Investigating Cube Attacks on the Authenticated Encryption Stream Cipher MORUS

Iftekhar Salam*, Leonie Simpson*, Harry Bartlett*, Ed Dawson*, Josef Pieprzyk*† and Kenneth Koon-Ho Wong*
*Science and Engineering Faculty
Queensland University of Technology, Brisbane, QLD 4066, Australia
Email: {m.salam, lr.simpson, h.bartlett, e.dawson, josef.pieprzyk, kk.wong}@qut.edu.au
†Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland

***Abstract*—We investigated the application of cube attacks to MORUS, a candidate in the CAESAR competition. We applied the cube attack to a version of MORUS where the initialization phase is reduced from 16 steps to 4. Our analysis shows that the cube attack can successfully recover the secret key of MORUS-640 with a total complexity of about 2^{10} for this reduced version, and similarly for MORUS-1280 with complexity 2^9. Additionally, we obtained cubes resulting in distinguishers for 5 steps of the initialization of MORUS-1280; these can distinguish the cipher output function from a random function with complexity of 2^8. All our attacks are verified experimentally. Currently, the cube attack does not threaten the security of MORUS if the full initialization phase is performed.**

1. Introduction

The cube attack is an algebraic cryptanalysis method introduced by Dinur and Shamir [1]. The attack generalises the idea of Higher Order Differential Attack [2] and Algebraic IV Differential Attack [3]. This paper investigates the applicability of cube attacks to the authenticated encryption (AE) cipher MORUS.

MORUS [4], [5] is a family of AE stream cipher algorithms, and is a third-round candidate in the CAESAR competition [6]. There are three variants: MORUS-640-128, MORUS-1280-128 and MORUS-1280-256; where the first number represents the state size and the latter one represents the key size. The cipher provides both confidentiality and integrity assurance for the input data.

MORUS ciphers use a key K of either 128-bits or 256-bits. The initialization vector V is 128-bits, for all the variants of MORUS. Confidentiality of plaintext P is achieved by XOR-ing P with the keystream generated by the cipher to obtain the ciphertext C. MORUS provides integrity assurance for the plaintext P and associated data D by injecting P and D into the internal state and computing a tag T in terms of the internal state.

There is very little analysis of MORUS in the public literature. Mileva et al. [7] described the existence of distinguishers for MORUS under the nonce-reuse scenario and analysed the state update function for collisions, reporting collisions in the internal state of MORUS when an adversary is able to inject specific differences into both the internal state and external inputs. Dwivedi et al. [8] analysed MORUS-640 for SAT based state recovery and found the attack has a complexity of 2^{370} which is not feasible.

1.1. Notations and Operations

- $K = k_0 k_1 ... k_{l_{k-1}}$: The secret key of size l_k bits.
- V: The initialization vector of size 128-bit.
- M^t: The external input to the state at step t.
- P^t: The input plaintext at step t.
- D^t: The input associated data at step t.
- S^t: The internal state at step t.
- S^t_j: The internal state at the j^{th} round of step t.
- $S^t_{j,k}$: k^{th} element of state S^t_j.
- $\oplus$: Bit-wise XOR operation.
- $\otimes$: Bit-wise AND operation.
- $\lll w_i$: Rotation to the left by w_i bits, where $0 \leq i \leq 4$.
- Word: A sequence of 32 bits or 64 bits, for MORUS-640 and MORUS-1280, respectively.
- Block: A sequence of 128 bits or 256 bits, for MORUS-640 and MORUS-1280, respectively.
- $Rotl\_xxx\_yy(x, b_i)$: Divide a xxx-bit block x into 4 yy-bit words and rotate each word to the left by b_i bits, where $0 \leq i \leq 4$.

2. Description of MORUS

MORUS has 5 state elements $S_{0,0}, \cdots, S_{0,4}$ where each element is a register of length either 128 bits or 256 bits, for MORUS-640 and MORUS-1280, respectively. Operations performed in MORUS can be divided into five phases: 1) Initialization 2) Processing associated data 3) Encryption 4) Finalization 5) Decryption and tag verification

Note that there are two versions of MORUS which differ only in the finalization phase. In this paper we mainly investigate the initialization phase of MORUS, so the difference in the two versions does not affect our analysis. Figure 1 shows the different components of MORUS in generic form.

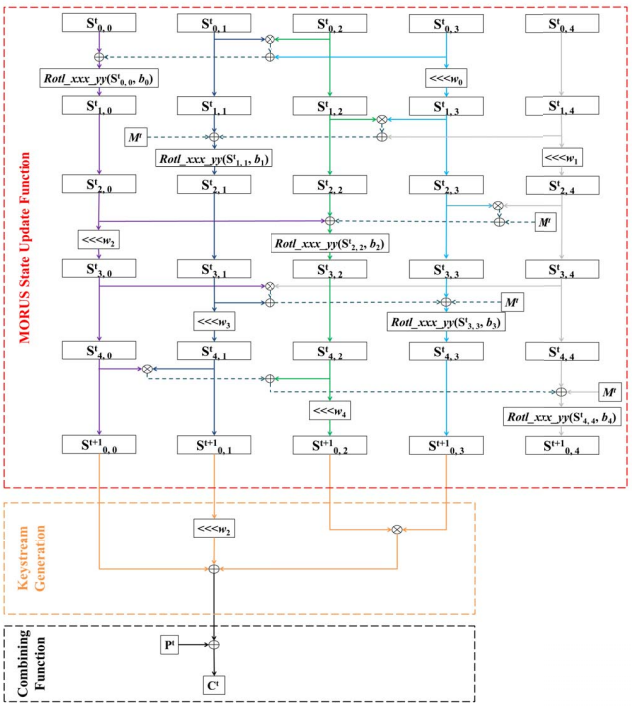

Figure 1. Generic Diagram of MORUS

One of the main component functions of MORUS is the state update function $Update(S^t, M^t)$. As shown in Figure 1, at each step t the state update function has 5 rounds with similar operations. Each round, two of the state elements $S^t_{j,k}$ are updated. The state update function takes input from the internal state bits and the external input M^t. Depending on the phase the cipher is in, M^t can be: all zero bits, the associated data, the plaintext, or a representation of the length of associated data and plaintext.

We describe here the initialization, associated data processing and encryption phase of MORUS as these are the phases where the cube attack is applicable.

2.1. Initialization

To start, the five state elements of MORUS are loaded with the key, initialization vector and specific constants. The interested reader can find the details in [4], [5]. After this, the state update function $Update(S^t, M^t)$ is applied 16 times with M^t set to zero. The cipher does not produce any output during this process. After these 16 updates the contents of state element $S^{t+16}_{0,1}$ are XORed with the key K. The state value at the end of this process is the initial internal state.

2.2. Processing Associated Data

This phase starts with the initial internal state. The associated data is divided into l_d blocks. At each step the associated data block D^t is used as the external input M^t. This process is continued l_d times to process all the associated data blocks, again without producing any output. This phase is omitted if there is no associated data.

2.3. Encryption

The input plaintext is processed after the associated data processing phase. The plaintext is divided into l_p blocks. At each step t, the cipher uses the keystream generation function to compute one output keystream block, and XORs this with P_t to form C_t. The plaintext block P^t is also used as the external input M^t to update the state of MORUS. This process is continued l_p times to process all the plaintext blocks.

3. Cube Attack

The goal of the cube attack is to generate and solve a system of linear equations to recover the secret key of a cryptosystem. The main observation of cube attack is that summing the cryptosystem's output polynomial over a specific set (cube) of public inputs (elements from V) might cancel out all the higher degree terms except terms associated with the monomials involving the cube variables; thus resulting in a linear equation. These polynomials constructed over the cube summation are called superpolys.

For an output polynomial of degree d the cube of size $d-1$ is guaranteed to produce a linear superpoly. Note that there may exist cubes of size less than $d-1$ which also result in linear superpolys.

Cube attack is performed in two phases: Preprocessing phase and Online phase. These are outlined below:

3.1. Preprocessing Phase

In the preprocessing phase, an adversary finds cubes resulting in linear superpolys. Let $f(K, V)$ define the underlying cryptographic polynomial, constructed over l_k secret variables in $K = \{k_0, \cdots, k_{l_k-1}\}$ and l_v public variables in $V = \{v_0, \cdots, v_{l_v-1}\}$. Generally, the full symbolic representation of $f(K, V)$ is very complex. So the cube attack needs to estimate the degree of $f(K, V)$.

Estimating the degree involves selecting cubes of different sizes and testing them using a probabilistic linearity test. This is performed using the BLR test [9] which selects two random inputs x, y and verifies that $f(0, V) + f(x, V) + f(y, V) = f(x + y, V)$. The superpoly f_c is nonlinear with probability 2^{-j}, if $f(K, V)$ passes the BLR test j times.

If a superpoly is linear with high probability, an adversary then needs to determine whether the linear superpoly will be useful for recovering the secret variables. Specifically, cubes that pass the linearity test must be checked to see whether they generate linear superpolys in terms of the secret variables or are just constants. The construction process of the linear superpoly f_c is carried out as follows:

- To find the constant: set all the secret variables to zero and the public variables to zero everywhere except the

cube bits. Sum over all possible values of the cube bits; the resulting bit is the constant for the linear superpoly.
- To find the coefficient of k_i: set all the secret variables to zero except k_i and the public variables to zero everywhere except the cube bits. Sum over all possible values of the cube bits; the resulting bit is the coefficient of k_i.

The steps involved for finding a valid cube in the preprocessing phase are summarized below:

1) Estimate degree d and select the cube size $l_c = d - 1$.
2) Select a random cube: Randomly select a subset of l_c public variables from $V = \{v_0, \cdots, v_{l_k-1}\}$ as the cube.
3) Perform the linearity test for the selected cube.
4) If the superpoly fails the linearity test then increase the cube size l_c by one and repeat from step 2.
5) If the superpoly passes the linearity test then compute its coefficients as described above.
6) If the constructed superpoly is a constant then decrease the cube size l_c by one and repeat from step 2.

To recover all the secret variables an adversary needs to find as many linearly independent superpolys as the total number of secret variables.

3.2. Online Phase

In the online phase, an adversary needs to determine the values of the linear superpolys for all the cubes found in the preprocessing phase. The steps involved in the this phase are outlined below:

1) For each cubes found in the preprocessing phase:
   - Evaluate $f(K, V)$ over all possible values of the selected cube and sum the resulting output.
   - Construct the linear equation by substituting the value of the linear superpoly.
2) Solve the resulting set of equations to recover K.

The adversary is assumed to have access to the ciphertext/keystream depending on the attack model. The plaintext has no effect on the cube summation if it is the same for all evaluations of a given cube; in such case an adversary can perform a ciphertext only attack. If this is not the case, the adversary needs access to both the ciphertext and plaintext (to access the keystream) and can perform a known plaintext attack. Both of these attack models require an adversary capable of manipulating the public variables.

3.3. Cube Testers

Cube testers were introduced as an extension of the cube attack [10]. The goal of cube testers is to distinguish the cipher output from the output of a random function, by observing whether the cube summation always results in a constant. An adversary can use these cubes as distinguisher.

4. Cube Attack on MORUS

This section discusses the applicability of the cube attack to different phases of MORUS. The output polynomial of MORUS is expected to contain variables from the key, initialization vector, associated data and plaintext. As a result, cube attacks on MORUS can be performed either in the initialization phase, the associated data loading phase, the encryption phase or the decryption phase. The goal of the attack is to recover the secret key if applied to the initialization phase. On the other hand, the goal of the attack is state recovery if applied to the associated data loading phase or encryption phase.

TABLE 1. ESTIMATED DEGREE ACCUMULATION OF MORUS-640

	Degree						
Step	S_0	S_1	S_2	S_3	S_4	Output	Attack Complexity
0	0	1	0	0	0	1	$2^0 \times 2^7$
1	1	1	1	1	2	2	$2^1 \times 2^7$
2	2	2	3	4	4	7	$2^6 \times 2^7$
3	5	7	8	9	12	17	$2^{16} \times 2^7$
4	15	17	21	27	32	48	$2^{47} \times 2^7$
5	38	48	59	70	86	128	$2^{127} \times 2^7$

Note that a cube attack on either the associated data loading phase or the encryption phase requires the attacker to choose the cube bits from the associated data and the plaintext bits, respectively. Therefore, the sum over all the possible values of the cube chosen from the associated data/plaintext needs to be calculated. This means that cube attack during these two phases requires authentication/encryption of multiple sets of associated data/plaintext using the same key and initialization vector. This falls under the nonce-reuse scenario, for which the designer of MORUS does not claim any security. Therefore we do not consider these further; instead, we focus our analysis on the initialization phase of MORUS.

4.1. Cube Attack on the Initialization Phase

In the initialization phase the adversary can only manipulate initialization vector bits. We consider the scenario where the cube bits are chosen from the initialization vector.

We start the **preprocessing phase** by finding appropriate cube bits in the initialization vector which generate linear superpolys, following the steps in Section 3.1. In the **online phase**, an adversary evaluates the output function for all cubes computed in the preprocessing phase to determine the value of the corresponding linear superpolys, following the steps in Section 3.2.

4.1.1. Estimated Complexity Analysis of Cube Attack.

The size of the cube is closely related to the degree d of the output function. Table 1 shows the estimated maximum degree accumulation for the MORUS-640 state contents and the output function.

The degree of the output function grows significantly with the increase in the number of steps. In each round the state update is expected to double the degree of the contents because of the application of the AND operation. Therefore the degree of the output polynomial is also expected to grow more than double at each step and is expected to reach the

TABLE 2. SEARCH SPACES FOR DIFFERENT CUBE SIZES OF MORUS

Cube Size	Search Space	Exhaustive Search Complexity
1	128	$2^7 \times 2^1$
2	8128	$2^{12.99} \times 2^2$
⋮	⋮	⋮
7	94525795200	$2^{36.46} \times 2^7$
⋮	⋮	⋮

maximum possible degree of 128 after only five steps of initialization phase. Table 1 also shows the increase in the complexity of cube attack with the increases in the number of steps. It seems that the cube attack would be infeasible just after few steps of initialization phase if the cube sizes are chosen as $d-1$.

However there may exist cubes which are of size less than $d-1$. This happens if the initialization vector is not mixed well with the secret key. This behaviour is expected at least in the first couple of steps of the initialization phase. We investigate existence of such lower dimension cubes.

Recall that we select cube bits from the 128-bit initialization vector. For a cube of size n, there are $\binom{128}{n}$ possible cube choices. Table 2 shows the increase in the number of possible search spaces with the increase in the cube sizes, and it is evident that it will not be possible to exhaustively test all the possible cubes over the whole search space for larger cube sizes.

5. Experimental Procedure and Results

We conducted experiments to analyse the feasibility of the cube attack on the initialization phase of MORUS. Our analysis is performed using Sage 6.4.1 [11] and Python 3.6, on a standard 3.4 GHz Intel Core i7 PC with 16 GB memory.

As discussed above, evaluating cubes of size $d-1$ requires high computational time for the full version of MORUS. So, we modified the MORUS design by considering fewer steps for the initialization phase.

Also, as illustrated in Table 2 for a reasonable cube size it is not possible to exhaustively test all cubes over the whole search space. Therefore we tested the existence of lower dimension cubes and the cubes are chosen randomly instead of searching over all possible cube choices. The chances of finding a cube with linear superpoly may increase with the increase in the numbers of cube tested; however, this also increases the time complexity of the preprocessing phase.

We conducted experiments on the reduced version of MORUS-640 and MORUS-1280-128, to find the existence of lower dimension cubes. In our experiments, the associated data length is set to zero to prevent degree accumulation.

5.1. Attack Algorithm

The steps for finding lower dimension cubes in the **preprocessing phase** of our experiments are outlined below:

1) Select the number of cubes to test n_c.
2) Select a cube size l_c. Continue step 3 to 6 until n_c randomly chosen cubes are tested.
3) Select a random cube of size l_c: Randomly select a subset of l_c initialization vector bits as the cube.
4) Do the linearity test using a randomly selected output bit of the first ciphertext/keystream block.
5) If the superpoly fails the linearity test: Discard the cube. Repeat the test from step 3 if the total number of cubes tested are less than n_c.
6) If the superpoly passes the linearity test: Construct the linear superpoly following the methods described in Section 3.1. Store the linear superpoly, the respective cube and the output index. Repeat from step 3 if the total number of cubes tested are less than n_c.

The **online phase** of our experiment follows the same steps as outlined in Section 3.2.

5.1.1. Finding New Cubes Using Existing Cube. To determine new cubes using the existing ones, we can increase/decrease the cube indices and the respective output index by one. The new cube is valid if it satisfies the linearity test and if the resultant linear superpoly is not a constant. When increasing the cube indices, the process is continued until the upper limit is reached for either any of the cube indices or the output index, i.e., any of the cube indices or the output index reaches the value 127 or 255 for MORUS-640 and MORUS-1280, respectively. When decreasing the cube indices, the process is continued until the lower limit is reached for either any of the cube indices or the output index, i.e., any of the cube indices or the output index reaches the value 0.

5.2. Cube Attack on MORUS-640

We applied the cube attack to MORUS-640 with an initialization phase of 4 steps. The **preprocessing phase** of the attack was performed following the steps mentioned in Section 5.1. To search for the cubes, we started with a cube size of 2. We tested over 20,000 random cubes of size 2; however, none of these random cubes passed the linearity test. We then increased the cube size to 3 and searched over 20,000 random cubes. Each cube are tested using 50 linearity tests. We found 2344 linear superpolys for cube size 3, among which only 192 are non-constant. Note that 103 of these 192 equations consists of only a single variable.

Further experiments reveals that a lot of these 192 cubes failed the linearity test when the number of tests are increased from 50 to 100. This indicates that 50 linearity tests are not sufficient. We expect that the superpolys found with the 100 linearity tests will be linear with high probability. We focused our experiments on the specific 103 cubes mentioned above and obtained only 31 cubes where the resultant superpoly passed 100 linearity tests. Example of some of these cubes are listed in Table 3.

We used the technique illustrated in Section 5.1.1 to find new cubes. Applying this technique to these 31 cubes, we obtained more than 300 linear superpolys. These superpolys cover all of the key bits except k_{22}, k_{53} and k_{118}.

TABLE 3. EXAMPLE OF CUBES FOR MORUS-640 (4-STEPS)

Cube Indices	Output Index	Superpoly
49, 13, 110	20	k_4
27, 107, 61	17	k_{18}
7, 6, 33	83	k_{44}
102, 83, 22	123	$k_{60} \oplus 1$
107, 104, 58	95	$k_{64} \oplus 1$
69, 21, 9	59	$k_{52} \oplus 1$
85, 10, 124	65	$k_{34} \oplus 1$
58, 68, 40	31	k_{49}
7, 104, 125	60	$k_{87} \oplus 1$
102, 119, 56	93	$k_{94} \oplus 1$
1, 66, 19	103	$k_{104} \oplus 1$
82, 106, 3	58	k_{120}

In the **online phase**, an adversary can select 125 independent linear superpolys with 125 variables. Most of these linear superpolys consist of only a single variable of the secret key. Only eight of the superpolys contains two variables. Thus, the complexity of solving these linear system of equation is negligible.

We implemented the **online phase** of the cube attack to verify the correctness of the linear superpolys. We started with a randomly generated 128-bit secret key and computed the values of all the linear superpolys by summing the output keystream bits for all the possible values (varying the corresponding initialization vector bits) of the respective cubes. The rest of the initialization vector bits are set to zero. To compute the value of the linear superpolys, we accessed $2^{9.97}$ keystream bits of the first output block constructed over $125 \times 2^3 \approx 2^{9.97}$ chosen initialization vectors. We then reconstructed and solved the linear equations. This correctly recovers 125 of the key bits. Note that three of the key bits do not appear in any of the linear superpolys found in our experiment. We need to guess these three secret key bits to recover the whole key. So the total attack complexity of the **online phase** is about $2^{9.97} + 2^3 \approx 2^{9.98}$.

5.3. Cube Attack on MORUS-1280

We applied the cube attack to MORUS-1280-128 with an initialization phase of 4 steps. In the **preprocessing phase** we conducted experiments following the steps mentioned in Section 5.1, to find cube variables from the initialization vector bits by randomly selecting cubes of different sizes and testing them for linearity. We started with cube size of 2 and tested 20,000 random cubes on MORUS-1280-128. Each of these cubes are tested for 100 linearity tests using a randomly selected output bit of the first ciphertext block.

We found 3947 cubes of size 2, where each cube passed at least 100 linearity tests. But most of these cubes resulted in a constant, only 13 cubes resulted in non-constant linear superpoly. Cubes resulting in non-constant linear superpolys are listed in Table 4.

We used the technique illustrated in Section 5.1.1 to find new cubes using the cubes listed in Table 4. We obtained total 408 linear superpolys with this technique. These linear superpolys cover all the 128 key bits of MORUS-1280-128.

TABLE 4. EXAMPLE OF CUBES FOR MORUS-1280-128 (4-STEPS)

Cube Indices	Output Index	Superpoly
7, 68	171	$k_{66} \oplus 1$
75, 64	158	k_{31}
79, 8	159	k_6
101, 117	127	$k_{58} \oplus 1$
66, 67	13	k_7
95, 20	1	$k_{62} \oplus 1$
53, 42	72	$k_{73} \oplus 1$
19, 75	183	k_{114}
78, 62	218	$k_{64} \oplus 1$
49, 48	251	$k_{117} \oplus 1$
89, 2	53	k_{27}
96, 40	42	k_{106}
0, 56	37	$k_{97} \oplus 1$

The cubes found for the reduced version of MORUS-640 and MORUS-1280-128 are of size 3 and 2, respectively. This indicates MORUS-1280 has a slower diffusion compare to MORUS-640. This is due to the differences in loading format of the internal state. In MORUS-1280, one of the state elements is loaded with all zero values which may have resulted in the comparatively slower diffusion.

In the **online phase**, an adversary can easily select 128 independent linear superpolys covering all the key bits and solve those to recover the key. All the linear superpolys obtained for MORUS-1280-128 consist of only a single variable. So the complexity of solving these equations are negligible.

To verify the correctness of the linear superpolys, we implemented the **online phase** of the cube attack on the 4 step initialization version of MORUS-1280. We started by selecting a randomly generated key and computed the value of all the linear superpolys by summing the output keystream bits for all the possible values (varying the corresponding initialization vector bits) of the respective cubes. The rest of the initialization vector bits are set to zero. To compute the sum over all the cube bits, we accessed 2^9 keystream bits of the first output block for $128 \times 2^2 = 2^9$ chosen initialization vectors. Following this we were able to reconstruct and solve the equations to recover the 128-bit secret key. The complexity of the attack is 2^9.

5.4. Cube Testers on MORUS-1280

We conducted experiments searching for cubes on modified version of MORUS with five or more initialization steps. In the **preprocessing phase** of these experiments, we did not find any cubes resulting in non-constant superpolys; however, for MORUS-1280 with five steps of initialization, we found cubes of size 9 resulting in constant superpoly.

This suggests that we can use these cubes of size 9 as distinguishers for MORUS-1280 with five initialization steps. Example of some of these cubes are listed in Table 5. These cubes passed at least 100 linearity tests.

In the **online phase**, an adversary can evaluate any of the cubes listed in Table 5. If the adversary is given access to 2^9 ciphertext/keystream for 2^9 chosen initialization vectors,

TABLE 5. EXAMPLE OF CUBES FOR MORUS-1280 (5 STEPS)

Cube Indices	Output Index
39, 30, 26, 110, 77, 56, 28, 70, 32	219
61, 29, 32, 46, 103, 115, 116, 26, 28	219
88, 25, 56, 39, 45, 70, 16, 13, 94	236
49, 109, 53, 78, 114, 127, 68, 59, 93	244
13, 111, 1, 12, 43, 28, 26, 120, 5	163

they can sum those ciphertext/keystream bits and use the sum to distinguish the output of the cipher from a randomly generated output. The complexity of distinguishing this modified version of MORUS-1280 is 2^9.

New distinguishers can be obtained using these existing cubes following the technique described in Section 5.1.1. Experimental analysis shows that with this technique we can find more cubes of size 9, resulting in distinguishers for five steps initialization version of MORUS-1280.

To obtain cubes of size 8 or less we extended our experiments by removing one or more of the cube variables from the cubes listed in Table 5. Evaluating the linearity of the superpolys for such cubes, we found some valid cubes of size 8. None of these cubes resulted in a non-constant superpoly and can only be used as a distinguisher. We did not find any cubes of size 7 or less. Thus, the best cubes for MORUS-1280 with five initialization steps can distinguish the cipher output from random with complexity 2^8.

6. Conclusion

We applied the cube attack to a reduced version of MORUS-640 and MORUS-1280-128, where the modification is a reduced initialization phase of 4 steps. The cube attack can successfully recover the key for reduced version of MORUS-640 and MORUS 1280-128 with complexity 2^{10} and 2^9, respectively. The cubes identified for reduced version of MORUS-640 and MORUS-1280-128 are of size 3 and 2, respectively; while the actual degree of the output equation after 4 steps for both of these two variants is much higher than 3. This means after 4 steps of the initialization phase there exists comparatively lower degree monomials in the output equation which does not appear together with any other monomials of that equation. This suggests that the key and initialization vectors are not mixed properly at this point of the initialization phase.

We also observed that the cubes obtained for the reduced version of MORUS-1280 are of smaller size compared to the ones obtained for MORUS-640. This suggests MORUS-1280 has a slower diffusion compared to MORUS-640.

We searched also for cubes with a higher number of steps in the initialization phase. We did not find any cubes resulting in non-constant linear superpolys when the number of initialization steps are more than 4 steps. However, for MORUS-1280 with 5 initialization steps, we obtained cubes which result in constant. These cubes can be used as distinguishers for MORUS-1280 with 5 steps of initialization.

For MORUS-1280 the cubes resulting in distinguisher for 5 steps initialization phase are of size 8 or 9; while the cubes for 4 steps initialization phase are of size 2. This shows the increases in the cube size are significant with the increase in the number of initialization steps.

It is hard to estimate the size of the cubes for higher number of initialization steps in MORUS without knowing the exact algebraic normal form of the output polynomial. We can choose cube size based on the estimated algebraic degree. However, the estimated growth in the degree is significant each step of the state update; thus the complexity of the attack is expected to grow exponentially which will be impractical just after a few steps of initialization phase. Currently, the cube attack does not threaten the security of MORUS if the full initialization phase is performed.

Acknowledgments

Iftekhar Salam was supported by QUTPRA, QUT HDR Tuition Fee Sponsorship & QUT Excellence Top Up Scholarship. Josef Pieprzyk was supported in part by a grant DEC-2013/09/D/ST6/03918, Polish National Science Center.

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] 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.

[3] 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.

[4] Wu, H. and Huang, T., The Authenticated Cipher MORUS (v1). CAESAR Competition. Retrieved from https://competitions.cr.yp.to/round1/morusv1.pdf, Accessed 23 February 2017.

[5] Wu, H. and Huang, T., The Authenticated Cipher MORUS (v2). CAESAR Competition. Retrieved from https://competitions.cr.yp.to/round3/morusv2.pdf, Accessed 23 February 2017.

[6] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available from: http://competitions.cr.yp.to/index.html, Accessed 10 September 2015.

[7] Mileva, A., Dimitrova, V., and Velichkov, V., Analysis of the Authenticated Cipher MORUS (v1). In E. Pasalic and Knudsen, L.R. (Eds.), Cryptography and Information Security in the Balkans, Vol. 9540, pp. 45-59, Springer International Publishing, 2015.

[8] Dwivedi, A. D., Klouček, M., Morawiecki, P., Nikolić, I., Pieprzyk, J., and Wójtowicz, S., SAT-based Cryptanalysis of Authenticated Ciphers from the CAESAR Competition. IACR ePrint Archive. 2016/1053. Retrieved from http://eprint.iacr.org/2016/1053.pdf, Accessed 03 March 2017.

[9] 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.

[10] Aumasson, J. P., Dinur, I., Meier, W., and Shamir, A., Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In O. Dunkelman (Ed.), Fast Software Encryption - FSE 2009, Vol. 5665, pp. 1-22, Springer Berlin Heidelberg, 2009.

[11] Stein, W., et al., Sage Mathematics Software (Version 6.4.1), The Sage Development Team, 2015, http://www.sagemath.org.