https://doi.org/10.1007/978-981-10-2741-3_2

Investigating Cube Attacks on the Authenticated Encryption Stream Cipher ACORN

Md Iftekhar Salam[1], Harry Bartlett[1], Ed Dawson[1], Josef Pieprzyk[1,2], Leonie Simpson[1], and Kenneth Koon-Ho Wong[1]

[1] Science and Engineering Faculty,
Queensland University of Technology,
Brisbane, QLD 4000, Australia
[2] Institute of Computer Science
Polish Academy of Sciences
Warsaw, Poland
{m.salam,h.bartlett,e.dawson,josef.pieprzyk,
lr.simpson,kk.wong}@qut.edu.au

Abstract. The cube attack is an algebraic attack that allows an adversary to extract low degree polynomial equations from the targeted cryptographic primitive. This work applies the cube attack to a reduced round version of ACORN, a candidate cipher design in the CAESAR cryptographic competition. The cube attack on 477 initialization rounds of ACORN can recover the 128 bit key with a total attack complexity of about 2^{35}. We have also shown that linear equations relating the initial state of the full version of ACORN can be be easily generated which can lead to state recovery attack with an attack complexity of about $2^{72.8}$.

Keywords: CAESAR, Authenticated Encryption, Cube Attack, ACORN

1 Introduction

The cube attack is an algebraic cryptanalysis method introduced by Dinur and Shamir at EUROCRYPT 2009 [1]. The attack is applicable to a wide variety of symmetric ciphers. In this paper, we analyze the applicability of the cube attack to the authenticated encryption (AE) stream cipher ACORN.

ACORNv1 [2] is a binary feedback shift register based AE stream cipher submitted to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [3] in May 2014. ACORNv1 is one of the cipher proposals selected for the second round of CAESAR. In September 2015, a tweaked version named ACORNv2 [4] was included in the second-round submission of the CAESAR competition.

Both versions of ACORN use a 128-bit secret key and a 128-bit initialization vector. The cipher provides authentication and encryption functionality for the input message. Encryption is performed by XOR-ing the plaintext message bitstream with the binary keystream output by the keystream generation function.

Message authentication is provided by a 128-bit tag computed from the plaintext message, secret key and the initialization vector. The cipher also provides authentication but not encryption for associated data (AD) if required.

2 ACORN Specification

ACORN uses a 128-bit key, $K \in \{0,1\}^{128}$ and a 128-bit initialization vector, $V \in \{0,1\}^{128}$. The cipher takes a plaintext message $P \in \{0,1\}^{*}$ of arbitrary length l_p within the range $0 \leq l_p \leq 2^{64}$. The input to the cipher may also include associated data $D \in \{0,1\}^{*}$, again of arbitrary length l_d within the range $0 \leq l_d \leq 2^{64}$. The associated data does not require confidentiality and so is not encrypted, but an integrity mechanism is applied. The output of ACORN consists of ciphertext $C \in \{0,1\}^{l_p}$ and an authentication tag $T \in \{0,1\}^{64 \leq l_{tag} \leq 128}$, where l_{tag} denotes the size of the tag. The designer of the cipher strongly recommends the use of a 128-bit tag. The structure of ACORN is based on six binary linear feedback shift registers (LFSRs) of lengths 61, 46, 47, 39, 37 and 59, respectively, and an additional 4 bit register. This gives the cipher a total internal state, $S = \{s_0, \cdots, s_{292}\}$ of 293 bits.

Operations performed in the ACORN stream cipher can be divided into four phases: Initialization, Encryption, Tag Generation and Decryption & Tag Verification. The differences between ACORNv1 and ACORNv2 occur mainly in the initialization phase and in the varying feedback functions used in the specific rounds of different phases. For the rest of the paper, unless specifically mentioned, ACORN will refer to both versions: ACORNv1 and ACORNv2. Figure 1 shows the generic diagram of ACORN encryption and decryption procedure.

ACORN uses three functions: an output keystream generation function, f_z, a nonlinear feedback function, f, and a state update function. ACORN state update function uses the output of the nonlinear feedback function and the input message, M to update internal state of the LFSRs. Depending on the phase the cipher is in, the input message M can denote either the key, initialization vector, padding vectors, associated data or plaintext. All of the register stages in the internal state are updated linearly except for the last register stage, s_{292}. The last register stage, s_{292}, is updated by combining the output of the nonlinear feedback function f with the input message M. Different variants of the feedback function f are used in different phases of the cipher. The output keystream generation function, f_z takes input from several register stages and generates a single keystream bit output at each round of the encryption/ decryption phase and during the final l_{tag} rounds of the tag generation phase.

During the initialization phase, ACORN takes as its input message, M, a sequence formed by the concatenation of the key, initialization vector, padding bits and associated data. Note that, in the original description of ACORN the designer considered only loading of the key, initialization vector and the padding bits as part of the initialization phase. However, we consider the associated data loading process as part of the initialization phase, because no keystream bits are output until after that process is complete. The initialization phases

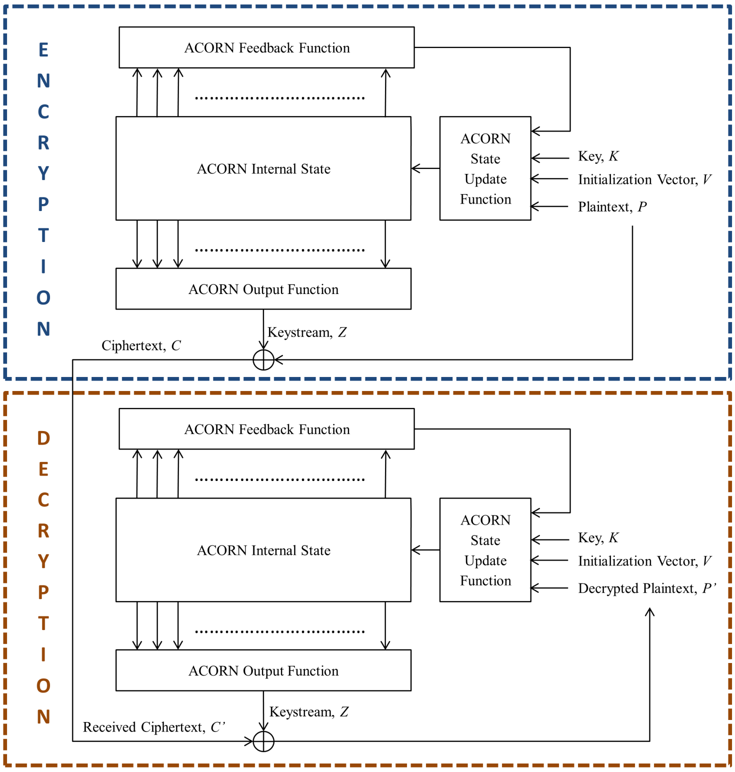

Fig. 1. Generic Diagram of ACORN

in ACORNv1 and ACORNv2 differ with respect to the feedback function used during the different rounds of the cipher, and also in the selection of the padding bits. For ACORNv1, the padding bits are constant whereas in ACORNv2 the padding bits are derived from the key bits. Both versions of ACORN have an initialization phase of $l_d + 2048$ rounds.

The encryption procedure is same for both ACORNv1 and ACORNv2. The cipher takes the plaintext, P, as the input message, M, and computes the output ciphertext, C, by XOR-ing the output keystream Z and plaintext P.

After all the plaintext bits have been processed, the keystream generator goes through a 1024 round finalization process. During these 1024 rounds, the input message M is a 1024-bit padding vector consisting of one bit with value 1 followed by 1023 bits of value 0. The difference in the finalization phase of ACORNv1 and ACORNv2 is the feedback function used during the different rounds. During the last l_{tag} rounds of the finalization phase, the output bits Z are computed and used as the tag.

The decryption procedure is the same as the encryption phase except that at each round the decrypted plaintext P' is computed by XOR-ing the ciphertext bit C with the corresponding keystream Z and this bit is fed back in to the internal state instead of the original plaintext. After all the ciphertext bits have been processed, the tag verification process first generates the tag using the same tag generation procedure as used earlier. Finally, the tag is verified by matching the generated tag with the received tag from the sender.

3 Cube Attack

The cube attack was introduced by Dinur and Shamir at EUROCRYPT 2009 [1]. The attack can be seen as a generalization of the Higher Order Differential attack [5] and the Algebraic IV differential attack (AIDA) [6]. The goal of the attack is to recover the secret key of a cryptosystem. In the original attack model, the adversary is given a blackbox that evaluates an unknown polynomial, Q constructed over l_k secret variables and l_v public variables. The adversary is also assumed to have access to a single output bit. This is a chosen plaintext attack which initially was applied to a reduced round version of the Trivium stream cipher [7], [1]. Later, several other symmetric ciphers were analyzed based on this attack [8], [9], [10].

A cube attack is a kind of algebraic attack which aims to recover the secret variable of a cryptographic scheme by manipulating and solving the polynomial equations defined by the scheme. Most of the symmetric cryptographic schemes can be defined by a single master polynomial over $GF(2)$, which contains some secret variables (e.g., secret key) and public variables (e.g., plaintext, ciphertext, initialization vector). The variants of the equations can be derived by changing these secret and public variables. The idea of the cube attack is to generate sufficient number of closely related low degree equations by manipulating the public variables. An adversary can then solve the generated low degree equations to recover the secret variables of the cryptosystem. The low degree equations are derived by evaluating the master polynomial over all the possible values of some specific public variables and then summing the resultant equations. This is called the cube attack since we sum over all the possible values of the n-dimensional Boolean cube. The observation found here is that, if an appropriate cube was chosen then summing over all the possible values of the cube will eliminate all the higher degree terms from the resultant equation. In general, the cube attack is performed in two phases: Preprocessing phase and Online phase.

3.1 Preprocessing Phase

In the preprocessing phase the adversary has access to both the public and secret variables. The goal of this phase is to construct linear equations in terms of the secret variables by using suitably chosen cubes. Let the blackbox polynomial, $Q(K, V)$ be constructed over the set of l_k secret variables in K and l_v public variables in V. Let $K = \{k_0, \cdots, k_{l_k-1}\}$ denote the set of the secret variables and $V = \{v_0, \cdots, v_{l_v-1}\}$ denote the set of the public variables of the cryptosystem. The adversary manipulates these secret and public variables to construct a linear variant of the blackbox polynomial, Q.

In the preprocessing phase, the adversary selects a cube of size l_c by randomly choosing l_c of the public variables $v_i \in V$. This selected subset of the public variables is known as the cube. The blackbox polynomial is evaluated and summed over all the possible values of this cube to determine if there exist any linear equation for that selection of the cube. The rest of the values of the public variables are usually set to zero. Let Q_c denote the equation obtained after summing over all the possible values of the cube.

At first, a linearity test is performed to verify that the equation obtained for the chosen cube is linear. This can be done by using the BLR test [11], which chooses two random input vectors $x, y \in \{0,1\}^{l_k}$ and then verifies that $Q[0] + Q[x] + Q[y] = Q[x+y]$. This is a probabilistic test which indicates Q_c is non-linear if the test fails. For randomly chosen x, y, the BLR test can detect non-linearity in Q_c with a probability of 0.5. Therefore, if Q_c passes the BLR test j times, then the probability of it being non-linear is 2^{-j}.

The adversary finds out the linear relation if the chosen cube passes the linearity test. To find out the linear coefficients of the secret variable k_i for a particular cube, the adversary sets k_i to 1 and all the other secret variables are set to zero. The public variables are set to zero everywhere except the l_c cube bits. Summing over all the possible values of the cube bits will give one bit of output which is the linear coefficient of k_i. The constant (either 0 or 1) in the linear equation for any cube is computed by summing over all the possible values of the cube with input values of zero everywhere except the chosen cube. This procedure is done to find out all the linear coefficient of the secret variables. The left hand side of the linear expression is then reconstructed once all the coefficients for the particular cube have been found. In the preprocessing phase, the adversary performs the procedure for different combinations of cubes and tries to find out sufficient number of cubes which output a linear equation. Note that not all choices of the cube will result in a linear equation.

3.2 Online Phase

In the online phase the adversary has access to the public variables and the cubes obtained during the preprocessing phase, to try to recover the secret variables. An adversary evaluates and sums the output of the master polynomial over all possible values of a cube to determine the right hand side of the corresponding linear equation. The resultant equations then can be solved by Gaussian elimination to recover the secret variables. Algorithm 1 provides a pseudo-code for the generic cube attack.

4 Cube Attack on ACORN

This section provides a description of the application of the cube attack to the authenticated encryption stream cipher ACORN. The attack can be performed either in the initialization phase, encryption phase or in the decryption phase. ACORN does not take any external input during the tag generation phase and

Algorithm 1 Algorithm for Cube Attack

```
Inputs: Output kesytream bits, Number of linearity test, Initial cube size, Number of cubes tested
Output: Secret variables of the cryptosystem
Preprocessing Phase
Select a random cube: Estimate the degree, $d$ of the polynomial, choose a initial cube size $l_c \leq d - 1$ and select a subset of $l_c$ public variables $v_i$
Do the linearity test and construct the linear equation
for Number of Cubes Tested do
  for Number of linearity test do
    if nonlinear then
      if Cubes Tested < Number of Cubes Tested then
        Select another cube of size $l_c$ and do the linearity test
      else
        Increase the number of cube variables $l_c$
      end if
    else
      Compute the coefficients of the secret variables by summing over all the possible values of the cube
      if all the coefficinets are zero then
        Select another subset of $l_c$ public variables
      else
        Output the coefficients and construct the linear equation
      end if
    end if
  end for
end for
Do the preprocessing phase till sufficient number of linear equations are generated
Online Phase
Find the right hand side of the linear equations:
for Each possible cube found in prepreocessing phase do
  Compute the output bit
  Sum all the output bits for all the possible values of the cube
end for
Solve the linear equations to recover the secret variables
```

therefore a cube attack is not applicable in this phase. In the following, we discuss the applicability of the cube attack to the initialization and encryption phases of ACORN.

4.1 Cube Attack during the Initialization Phase

In the initialization phase the key, initialization vector and associated data are loaded in to the internal state of ACORN. In general, the cube can be selected either from the input key, the initialization vector or the input associated data set. However, an adversary needs to have the ability to manipulate the key bits if the cube bits are chosen from the input key. On the other hand, the attack scenario falls under the nonce-reuse scenario when the cube bits are chosen from the associated data set. This is since multiple associated data will be authenticated using the same key and initialization vector, if we choose to select the cube bits from the the associated data set. The designer of ACORN does not claim any security when the same initialization vector is used with a given (the same) key to encrypt or authenticate multiple sets of data.

We consider the scenario where the cube is chosen from the initialization vector set. This requires the preprocessing phase to identify suitable cube bits chosen from the input initialization vector, which generate linear equations in terms of the key bits. Each of the linear equations is computed by summing the output function of ACORN for all the possible values of the corresponding cube. Therefore in the online phase, an adversary first needs to compute the right hand side of these equations for the corresponding cube. This follows the chosen plaintext model, where an adversary encrypts the plaintext with the key and chosen initialization vectors (varying the cube bits obtained from the preprocessing phase) and sums the output bits over n-dimensional Boolean cube to compute the right hand side of the corresponding equation. The secret key can be recovered by solving these equations if sufficient number of equations are generated. The attack requires an output bit from the ACORN output function for $n_c \times 2^{l_c}$ chosen initialization vectors where n_c and l_c represent the total number and the length of the cubes, respectively. So, the complexity of finding the right hand side of the linear equations during the online phase of the attack is no more than $n_c \times 2^{l_c}$. This will be followed by solving the equations using Gaussian elimination, which will require approximately n_c^3 operations when $n_c = 128$. Thus the total complexity of the attack is about $n_c \times 2^{l_c} + n_c^3$. If the equations generated are insufficient, i.e., $n_c < 128$, an adversary can achieve partial key recovery by solving the equations and the rest of the key bits can be found by exhaustive search.

4.2 Cube Attack during the Encryption Phase

In the encryption phase, plaintext bits are loaded in to the internal state of ACORN. Therefore plaintext can be considered as the public variables in this phase and cubes can be chosen from the input plaintext set. In this case, during the preprocessing phase an adversary manipulates the plaintext bits to generate linear equations in terms of the state bits. The linear equations are computed by summing the output function of ACORN for all the possible values of the corresponding cube. In the online phase, the adversary first needs to find the right hand side of these equations. This is a chosen plaintext attack, where an adversary encrypts the chosen plaintext (varying the cube bits obtained from the preprocessing phase) with the same key and initialization vector and sums the output bits over n-dimensional Boolean cube to compute the right hand side of the corresponding equation. Finally the initial state of the cipher can be recovered by solving the generated equations if sufficient equations are generated, i.e., $n_c = 293$. The attack requires the output bit from the ACORN output function for $n_c \times 2^{l_c}$ chosen plaintext vectors. Following the similar computation as shown in the previous section, the total attack complexity of the state recovery attack requires about $n_c \times 2^{l_c} + n_c^3$ operations.

Unlike the initialization phase, the encryption phase of ACORN does not need to go through a large number of rounds before producing the output bits. So the degree of the output polynomial is expected to be low if an adversary searches for the cube bits from the plaintext variables. Note that cube attack is more effective on low degree polynomials since a lower degree polynomial will require a smaller cube size. For ACORN an adversary can manipulate the plaintext bits to generate linear equation from the 58th round of the encryption phase when the first plaintext bit p_0 reaches to the output keysetream generation function f_z. The degree of the output polynomial of ACORN at the 58th round of encryption phase is only 3. Therefore, at that point of the encryption phase an adversary needs a cube size of 2 at most. This makes it trivial to find linear relations in terms of the state bits.

Note that this attack falls under the nonce-reuse scenario. This means an adversary chooses different set of cubes from the plaintext variables and evaluate the output function of ACORN with a fixed key and initialization vector. As mentioned earlier, the designer of ACORN does not claim any security when the same initialization vector is used with the same key to encrypt or authenticate multiple sets of data.

5 Experimental Results

Our experimental analysis of the cube attack on ACORN is conducted using Sage version 6.4.1 [12] on a standard 3.4GHz Intel Core i7 PC with 16GB memory. Experiments are performed on both the initialization and encryption phases of ACORN to identify suitable cubes. The following sections discuss the application of the cube attack to these two different phases of ACORN.

5.1 Cube Attack using the Initialization Vector

The initialization of ACORN has a large number of rounds: $l_d + 2048$ rounds. This has a minimum value of 2048 when there are no associated data inputs, i.e., $l_d = 0$. The degree of the output function of ACORN grows with the increase in the number of rounds and is expected to be quite high when the full 2048 rounds are used. So, the size of the cube is also expected to be high if we choose to find a cube after 2048 rounds. This requires a significant amount of computational time. We therefore tested the cube attack on reduced round versions of ACORN.

For a reduced round version of ACORNv1, with initialization phase of 500 rounds, we have a total of 21 linear equations in terms of the secret key bits. These equations are derived during the preprocessing phase of the cube attack. For this 8000 random cubes of size 2 were tested; however, none of these cubes passed the linearity test. So we increased the cube size and checked if there exist linear equations when the cube size is increased. No suitable cubes were found for a cube size of 3 and 4 after searching over 1000 random cubes. For a cube size of

5, 1000 randomly chosen cubes were tested among which 20 passed the linearity test, but for most of these cubes the linear coefficient of the secret variable was found to be 0, i.e., the cube summation results only in a constant. Only 3 cubes $\{v_{120}, v_{124}, v_{93}, v_7, v_{63}\}$, $\{v_{31}, v_{124}, v_{115}, v_{18}, v_{122}\}$, $\{v_{39}, v_{51}, v_{124}, v_{76}, v_{115}\}$ were found which give linear coefficients in terms of the secret variable. These cubes were obtained by running the preprocessing phase for about a week.

Note that there are possibly more cubes of size 5 after 500 initialization round, however to find more cubes an adversary needs to increase the cube search space. Instead of increasing the search space, we used the following method to obtain new cubes from the randomly chosen cubes. For a given randomly chosen cube, increase both the cube indices and the number of rounds by one. Choose the new cube as a valid one if it satisfies the linearity test. This reduces the time complexity in the preprocessing phase since the adversary does not need to search for a suitable random cube from a total possible search space of $\binom{128}{5}$. With this technique, we were able to find total 12 cubes of size 5 which gives 12 linear equations in terms of the secret key bits. Further experiments are performed by choosing the cubes from a smaller subset of the previously found cubes, i.e., the cubes of size 5 were selected from the subset of the cube indices $\{120, 124, 93, 7, 63, 31, 115, 18, 122, 39, 51, 76\}$. This technique gave us two more new cubes: $\{v_{39}, v_{93}, v_{31}, v_{124}, v_{122}\}$, $\{v_{120}, v_{51}, v_{124}, v_7, v_{63}\}$ at round 500 and an additional cube at round 503: $\{v_{42}, v_{125}, v_{21}, v_{127}, v_{118}\}$. Using these cubes, combined with the previously mentioned technique, we were able to find 9 more cubes which result in distinct linear equations. In total 21 linear equations are found after the 500 initialization round of ACORNv1. Therefore for ACORNv1 with 500 initialization rounds, adversary can guess 107 key bits and the remaining 21 bits can be found by solving these linear equations. This is dominated by the exhaustive search. In the following we discuss about reducing the dominance of the exhaustive search by reducing the initialization round further.

Observing the linear equations generated for round 500, we have found that the equations consist only of the first 99 variables of the key. Therefore, an adversary needs 99 independent linear equations to find out these 99 key bits. To reduce the dominance of the exhaustive search we have further examined on generating more linear relations by reducing the number of rounds and the cube indices. The new cube is considered valid if it still satisfies the linearity test. The experiment is repeated by reducing the number of rounds and the cube indices, till 99 or more cubes are found. These equations were found after reducing the round of ACORN initialization phase from 500 to 477.

During the online phase of the attack, an adversary first needs to find the right hand side of these linear equations. An adversary encrypts the plaintext with the key and chosen initialization vectors (varying the cube bits) and sums the respective output bits over the n-dimensional Boolean cube to compute the right hand side of the corresponding equation. This requires a total $99 \times 2^5 \approx 2^{11.6}$ chosen initialization vectors. Therefore, an adversary can expect to recover the key bits with complexity less than exhaustive search, if the initialization phase of ACORN is reduced to 477.

We implemented the attack to verify the online phase of the cube attack on the 477 initialization round version of ACORNv1. We started by computing the right hand side of each equation by summing the output bits for all the possible values of the corresponding cube. This requires access to $2^{11.6}$ specific output bits. The equations are then solved once the right hand side of all the equations are computed. We found that the solution was not unique for some of the key bits. Some of the solutions for the secret key bits are found to be dependent on the key bits: $k_{91}, k_{94}, k_{95}, k_{96}, k_{97}, k_{98}$. This is because some of the computed equations are linearly dependent. Guessing these six key bits correctly results in a unique solution which provides the key bits for $k_0, \cdots, k_{90}, k_{92}, k_{93}$. We also need to guess the rest of the 29 key bits to recover the whole key. Therefore, the attack require to guess total $29 + 6 = 35$ of the key bits. The linear equations can be solved using row reduction and back substitution, and the resulting solutions must then be checked with each of the 2^{29} possibilities for the remaining key bits. So, in practice the total attack complexity is about $2^{11.6} + 93^3 + 6 \times 93 \times 2^6 + 2^6 \times 2^{29} \approx 2^{35}$.

We have also performed the experiments for the reduced round variant of ACORNv2. For this experiment, we tested the same cubes that were found for ACORNv2 (as above). Interestingly we noticed that the same cube sets result in linear equations for ACORNv2 as well. This was also verified by running the online phase of the attack to recover the key bits.

5.2 Cube Attack using Plaintext

This section discusses the application of the cube attack to the encryption phase of ACORN. Unlike the initialization phase, the degree of the output polynomial during the encryption phase is expected to be comparatively low. Therefore cube attack using the plaintext can be applied to the full version of ACORN. In the following, we describe the cube attack to the encryption phase of ACORN.

For searching suitable cubes we first represented the internal state, S of ACORN symbolically and checked the output function Z while loading each bit of plaintext P. At each round of the plaintext loading we looked into the symbolic output function to determine whether it becomes linear when differentiating it with respect to a subset of plaintext variables. Any such set of plaintext, if it exists, is equivalent to the set of cubes in the cube attack. For the first 57 round of the plaintext loading phase no such set were found because the plaintext bit does not reach the output function till the 58th round. At the 58th round of the plaintext loading, we found that differentiating the output equation Z^{58} with respect to plaintext p_0 results in a linear equation. That also means that if we do a cube attack numerically, then summing the output polynomial Z^{58} over the cube p_0 will result a linear equation. However, when running the experiments symbolically, the software runs out of resource after a few number of rounds. The symbolic computation with Sage was able to successfully compute the output function until 148th round of the plaintext loading phase. This method gave us 103 linear equations with 293 unknowns. This is not sufficient to reduce the complexity below the exhaustive search of 2^{128}.

To find more cubes that produce linear equations we performed the experiment numerically. We started with the cubes $\{p_0\}$, $\{p_0, p_7\}$ and $\{p_0, p_{21}\}$ for round 58, 107 and 121, respectively, which are found using the symbolic computation. We then used the following method to obtain a new cube: Increase the cube indices and the number of round by one and choose the new cube as a valid one if it satisfies the linearity test. 100 linearity tests were performed for each of the cube. With this technique, we have obtained 245 linear equations with 293 variables. Among the 245 cubes for these equations, 42 are of size 1 and the rest are of size 2.

In the online phase of the attack an adversary needs to compute the right hand side of these equations and then solve the equations to recover the state bits. An adversary first encrypts the chosen plaintexts (varying the cube bits) with the same key and initialization vector and sums the respective output bits over the n-dimensional Boolean cube to compute the right hand side of the corresponding equation. This requires about $42 \times 2^1 + 203 \times 2^2 \approx 2^{9.81}$ chosen plaintext bits to find the right hand side of these 245 equations.

To verify the online phase of the attack, we implemented the attack by solving these linear equations. We first computed the right hand side of each equation by summing the output bits for all the possible values of the corresponding cube. This requires an adversary to have access to $2^{9.81}$ specific output bits. We then represented these linear equations by a matrix and found that the rank of the matrix is 234. Applying Gaussian elimination to the underdetermined system yields a row reduced matrix that can be used with each guess of the 59 underdetermined variables to calculate the remaining 234 variables. This process enables us to recover all the state bits. Therefore in practice the total attack complexity is about $2^{9.81} + 234^3 + 59 \times 234 \times 2^{59} \approx 2^{72.8}$.

6 Conclusion

We applied the cube attack to the reduced round version of ACORN. Our analysis shows that cube attack can recover the secret key with a complexity of 2^{35} when the number of initialization rounds is reduced to 477. We have also tested and verified the attack by recovering the actual key bits for a randomly chosen key. The attack can be possibly extended to higher number of initialization rounds, but it will require a larger cube size which requires a search over very large cube spaces. It is difficult to evaluate the performance of the cube attack for larger versions of ACORN without knowing the suitable choices of the cube. Due to the high time complexity of searching larger cubes, our experiments were conducted only for smaller cube sizes. Also, note that the cubes identified for the 477 initialization rounds are only of size 5, whereas the degree of the output function after 477 round is expected to be much higher than 5. This suggests that the key and the initialization vector are not mixed properly yet after the 477 rounds; however, it would be interesting to check if this behaviour continues for higher rounds of the initialization phase as well. Currently, the cube attack seems to be impractical for the full round version of ACORN.

We have also shown that it is trivial to recover the state bits of the full round version of ACORN with complexity less than exhaustive search, if the same key and initialization vector is used to encrypt or authenticate multiple sets of input plaintext. This does not threaten the security of ACORN if it is used as the designers suggested.

Acknowledgements. Md Iftekhar Salam was supported by the QUT Postgraduate Research Award (QUTPRA), QUT Higher Degree Research Tuition Fee Sponsorship and QUT Excellence Top Up Scholarship. Josef Pieprzyk was supported by the Polish National Science Center Grant DEC-2014/15/B/ST6/05130.

References

1. Dinur, I. and Shamir, A., Cube Attacks on Tweakable Black Box Polynomials. In A. Joux (Ed.), Advances in Cryptology - EUROCRYPT 2009, Vol. 5479, pp. 278-299, Springer Berlin Heidelberg, 2009.
2. Wu, H., ACORN: A Lightweight Authenticated Cipher (v1). CAESAR Competition. Retrieved from http://competitions.cr.yp.to/round1/acornv1.pdf, Accessed 29 May 2015.
3. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available from: http://competitions.cr.yp.to/index.html, Accessed 10 September 2015.
4. Wu, H., ACORN: A Lightweight Authenticated Cipher (v2). CAESAR Competition. Retrieved from http://competitions.cr.yp.to/round2/acornv2.pdf, Accessed 10 September 2015.
5. Lai, X., Higher Order Derivatives and Differential Cryptanalysis. In R.E. Blahut, Costello , D.J., and Maurer, U.M., T. (Eds.), Communications and Cryptography: Two Sides of One Tapestry, Vol. 276, pp. 227-233, Springer US, 1994.
6. Vielhaber, M., Breaking One.Fivium by AIDA an Algebraic IV Differential Attack. IACR ePrint Archive. 2007/413. Retrieved from https://eprint.iacr.org/2007/413.pdf, Accessed 28 May 2016.
7. De Cannire, C. and Preneel, B., Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In S.K. Katsikas, et al. (Eds.), Information Security - ISC 2006, Vol. 4176, pp. 171 - 186, Springer Berlin Heidelberg, 2006.
8. Mroczkowski, P. and Szmidt, J., The Cube Attack on Courtois Toy Cipher. IACR ePrint Archive. 2009/497. Retrieved from https://eprint.iacr.org/2009/497.pdf, Accessed 17 June 2016.
9. Dinur, I. and Shamir, A., Breaking Grain-128 with Dynamic Cube Attacks. In A. Joux (Ed.), Fast Software Encryption (FSE), Vol. 6733, pp. 167 - 187, Springer Berlin Heidelberg, 2011.
10. Sarkar, S., Maitra , S., and Baksi, A., Observing biases in the state: case studies with Trivium and Trivia-SC. Designs, Codes and Cryptography, pp. 1-25, 2016.
11. Blum, M., Luby, M., and Rubinfeld, R., Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences, 47, pp. 579-595, 1993.
12. Stein, W., et al., Sage Mathematics Software (Version 6.4.1), The Sage Development Team, 2015, http://www.sagemath.org.