randomly chosen keys, out of which 8 gave a very significant bias in which at least 50 of the 51 cubes sums were zero. This is expected to occur in a random function with probability $p < 2^{-45}$, and thus we estimate that for about 7.5% of the keys, $g \approx \max\{2^{-45} \times 2^{39}, 1\} = 1$ and thus the correct guess of the 39 secret expressions will be the first in the sorted score list (additional keys among those we tested had smaller biases, and thus a larger g). The complexity of online step 2 of the attack is thus expected to be about 2^{90} cipher executions, which dominates the complexity of the attack (the complexity of online step 1 is about 2^{95} bit operations, which we estimate as $2^{95-10} = 2^{85}$ cipher executions). This gives an improvement factor of 2^{38} over the 2^{128} complexity of exhaustive search for a non-negligible fraction of keys, which is significantly better than the improvement factor of 2^{15} announced in [1] for the small subset of weak keys considered in that attack. We note that for most additional keys there is a continuous tradeoff between the fraction of keys that we can attack and the complexity of the attack on these keys.

Table 1. Parameter set for the attack on the full Grain-128, given output bit 257

Cube Indexes	{0,2,4,11,12,13,16,19,21,23,24,27,29,33,35,37,38,41,43,44,46, 47,49,52,53,54,55, 57,58,59,61,63,65,66,67,69,72,75,76,78,79,81,82,84,85,87,89,90,92,93}
Dynamic Variables	{31,3,5,6,8,9,10,15,7,25,42,83,1}
State Bits Nullified	$\{b_{159}, b_{131}, b_{133}, b_{134}, b_{136}, b_{137}, b_{138}, b_{145}, s_{135}, b_{153}, b_{170}, b_{176}, b_{203}\}$

4 Description of the Dedicated Hardware Used to Attack Grain-128

Cube attacks and testers are notoriously difficult to analyze mathematically. To test our attack experimentally and to verify its complexity, we had to try dozens of random keys, and thus to run thousands of cube summations of dimension 49 and 50 for multiple random keys. This is only marginally feasible on a large cluster of PCs, which are ill-suited for performing computations relying heavily on bit-permutations as needed for this kind of attack. We thus decided to experimentally verify our attack on dedicated reconfigurable hardware.

4.1 Architectural Considerations

We start with an evaluation of the online phase of the attack (for the correct guess of the 39 secret expression values) regarding possible optimizations in hardware. To get a better understanding of our implementation, we describe the basic work-flow in Figure 1: The software implementation of the attack uses a parameter set as input, e. g., the cube dimension, the cube itself, a base IV and the number of keys to attack. It selects a random key to attack and divides the big cube into smaller worker cubes and distributes them to worker threads running in parallel. Please note that for simplicity the figure shows only one worker. If 2^w workers are used, the iterations per worker are reduced from 2^d to 2^{d-w}.

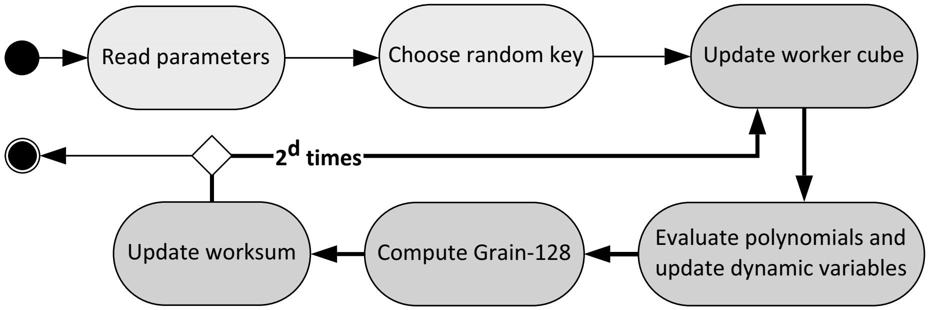

Fig. 1. Cube Attack — Program flow for cube dimension d

The darker nodes and the bold path show the steps of each independent thread: As each worker iterates over a distinct subset of the cube, it evaluates polynomials on the worker cube (dynamic variables) and updates the IV input to Grain-128. Using the generated IV and the random key, it computes the output of Grain-128 after the initialization phase. With this output, the thread updates an intermediate value — the worker sum — and starts the next iteration. In the end, the software combines all worker sums, evaluates the result and can chose a new random key to start again.

With a cube of dimension d, the attack on one key (for the correct guess of the 39 secret expression values) computes the first output bit of Grain-128 2^d times. Thus, in order to speed-up the attack, it is necessary to implement Grain-128 as efficiently as possible. The design of the stream cipher is highly suitable for hardware implementations: It consists mainly of two shift registers and some logic cells. As already proposed for cube testers on Grain-128 in [4], a fast and small FPGA implementation is a very good choice in comparison to a (bit-sliced) software implementation.

To create an independent worker on the FPGA, it is also required to implement the IV generation. To estimate the effort of building a full worker in hardware, we need to know how many dynamic inputs we have to consider: While dynamic modifications, e. g., iterating over arrays with dynamic step sizes, pose no problems in software, they can be very inefficient in hardware.

In order to compute the cipher, we need a key and an IV. The value of the key varies, as it is chosen at random. The IV is a 128 bit value, where each bit utilizes one of three functions: it is either a value given by the base IV provided by the parameter set, part of the (worker) cube or a dynamic variable. As the function of each bit is modified not only per parameter set, but also when assigning partial cubes to different workers, this input also varies. The first two functionalities are both restricted and can be realized by simple multiplexers in hardware. The dynamic variable on the other hand stores the result of a polynomial. As we have no set of pre-defined polynomials and they are derived at runtime, every possible combination of boolean functions over the worker cube (and thus over the complete 128 bits) must be realized. Even with tight restrictions like a maximum of terms per monomial and monomials per polynomial, it is impossible to provide the reconfigurable structure in hardware.

As a consequence, a fully dynamic approach leads to extremely large multiplexers and thus to very high area consumption on the FPGA, which is prohibitively slow. The completely opposite approach would be to utilize the complete area of an FPGA for massive parallel Grain-128 computations without additional logic. In this case, the communication between the host and the FPGA will be the bottleneck of the system and the parallel cores on the FPGA will idle.

For our attack, we use the RIVYERA special-purpose hardware cluster described in greater detail in Appendix A. For the following design decisions we remark that RIVYERA provides 128 powerful Spartan-3 FPGAs, which are tightly connected to an integrated server system powered by an Intel Core i7 920 with 8 logical CPU cores. This allows us to utilize dedicated hardware and use a multi-core architecture for the software part.

In order to implement the attack on the RIVYERA and benefit from its massive computing power, we propose the following implementation. Figure 2 shows the design of the modified attack. The software design is split into two parts: We use all but one core of the CPU to generate attack specific bitstreams, i. e., configuration files for the FPGAs, in parallel to prepare the computation on the FPGA cluster. Each of these generated designs configures the RIVYERA for a complete attack on one random key provided by the host PC. As soon as one bitstream was generated and waits in the queue, the remaining core programs all 128 FPGAs with it, starts the attack, waits for the computation to finish and stores the results.

In contrast to the first approach, which uses the generic structure realizable in software, we generate custom VHDL code containing constant settings and fixed boolean functions of the polynomials derived from the parameter set and the provided key. Building specific configuration files for each attack setup allows us to implement as many fully functional, independent, parallel workers as possible without the area consumption of complex control structures. In addition, only a single 7-bit parameter is necessary at runtime - to split the workspace between all 128 FPGAs - to start the computation and receive a d-bit return value. This efficiently circumvents all of the problems and overhead of a generic hardware design at the cost of rerunning the FPGA design flow for each parameter/key pair.

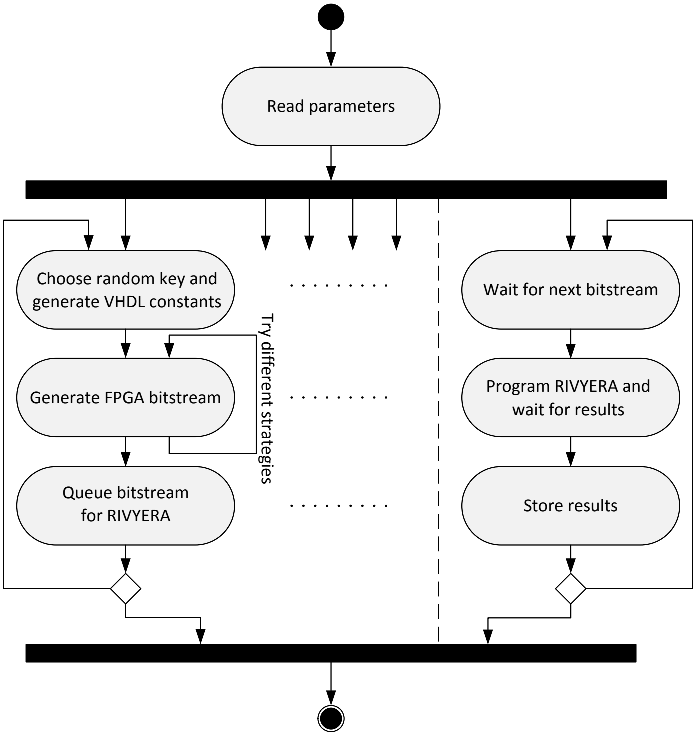
Read parameters
Choose random key and generate VHDL constants
Generate FPGA bitstream
Queue bitstream for RIVYERA
Try different strategies
Wait for next bitstream
Program RIVYERA and wait for results
Store results

Fig. 2. Cube Attack using RIVYERA

Please note that in this approach the host software modifies a basic design by hard-coding conditions and adjusting internal bus and memory sizes for each attack. We optimized the basic layout as much as possible, but the different choices of polynomial functions lead to different combinatorial logic paths and routing decisions, which can change the critical path in hardware. As the clock frequency is linked to the critical path, we implemented different design strategies as well as multiple fall-back options to modify the clock frequency constraints in order to prevent parameter/key pairs from resulting in an invalid hardware configurations.

4.2 Hardware Implementation Results

In this section, we give a brief overview of the implementation and present results. As the total number of iterations for one attack (for the correct guess of the 39 secret expression values) is 2^d, the number of workers for an optimal setup has to be a power of two. Considering the area of a Spartan-3 5000 FPGA, we chose to implement a set of 2^4 independent workers per FPGA.

Figure 3 shows the top level overview. As mentioned before, creating an attack specific implementation allows us to strip down the communication interface and

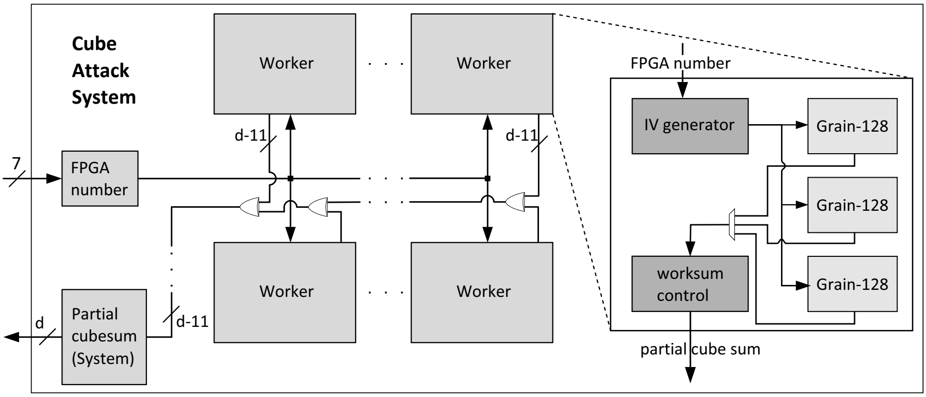

Fig. 3. FPGA Implementation of the online phase for cube dimension d

data paths to a minimum. This is very important as we cannot predict the impact of the (unknown) parameters and need to relax the design as much as possible.

Each of the workers consists of its own IV generator and controls three Grain-128 instances. The IV generator needs three clock cycles per IV and we need a corresponding number of Grain instances to process the output directly. As it is possible to run more than one initialization step per clock cycle in parallel, we had to find the most suitable time/area trade-off for the cipher implementation. Table 2 shows the synthesis results of our Grain implementation. In comparison, Aumasson *et al.* used 2^5 parallel steps, which is the maximum number of supported parallel steps without additional overhead, on the large Virtex-5 LX330 FPGA used in [4].

Table 2. Synthesis results of Grain-128 implementation on the Spartan-3 5000 FPGA with different numbers of parallel steps per clock cycle

Parallel Steps	2^0	2^1	2^2	2^3	2^4	2^5
Clock Cycles (Init)	256	128	64	32	16	8
Max. Frequency (MHz)	227	226	236	234	178	159
FPGA Resources (Slices)	165	170	197	239	311	418

The resulting attack system for the online phase — consisting of the software and the RIVYERA cluster — uses 16 workers per FPGA and 128 FPGAs on the cluster in parallel. This means that the number of Grain computations per worker is reduced to 2^{d-11}. The design ensures that each key can be attacked at the highest possible clock frequency, while it tries to keep the building time per configuration moderate.

Table 3 reflects the results of the generation process and the distribution of the configurations with respect to the different clock frequencies. It shows that the impact of the unknown parameters is predictable and that fallback strategies

Table 3. Results of the generation process for cubes of dimension 46, 47 and 50. The Duration is the time required for the RIVYERA cluster to complete the online phase. The Percentage row gives the percentage of configurations built with the given clock frequency out of the total number of configurations built with cubes of the same dimension.

Cube Dimension d	46			47	50	
Clock Frequency (MHz)	100	110	120	120	110	120
Configurations Built	1	7	8	6	60	93
Percentage	6.25	43.75	50	100	39.2	60.8
Online Phase Duration	17.2 min	15.6 min	14.3 min	28.6 min	4h 10 min	3h 49 min

are necessary. Please note that the new attack tries to generate configurations for multiple keys in parallel. This process — if several strategies are tried — may require more than 6 hours before the first configuration becomes available. Smaller cube dimensions, i. e., all cube dimensions lower than 48, result in very fast attacks and should be neglected, as the building time will exceed the duration of the attack in hardware. Further note that the duration of the attack increases exponentially in d, e. g., assuming 100 MHz as achievable for larger cube dimensions, $d = 53$ needs 1.5 days and $d = 54$ needs 3 days.

5 Conclusions

We presented the first attack on Grain-128 which is considerably faster than exhaustive search, and unlike previous attacks makes no assumptions on the secret key. While the full attack is infeasible, we can convincingly estimate its results by running a partial version in which all the e unknown secret expressions are set to their correct value. Due to its high complexity and hardware-oriented nature, the attack was developed and verified using a new type of dedicated hardware. Our experimental results show that for about 7.5% of the keys we get a huge improvement factor of 2^{38} over exhaustive search.

Acknowledgements. The authors thank Martin Ågren and the anonymous referees for their very helpful comments on this paper.

A Design and Architecture of the RIVYERA Cluster

In this work we employ an enhanced version of the COPACOBANA special-purpose hardware cluster that was specifically designed for the task of cryptanalysis [13]. This enhanced cluster (also known as RIVYERA [14]) is populated with 128 Spartan-3 XC3S5000 FPGAs, each tightly coupled with 32MB memory. Each Spartan-3 XC3S5000 FPGA provides a sea of logic resources consisting of 33,280 slices and 104 BRAMs enabling the implementation even of complex functions in reconfigurable hardware. Eight FPGAs are soldered on individual card modules that are plugged into a backplane which implements a global systolic ring bus for high-performance communication. The internal ring bus is further connected via PCI Express to a host PC which is also installed in the same 19" housing of the cluster. Figure 4 provides an overview of the architecture of the RIVYERA special purpose cluster.

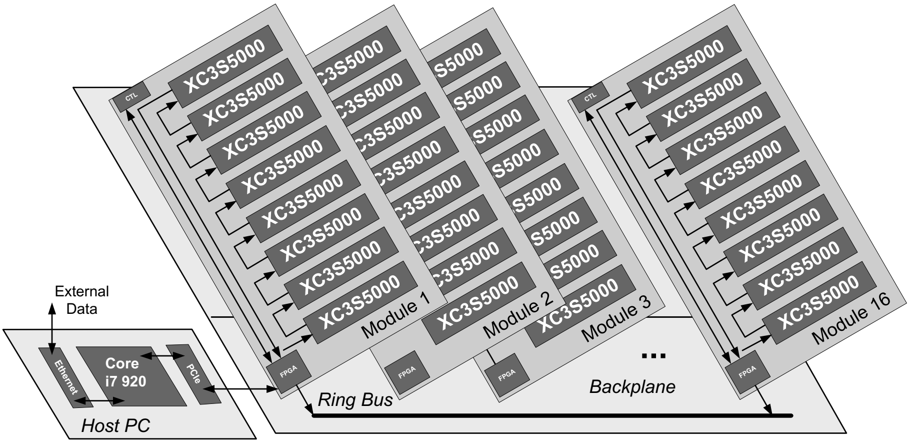

Fig. 4. Architecture of the RIVYERA cluster system

References

1. Dinur, I., Shamir, A.: Breaking Grain-128 with Dynamic Cube Attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011)
2. 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)
3. Hell, M., Johansson, T., Maximov, A., Meier, W.: A Stream Cipher Proposal: Grain-128. In: IEEE International Symposium on Information Theory, ISIT 2006 (2006)
4. 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: Workshop on Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS 2009, September 9-10 (2009)
5. 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)
6. 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)
7. 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)

8. 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)
9. De Cannière, C., Küçük, Ö., Preneel, B.: Analysis of Grain's Initialization Algorithm. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 276–289. Springer, Heidelberg (2008)
10. Lee, Y., Jeong, K., Sung, J., Hong, S.: 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 60th Birthday, pp. 227–233 (1994)
12. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall, pp. 285–286
13. Güneysu, T., Kasper, T., Novotný, M., Paar, C., Rupp, A.: Cryptanalysis with COPACOBANA. IEEE Transactions on Computers 57(11), 1498–1513 (2008)
14. Güneysu, T., Pfeiffer, G., Paar, C., Schimmler, M.: Three Years of Evolution: Cryptanalysis with COPACOBANA. In: Workshop on Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS 2009, September 9-10 (2009)
15. Budiansky, S.: Battle of Wits: the Complete Story of Codebreaking in World War II. Free Press (2000) ISBN: 9780684859323
16. Gilmore, J.: Cracking DES: Secrets of Encryption Research. Wiretap Politics & Chip Design. O'Reilly (July 1998)
17. Matsui, M.: The First Experimental Cryptanalysis of the Data Encryption Standard. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)
18. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413 (2007)
19. 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)