symmetry

MDPI

Article

Rotational Cryptanalysis of MORUS

Iftekhar Salam

School of Electrical and Computer Engineering, Xiamen University Malaysia, Sepang 43900, Malaysia; iftekhar.salam@xmu.edu.my

Abstract: MORUS is one of the finalists of the CAESAR competition. This is an ARX construction that required investigation against rotational cryptanalysis. We investigated the power of rotational cryptanalysis against MORUS. We show that all the operations in the state update function of MORUS maintain the rotational pairs when the rotation distance is set to a multiple of the sub-word size. Our investigation also confirms that the rotational pairs can be used as distinguishers for the full version of MORUS if the constants used in MORUS are rotational-invariant. However, the actual constants used in MORUS are not rotational-invariant. The introduction of such constants in the state update function breaks the symmetry of the rotational pairs. Experimental results show that rotational pairs can be used as distinguishers for only one step of the initialization phase of MORUS. For more than one step, there are not enough known differences in the rotational pairs of MORUS to provide an effective distinguisher. This is due to the XOR-ing of the constants that are not rotational-invariant. Therefore, it is unlikely for an adversary to construct a distinguisher for the full version of MORUS by observing the rotational pairs.

Keywords: authenticated encryption; CAESAR competition; MORUS; stream cipher; rotational cryptanalysis

Citation: Salam, I. Rotational Cryptanalysis of MORUS. *Symmetry* **2021**, *13*, 2426. https://doi.org/10.3390/sym13122426

Academic Editor: Yu-Chi Chen

Received: 14 October 2021
Accepted: 1 December 2021
Published: 15 December 2021

1. Introduction

MORUS [1,2] is a family of authenticated encryption stream cipher algorithms, one of the finalists in the CAESAR Authenticated Encryption (AE) competition [3]. 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 represents the key size. The cipher is intended to provide confidentiality and integrity assurance for the input data. This is a popular candidate from the CAESAR competition. Third-party security analysis is important for promoting and validating the design claims of MORUS. This paper provides a third-party analysis of MORUS subject to rotational cryptanalysis.

The MORUS family of ciphers has been subject a wide range of third-party security analyses. We first briefly discuss the existing security analyses of MORUS. Mileva et al. [4] proposed a distinguisher for MORUS-640 under the nonce-reuse scenario. This attack can analogously be used for MORUS-1280. They also reported collisions in the internal state of MORUS. However, an adversary must inject specific differences in the internal state to obtain the collisions. Dwivedi et al. [5] discussed the use of SAT solvers for state recovery with a complexity of 2^{370}. The complexity of this attack is beyond the designer's claimed security margin. Dwivedi et al. [6] also performed differential and rotational cryptanalysis with a reduced version of MORUS. The best result from this work was a theoretical key recovery attack against MORUS-1280-256, where the initialization phase was reduced to 18 rounds (3.6 steps). Salam et al. [7] applied cube attacks that obtained distinguishers with up to 4/5 steps for MORUS-640/1280 with negligible complexity. Kales et al. [8] and Vaudenay et al. [9] analyzed the security of MORUS under the nonce-reuse scenario. Their attacks included state recovery, key recovery, forgery with practical complexities and a small number of nonce-misuse queries. Salam et al. [10] described key recovery and forgery against MORUS using fault attacks. This work considered different fault models.

The best result from this work was a full key recovery with a transient fault model. Nozaki and Yoshikawa [11] investigated a power analysis attack against MORUS. Their work implemented the attack on an FPGA, which demonstrated the vulnerability of MORUS to power analysis. Ashur et al. [12] presented a linear correlation in the output of the full version of MORUS. This correlation can be used to distinguish the MORUS output keystream from randomness. Their work shows a plaintext recovery attack can be applied to MORUS-1280 using about 2^{152} blocks of data. They also presented a forgery attack on a reduced version of MORUS, where the finalization was reduced to three steps, and a key recovery attack under a nonce-reuse scenario for a reduced 10-step initialization. Li and Wang [13] analyzed division property and differential trails against MORUS. They reported 6/6.5-step integral distinguishers for MORUS-640/1280 and 4.5-step differential distinguishers for MORUS-1280. Shi and Guan [14] described a state recovery attack against MORUS under a nonce-reuse scenario. The attack reuses the nonce seven times to recover the internal state bits with practical complexity. Ye et al. [15] proposed a new existence terms detection method using the cube attack and applied it to MORUS. This method demonstrated key recovery attacks on 6/7-step MORUS-640-128 and integral distinguishers against 7-step MORUS-640-128/MORUS-1280-256. Shi et al. [16] proposed an algorithm to compute the correlation of quadratic Boolean functions. They applied the method to analyze the linear trails of MORUS. They identified a set of trails with correlation 2^{-38}, which led to distinguishing and message-recovery attacks for all versions of MORUS with a complexity of 2^{76}. Chen et al. [17] discussed a method to search for cubes based on the division property. This work demonstrated a cube attack on 4.4-step MORUS-640-128 with a complexity of $2^{27.91}$. He et al. [18] also investigated cube attacks against MORUS. The best result from this work used a cube of size 22 to apply a key recovery attack against 6-step MORUS-640-128.

Rotational cryptanalysis was applied [19] to ciphers composed of three operations: addition, rotation and XOR (ARX). MORUS is an ARX-like cipher, except that it uses bitwise multiplication instead of the addition operation. For ARX-like constructions, there is a need for analysis against rotational cryptanalaysis. Rotational cryptanalysis investigates the propagation of rotational pairs in the outputs of a cryptographic scheme for given rotational input pairs. The term rotational cryptanalysis was coined by Khovratovich and Nikolić in 2010 [19]; however, the attack technique was known and applied prior to this work by Biham [20]. To the best of our knowledge, there have been few third-party analyses of MORUS components and operations using rotational cryptanalysis. Therefore, we considered investigating the application of rotational cryptanalysis to MORUS, since it has similar ties to the ARX ciphers. In particular, we investigated different components and operations in MORUS and their rotational properties. Note that a rotational attack on MORUS was also investigated by Dwivedi et al. [6]. Their application of rotational cryptanalysis tried to recover the secret key of MORUS. They have developed a key recovery attack on round-reduced MORUS based on rotational cryptanalysis. On the other hand, we used rotational cryptanalysis to build a distinguisher for MORUS. Our approach of rotational cryptanalysis can build the distinguisher for just one step of the initialization phase of MORUS. Both of these works show that MORUS provides a large security margin against rotational cryptanalysis. The work by Dwivedi et al. [6] suggests that all the operations, except for the XOR-ing of constants, preserve the rotational pairs in MORUS for any arbitrary rotation distance. However, according to our analysis, this is not always true. We note that the $Rotl\_xxx\_yy(x, b)$ operation in MORUS does not preserve rotational pairs in all the bits for arbitrary rotation distance; instead, it preserves the rotational pairs in all the bits when the rotation distance is a multiple of the sub-word length yy. This does not affect their cryptanalysis, since it uses known rotational characteristics in specific bits, which are computed in a pre-computation phase.

2. Description of MORUS

The MORUS family of stream ciphers supports a key K of either 128-bits or 256-bits. The initialization vector V is 128 bits for all the variants of MORUS. It takes an input plaintext P of arbitrary length. Confidentiality is achieved by encrypting the plaintext P by XOR-ing with the output keystream generated by the cipher to obtain the ciphertext C. The cipher also takes associated data D of arbitrary length as input. The associated data are not encrypted. MORUS provides integrity assurance for both the plaintext P and associated data D. This is done by injecting the plaintext and associated data into the internal state of the cipher and computing a tag τ in terms of the internal state.

MORUS has five state elements $S_{0,0}, \cdots, S_{0,4}$. Each element is a register of length either 128 bits or 256 bits, for MORUS-640 and MORUS-1280, respectively. This gives a total internal state size of 640 or 1280 bits, for MORUS-640 and MORUS-1280, respectively. Before going into the details of MORUS structure and operation phases, we first introduce the notations and operations used in this paper, as below:

$\oplus$: Bit-wise XOR operation.
$\otimes$: Bit-wise AND operation.
$\|$: Concatenation
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.
$Rotl\_xxx\_yy(x, b)$: Divide a xxx-bit block x into 4 yy-bit words and rotate each word to the left by b bits.
$K = k_0 k_1 \ldots k_{l_{k-1}}$: The secret key of size l_k bits.
V: The initialization vector of size 128 bits.
$const_0$: A 128-bit constant `0x000101020305080d1522375990e97962` in hexadecimal format.
$const_1$: A 128-bit constant `0xdb3d18556dc22ff12011314273b528dd` in hexadecimal format.
M^t: The external input message to the state at step t.
P^t: The input plaintext block at step t.
D^t: The input associated data block at step t.
Z^t: The output keystream block at step t.
C^t: The output ciphertext block at step t.
τ: Authentication tag.
S^t: The internal state at step t.
S^t_j: The internal state at the jth round of step t.
$S^t_{j,k}$: kth element of state S^t_j.
$\overleftarrow{X}^r$: Rotation of the input X to the left by r bits.
X_{rotl_b}: $Rotl\_xxx\_yy(x, b)$ operation applied to X. xxx and yy assumed to be known from context.

2.1. Phases of Operation

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 the MORUS family of authenticated encryption cipher: MORUSv1 [1] and MORUSv2 [2]. The two versions differ only in the finalization phase. The general description provided in this paper is based on MORUSv2 [2]. Here we briefly describe the initialization of MORUS, as we investigated rotational cryptanalysis

on this phase. Interested readers are referred to the original specifications of MORUS for further details.

Initialization of MORUS

The initialization of MORUS starts with loading the key, initialization vector and two 128-bit constants $const_0$ and $const_1$ into the internal state in a particular format. The loaded state of MORUS-640 is defined as $(S_{0,0}^{t=0} \parallel S_{0,1}^{t=0} \parallel S_{0,2}^{t=0} \parallel S_{0,3}^{t=0} \parallel S_{0,4}^{t=0}) = (V \parallel K^{128} \parallel 1^{128} \parallel const_0 \parallel const_1)$. For MORUS-1280, two different key sizes may be used: a 128-bit key or a 256-bit key. The loaded state of MORUS-1280 with a 128-bit key is defined as $(S_{0,0}^{t=0} \parallel S_{0,1}^{t=0} \parallel S_{0,2}^{t=0} \parallel S_{0,3}^{t=0} \parallel S_{0,4}^{t=0}) = (V \parallel 0^{128} \parallel K^{128} \parallel K^{128} \parallel 1^{256} \parallel 0^{256} \parallel const_0 \parallel const_1)$. Similarly, the loaded state of MORUS-1280 with a 256-bit key is defined as $(S_{0,0}^{t=0} \parallel S_{0,1}^{t=0} \parallel S_{0,2}^{t=0} \parallel S_{0,3}^{t=0} \parallel S_{0,4}^{t=0}) = (V \parallel 0^{128} \parallel K^{256} \parallel 1^{256} \parallel 0^{256} \parallel const_0 \parallel const_1)$. After forming the loaded state, the state of MORUS is updated for 16 clocks (steps) using the state update function $Update(S^t, M^t)$. During these updates, the external input M^t is set to zero and the cipher does not produce any output. After these 16 updates, the content of state element $S_{0,1}^{t=16}$ is XORed with the key. The state obtained at the end of this process is the initial state of MORUS-640.

2.2. Functions Used in MORUS

The operations performed in different phases of MORUS are based on several component functions. These are:

1. State Update Function
2. Keystream Generation Function
3. Combining Function

2.2.1. State Update Function

One of the main component functions of MORUS is the state update function $Update(S^t, M^t)$. At each clock (step) t of the state update function, there are 5 rounds with similar operations. The operations in the state update function include AND, XOR and rotation operation. MORUS uses the bitwise left rotation $\overleftarrow{X}^{w_i}$, a simple rotation of the input X to the left by w_i bits, where $0 \leq i \leq 4$. It also uses the $Rotl\_xxx\_yy(x, b_i)$ operation which divides a xxx-bit block input x into 4 yy-bit words and rotates each word to the left by b_i bits, where $0 \leq i \leq 4$. The rotation constants b_i and w_i for different variants of MORUS are listed in Table 1.

Table 1. Rotation constants used in MORUS.

	MORUS-640	**MORUS-1280**
b_0	5	13
b_1	31	46
b_2	7	38
b_3	22	7
b_4	13	4
w_0	32	64
w_1	64	128
w_2	96	192
w_3	64	128
w_4	32	64

Figure 1 shows the operations involved in the state update function of MORUS in generic form. As shown in Figure 1, in each round, two of the state elements $S_{j,k}^t$ are updated using the operations $Rotl\_xxx\_yy(x, b_i)$ and $\overleftarrow{X}^{w_i}$. The state update function takes input from the internal state and external input M. Depending on the phase the cipher

is in, the external input M can be: all zero bits, the associated data, the plaintext or a representation of the length of the associated data and plaintext. During the initialization phase, the external input M^t is set to zero. The external inputs M^t are defined in terms of the plaintext P^t and the associated data D^t for the encryption and associated data loading phases, respectively. The external input M^t is defined in terms of the length of the plaintext and associated data for the finalization phase.

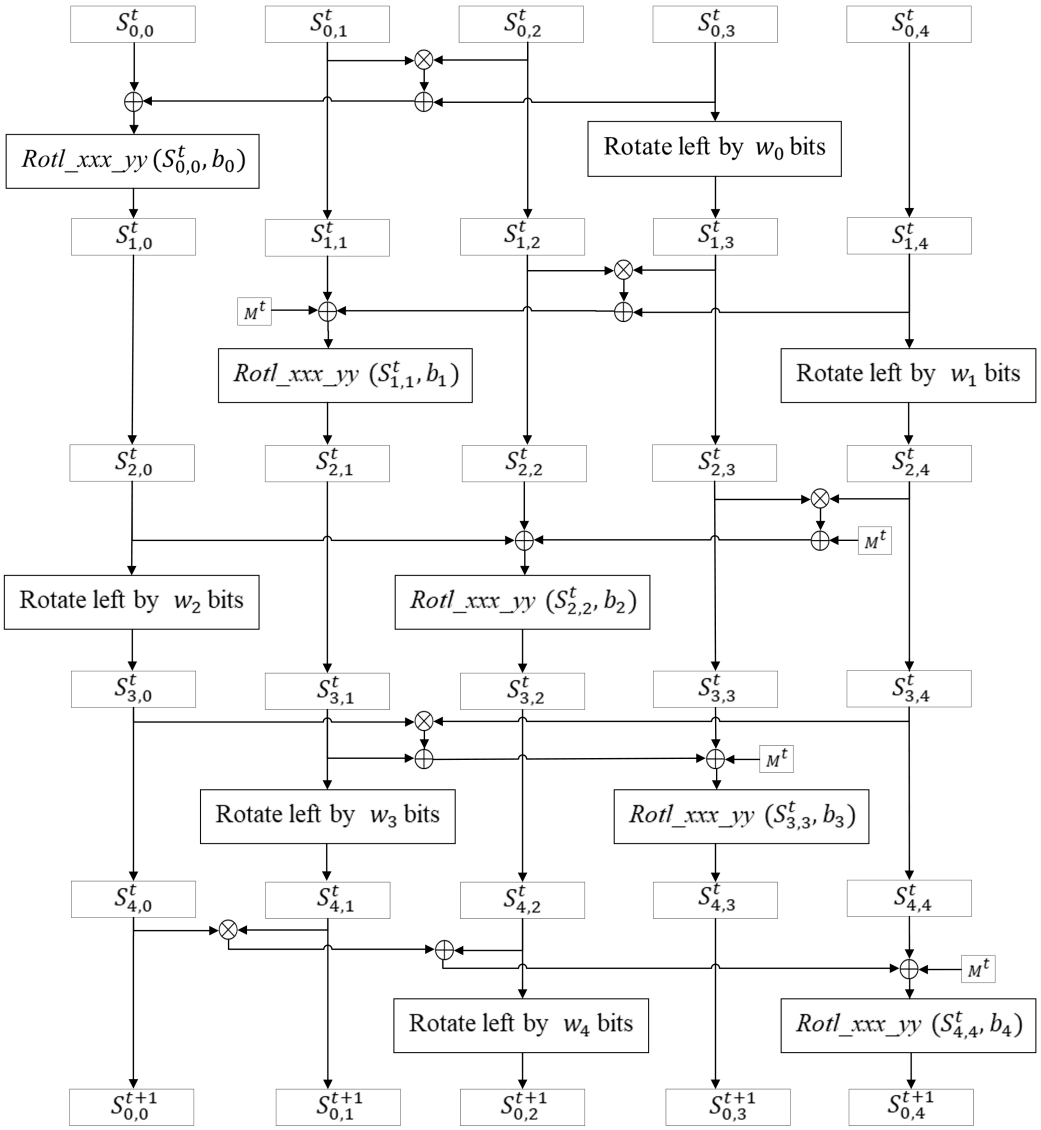

Figure 1. State update function of MORUS.

2.2.2. Keystream Generation Function

The keystream generation function outputs a keystream block at each step of the encryption phase. The first four state elements $(S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3})$ of MORUS are used in the keystream function, as shown in

$$Z^t = S^t_{0,0} \oplus \overleftarrow{S^t_{0,1}}^{w_2} \oplus (S^t_{0,2} \otimes S^t_{0,3}). \tag{1}$$

In the keystream generation function, w_2 is the left rotation constant, set to 96 bits or 192 bits for MORUS-640 and MORUS-1280, respectively.

2.2.3. Combining Function

MORUS is a binary additive stream cipher. Hence, it uses a simple XOR operation as the combining function during the encryption phase. That is, the ciphertext block is computed by XOR-ing the input plaintext block and the keystream block. At each clock t of the encryption phase, MORUS outputs a keystream block using Equation (1). This output keystream block Z^t is XOR-ed with the plaintext block P^t to compute the respective ciphertext block. The ciphertext computation is shown in

$$\begin{aligned} C^t &= P^t \oplus Z^t \\ &= P^t \oplus S^t_{0,0} \oplus \overleftarrow{S^t_{0,1}}^{w_2} \oplus (S^t_{0,2} \otimes S^t_{0,3}). \end{aligned} \tag{2}$$

3. Rotational Cryptanalysis of MORUS

Rotational cryptanalysis investigates the propagation of rotational pairs in the outputs of a cryptographic scheme for any given rotational input pairs. Let X and $\overleftarrow{X}^r$ denote two input vectors, where $\overleftarrow{X}^r$ is equivalent to the r-bit left rotated version of the input vector X. The input vector pair $(X, \overleftarrow{X}^r)$ is called a rotational input pair. In rotational cryptanalysis, the adversary inputs such a rotational pair $(X, \overleftarrow{X}^r)$ into the underlying cryptographic algorithm and observes the behavior of the output pairs. Let Z and Z' denote the resulting output pair for the inputs X and $\overleftarrow{X}^r$, respectively; and $\overleftarrow{Z}^r$ denotes the r-bit left rotated version of Z. Having $Z' = \overleftarrow{Z}^r$ implies that the operations involved in the cryptographic algorithm do not affect the rotational relations in the output pair (Z, Z'). That is, in such a scenario, the rotational relation is preserved in the output pair (Z, Z') for the corresponding input pair $(X, \overleftarrow{X}^r)$. Clearly, if the rotational relation is preserved, the adversary can observe it in the output pair to build a distinguisher for the underlying cryptographic algorithm.

In the attack model, the adversary needs to select a pair of inputs. In the initialization phase of stream cipher based scheme, the adversary can select the rotational input pairs from the key or the initialization vector or combination of both key-initialization vector. Therefore, the attack model follows the chosen key attack, or the chosen initialization vector attack, or the chosen key-initialization vector attack. Additionally, note that the inputs in the rotational pair are related to each other; thus, this can also be considered as a related key-initialization vector attack.

This section discusses the applicability of rotational cryptanalysis to MORUS. We investigated the basic rotational properties of different operations used in the MORUS state update function. XOR-ing of constants also plays an important role in the analysis of rotational pairs. Our investigation also included the analysis of rotation invariant bits in the constants used for MORUS. We then investigated the rotational properties of MORUS state contents using the above mentioned operations. The goal of these investigations was to construct a distinguisher for MORUS, if the rotational properties were preserved in the input and the output.

3.1. Rotational Properties of Operations Used in MORUS

The operations in the state update function of MORUS include AND, XOR and rotation operations. It also uses the $Rotl\_xxx\_yy(x, b)$ operation, which divides a xxx-bit block input x into 4 yy-bit words and rotates each word to the left by b bits.

The rotational properties of bitwise XOR and rotation operation were investigated by Khovratovich and Nikolić [19]. In the following, we review these properties. We also analyze the properties of the $Rotl\_xxx\_yy(x, b)$ operation, which were not explored in any previous literature.

3.1.1. Rotational Properties of XOR and Rotation Operation

Khovratovich and Nikolić stated that XOR operation preserves rotational pairs [19]. That is, for an r-bit rotation, Equation (3) is true. Their work stated that Equation (3) is

true, but did not include the proof of this statement. For completeness, we provide a proof for this.

Theorem 1. *Bitwise XOR operation applied to a binary string X preserves the rotational pairs for any arbitrary rotation distance r. That is,*

$$\overleftarrow{X}^r \oplus \overleftarrow{Y}^r = \overleftarrow{X \oplus Y}^r. \tag{3}$$

Proof of Theorem 1. Suppose $X = [X_L|_i X_R]$ denotes the notation for n-bit string $X = x_{n-1} \cdots x_0$, where $X_L = x_{n-1} \cdots x_i$ and $X_R = x_{i-1} \cdots x_0$. Similarly we define the string $Y = [Y_L|_i Y_R]$ for n-bit string $Y = y_{n-1} \cdots y_0$, where $Y_L = y_{n-1} \cdots y_i$ and $Y_R = y_{i-1} \cdots y_0$. Then, for a r-bit arbitrary left rotation with $0 \leq r \leq n$, we can represent X and Y as $X = [X_L|_{n-r} X_R]$ and $Y = [Y_L|_{n-r} Y_R]$, respectively. We can write

$$\begin{aligned}
\overleftarrow{X}^r \oplus \overleftarrow{Y}^r &= \overleftarrow{[X_L|_{n-r} X_R]}^r \oplus \overleftarrow{[Y_L|_{n-r} Y_R]}^r \\
&= [X_R|_r X_L] \oplus [Y_R|_r Y_L] \\
&= [(X_R \oplus Y_R)|_r X_L \oplus Y_L]
\end{aligned}$$

and

$$\begin{aligned}
\overleftarrow{X \oplus Y}^r &= \overleftarrow{[X_L|_{n-r} X_R] \oplus [Y_L|_{n-r} Y_R]}^r \\
&= \overleftarrow{[(X_L \oplus Y_L)|_{n-r} (X_R \oplus Y_R)]}^r \\
&= [(X_R \oplus Y_R)|_r X_L \oplus Y_L].
\end{aligned}$$

Therefore, $\overleftarrow{X}^r \oplus \overleftarrow{Y}^r = \overleftarrow{X \oplus Y}^r$. □

The analysis show that the bitwise rotation operation also preserves the rotational pairs. Therefore, XOR and rotation operations in the state update function of MORUS do not break the symmetry in the rotational pairs.

3.1.2. Rotational Properties of AND Operation

Similar to the XOR operation, it is easy to prove that the bitwise AND operation always preserves the rotational pairs. That is, Equation (4) is true for any arbitrary rotation distance r.

Theorem 2. *Bitwise AND operation applied to a binary string X preserves the rotational pairs for any arbitrary rotation distance r. That is,*

$$\overleftarrow{X}^r \otimes \overleftarrow{Y}^r = \overleftarrow{X \otimes Y}^r. \tag{4}$$

Proof of Theorem 2. Suppose $X = [X_L|_i X_R]$ denotes the notation for n-bit string $X = x_{n-1} \cdots x_0$, where $X_L = x_{n-1} \cdots x_i$ and $X_R = x_{i-1} \cdots x_0$. Similarly we define the string $Y = [Y_L|_i Y_R]$ for n-bit string $Y = y_{n-1} \cdots y_0$, where $Y_L = y_{n-1} \cdots y_i$ and $Y_R = y_{i-1} \cdots y_0$. Then, for a r-bit arbitrary left rotation with $0 \leq r \leq n$, we can represent X and Y as $X = [X_L|_{n-r} X_R]$ and $Y = [Y_L|_{n-r} Y_R]$, respectively. We can write

$$\begin{aligned}
\overleftarrow{X}^r \otimes \overleftarrow{Y}^r &= \overleftarrow{[X_L|_{n-r} X_R]}^r \otimes \overleftarrow{[Y_L|_{n-r} Y_R]}^r \\
&= [X_R|_r X_L] \otimes [Y_R|_r Y_L] \\
&= [X_R \otimes Y_R|_r X_L \otimes Y_L]
\end{aligned}$$

and

$$\begin{aligned}\overleftarrow{X \otimes Y}^r &= \overleftarrow{[X_L|_{n-r} X_R] \otimes [Y_L|_{n-r} Y_R]}^r \\ &= \overleftarrow{[(X_L \otimes Y_L)|_{n-r} (X_R \otimes Y_R)]}^r \\ &= [(X_R \otimes Y_R)|_r X_L \otimes Y_L].\end{aligned}$$

Therefore, $\overleftarrow{X}^r \otimes \overleftarrow{Y}^r = \overleftarrow{X \otimes Y}^r$. □

Therefore, bitwise AND operation in the state update function of MORUS does not break the symmetry in the rotational pairs.

3.1.3. Rotational Properties of $Rotl\_xxx\_yy(x, b)$ Operation

MORUS uses the $Rotl\_xxx\_yy(x, b)$ operation which is a composition of two operations. This includes dividing the input word into four sub-words of equal length and then applying bitwise left rotation operation to these sub-words of MORUS. This operation can be considered as a bit-wise permutation. In general, bit-wise permutation does not preserve the rotational pairs. This is illustrated below with a simple example.

Let $X = x_0x_1x_2x_3x_4x_5x_6x_7$ be a sequence of 8 bits. The application of $Rotl\_8\_2(x, 1)$ operation on this 8-bit sequence X results in:

$$X_{rotl_1} = Rotl\_8\_2(X, 1) = x_1x_0x_3x_2x_5x_4x_7x_6.$$

Following this, the application of a one-bit left rotation applied to X_{rotl_1} results in:

$$\overleftarrow{X_{rotl_1}}^1 = x_0x_3x_2x_5x_4x_7x_6x_1.$$

Now, consider the alternative where first, the one bit left rotation is applied to the 8 bit sequence X, followed by the application of $Rotl\_8\_2(x, 1)$ operation on the rotated sequence. The one bit left rotation applied to the sequence X results in:

$$\overleftarrow{X}^1 = x_1x_2x_3x_4x_5x_6x_7x_0.$$

Following that, application of $Rotl\_8\_2(x, 1)$ operation on the rotated sequence $\overleftarrow{X}^1$ results in:

$$\overleftarrow{X}^1_{rotl_1} = x_2x_1x_4x_3x_6x_5x_0x_7.$$

Clearly $\overleftarrow{X_{rotl}}^1 \neq \overleftarrow{X}^1_{rotl}$ in general, and therefore a rotational pair is not guaranteed to be preserved.

Conditions for Preserving a Rotational Pair under $Rotl\_xxx\_yy(x, b)$

We observe that the operation $Rotl\_xxx\_yy(x, b)$ preserves the rotational pair, if the distance r of the left rotation applied to X equals to a multiple of the sub-word length yy. For the above example, the sub-word length is two. Applying two bit left rotation to the sequence X results in:

$$\overleftarrow{X}^2 = x_2x_3x_4x_5x_6x_7x_0x_1.$$

Application of $Rotl\_8\_2(x, 1)$ operation on the rotated sequence $\overleftarrow{X}^2$ results in:

$$\overleftarrow{X}^2_{rotl_1} = x_3x_2x_5x_4x_7x_6x_1x_0.$$

Alternatively, application of two bit left rotation applied to X_{rotl_1} results in:

$$\overleftarrow{X_{rotl_1}}^2 = x_3x_2x_5x_4x_7x_6x_1x_0.$$

As shown in the example, $\overleftarrow{X}^{2}_{rotl} = \overleftarrow{\overleftarrow{X}}^{2}_{rotl}$ when the distance of the rotation applied is equal to the sub-word size. Experimental verification also confirms that the rotational pairs are preserved when the distance of the left rotation applied to X is equal to a multiple of the sub-word length yy. Additionally, the rotational pair $(X, \overleftarrow{X}^{r})$ will be preserved if the sequence X is rotation invariant for any arbitrary rotation distance.

We define the following theorem which identifies the condition for preserving rotation pairs when the $Rotl\_xxx\_yy(x, b)$ operation is applied.

Theorem 3. *The $Rotl\_xxx\_yy(x, b)$ operation applied to a binary string X preserves the rotational pairs if the distance of the rotation r applied is equal to a multiple of the sub-word size or if the input X is rotation invariant. That is,*

$$\overleftarrow{X}^{r}_{rotl_b} = \overleftarrow{\overleftarrow{X}}^{r}_{rotl_b}, \tag{5}$$

if r is a multiple of the sub-word size.

Proof of Theorem 3. Suppose $X = X_1|X_2|X_3|X_4$ denotes a n-bit string $X = x_{n-1} \cdots x_0$, where $X_1 = x_{n-1} \cdots x_{n-\frac{n}{4}}$, $X_2 = x_{n-\frac{n}{4}-1} \cdots x_{n-\frac{2n}{4}}$, $X_3 = x_{n-\frac{2n}{4}-1} \cdots x_{n-\frac{3n}{4}}$ and $X_4 = x_{n-\frac{3n}{4}-1} \cdots x_0$. Let $X = [X_L|_i X_R]$ denote the notation for n-bit string $X = x_{n-1} \cdots x_0$, where $X_L = x_{n-1} \cdots x_i$ and $X_R = x_{i-1} \cdots x_0$. Let b be the rotation distance applied in the $Rotl\_xxx\_yy(x, b)$ operation. Then, $X_{rotl_b} = (X_1|X_2|X_3|X_4)_{rotl_b}$ can be represented as $X_{rotl_b} = (\overleftarrow{X_1}^{b}|\overleftarrow{X_2}^{b}|\overleftarrow{X_3}^{b}|\overleftarrow{X_4}^{b})$.

The length of the sub-word is $n/4$ for a n bit string. Thus, we need to show that the rotation distance $r = n/4, 2n/4, 3n/4, n$ on the $Rotl\_xxx\_yy(x, b)$ operation will preserve the rotational pairs.

For a r-bit left rotation with $r = n/4$, we can represent the left hand side of Equation (5) as:

$$\begin{aligned}
\overleftarrow{X}^{r=n/4}_{rotl} &= \overleftarrow{(X_1|X_2|X_3|X_4)_{rotl_b}}^{r=n/4} \\
&= (X_2|X_3|X_4|X_1)_{rotl_b} \\
&= (\overleftarrow{X_2}^{b}|\overleftarrow{X_3}^{b}|\overleftarrow{X_4}^{b}|\overleftarrow{X_1}^{b}) \\
&= (\overleftarrow{[X_{2L}|_{n/4-b}X_{2R}]}^{b}|\overleftarrow{[X_{3L}|_{n/4-b}X_{3R}]}^{b}|\overleftarrow{[X_{4L}|_{n/4-b}X_{4R}]}^{b}|\overleftarrow{[X_{1L}|_{n/4-b}X_{1R}]}^{b}) \\
&= ([X_{2R}|_b X_{2L}]|[X_{3R}|_b X_{3L}]|[X_{4R}|_b X_{4L}]|[X_{1R}|_b X_{1L}]).
\end{aligned}$$

For a r-bit left rotation with $r = n/4$, we can represent the right hand side of Equation (5) as:

$$\begin{aligned}
\overleftarrow{\overleftarrow{X}}^{r=n/4}_{rotl_b} &= \overleftarrow{(X_1|X_2|X_3|X_4)_{rotl_b}}^{r=n/4} \\
&= (\overleftarrow{X_1}^{b}|\overleftarrow{X_2}^{b}|\overleftarrow{X_3}^{b}|\overleftarrow{X_4}^{b}) \\
&= \overleftarrow{(\overleftarrow{[X_{1L}|_{n/4-b}X_{1R}]}^{b}|\overleftarrow{[X_{2L}|_{n/4-b}X_{2R}]}^{b}|\overleftarrow{[X_{3L}|_{n/4-b}X_{3R}]}^{b}|\overleftarrow{[X_{4L}|_{n/4-b}X_{4R}]}^{b})}^{r=n/4} \\
&= \overleftarrow{([X_{1R}|_b X_{1L}]|[X_{2R}|_b X_{2L}]|[X_{3R}|_b X_{3L}]|[X_{4R}|_b X_{4L}])}^{r=n/4} \\
&= ([X_{2R}|_b X_{2L}]|[X_{3R}|_b X_{3L}]|[X_{4R}|_b X_{4L}]|[X_{1R}|_b X_{1L}]).
\end{aligned}$$

Therefore, $\overleftarrow{X}^{r}_{rotl_b} = \overleftarrow{\overleftarrow{X}}^{r}_{rotl_b}$ when $r = n/4$. Similarly, we can prove that Theorem 3 is true for a rotation distance $r = 2n/4, 3n/4, n$. Additionally, if the input sequence X is rotation invariant for arbitrary rotation distance r, then application of $Rotl\_xxx\_yy(x, b)$ to the input X will not have any effect on it. Therefore, the operation $Rotl\_xxx\_yy(x, b)$ will preserve the rotational pairs if the input X is rotation invariant. □

In MORUS-640, the sub-word size is 32 bits. Thus, to maintain the rotational pairs in the $Rotl\_xxx\_yy(x, b)$ operation of MORUS-640, the distance of the rotations applied should be a multiple of 32. Analogously, the sub-word size is 64 bits for MORUS-1280. Therefore, the distance of the rotations applied needs to be a multiple of 64 to preserve the rotational pair in the $Rotl\_xxx\_yy(x, b)$ operation of MORUS-1280.

3.2. Rotational Properties of the Constants in MORUS

MORUS uses two constants $const_0$ and $const_1$ at the beginning of the initialization phase, which are used to load some of the state elements. These constants are XOR-ed with the contents of specific state elements as a part of the state update process. Khovratovich and Nikolić show that addition/XOR of a constant breaks the symmetry in rotational relations, if the constant is not rotation invariant [19]. Therefore, XOR-ing of constants plays an important role in the rotational cryptanalysis. In the following, we discuss the rotational properties of the constants used in MORUS.

3.2.1. Rotational Properties of MORUS-640 Constants

We analyzed the MORUS-640 constants when r-bit rotation is applied to it. We investigated the number of bits in the constant which are rotation invariant for the r-bit rotation. MORUS-640 applies the $Rotl\_128\_32(x, b)$ operation in its state update function. According to Theorem 3, the $Rotl\_128\_32(x, b)$ preserves the rotational pairs if the rotation distance is a multiple of 32 bits, i.e., 32, 64 or 96. Therefore, for the analysis of MORUS-640 constants, we set the rotation distance r as a multiple of 32.

With the above conditions, the constant also needs to be 32, 64, 96 or 128 bits rotation invariant depending on the distance of the rotations applied. If the constant is 32-bit rotation invariant (i.e., the constant can be divided into 4 small sub-word of 32 bits where each of the sub-word has same content), then there are 2^{32} rotation invariant constants. For a 64 bits rotation invariant constant, there are 2^{64} rotation invariant constants.

We investigated the rotation invariant bits in the constants for any distance of the rotation which are multiples of $r = 32$. For any multiples of $r = 32$, the constants used in MORUS-640 are not rotation invariant among all the bits; except for the trivial one $r = 128, 256, \cdots$ which is basically the same constant. In the following, we discuss the number of bits that remain rotation invariant for different rotation distances set based on above criteria.

In MORUS-640, $const_0$ and $const_1$ are used to load the state element $S_{0,3}$ and $S_{0,4}$. We examined the number of rotation invariant bits in these constants for any rotation distance which is a multiples of 32. The rotated constant $\overleftarrow{const_0}^{r}$ has 74, 60 and 74 rotation invariant bits for 32, 64, and 96 bit rotation distances, respectively. Similarly, the rotated constant $\overleftarrow{const_1}^{r}$ has 64, 62 and 64 rotation invariant bits for 32, 64, and 96 bit rotation distances, respectively. This means XOR-ing of these constants with the contents of any state element inverts about half of the bits in the corresponding state element compared to the result of XOR-ing with the rotated version of the constant.

Additionally, in the first step of the initialization phase, $\overleftarrow{const_0}^{32} \oplus const_1$ is used to update $S_{1,1,}$. The rotated version of this constant $\overleftarrow{\overleftarrow{const_0}^{32} \oplus const_1}^{r}$ has 62, 74 and 62 rotation invariant bits for 32, 64, and 96 bit rotation distances, respectively.

The constant $(\overleftarrow{const_0}^{32} \otimes \overleftarrow{const_1}^{64}) \oplus Rotl\_128\_32(const_0, 5)$ is used at the first step of initialization phase to update the state element $S_{2,2}$. The rotated version of this constant $\overleftarrow{(\overleftarrow{const_0}^{32} \otimes \overleftarrow{const_1}^{64}) \oplus Rotl\_128\_32(const_0, 5)}^{r}$ has 72, 64 and 72 rotation invariant bits for rotation distances 32, 64, and 96 bits, respectively.

3.2.2. Rotational Properties of MORUS-1280 Constants

We investigated the constants used in MORUS-1280, when r-bit rotation is applied to it. The investigation explores the number of rotation invariant bits in the constant for

rotation distance r. MORUS-1280 applies the $Rotl\_256\_64(x, b)$ operation in its state update function. According to Theorem 3, the operation $Rotl\_256\_64(x, b)$ preserves the rotational pairs if the rotation distance is a multiple of 64 bits, i.e., 64, 96, 192 or 256. Therefore, for the analysis of MORUS-1280 constants, we set the rotation distance r as a multiple of 64.

With the above conditions, the constant also needs to be 64, 128, 192 or 256 bits rotation invariant depending on the distance of rotations applied. For MORUS-1280, if the constant is 64-bit rotation invariant (i.e., the constant can be divided into four small sub-word of 64 bits where each of the sub-word has same content), then there are 2^{64} rotation invariant constants. For a 128 bits rotation invariant constant, there are 2^{128} rotation invariant constants which will preserve the rotational pairs for MORUS-1280.

We investigated the rotation invariant bits in the constants for any distances of rotation which are multiples of $r = 64$. For any multiples of $r = 64$, the constant $const_0||const_1$ used in MORUS-1280 is not rotation invariant among all the bits; except for the trivial one $r = 256, 512, \cdots$ which is basically the same constant. In the following, we discuss the number of bits that remain rotation invariant for different rotation distances set based on above criteria.

In MORUS-1280, $const_0||const_1$ is used to load the state element $S_{0,4}$. The rotated version of this constant $\overleftarrow{const_0||const_1}^{r}$ has 138, 114 and 138 rotation invariant bits for rotation distances 64, 128 and 192 bits, respectively. This means that XOR-ing this constant $const_0||const_1$ with the contents of the state element $S_{0,4}$ will preserve the rotational pairs in 138 bits when the rotation distance in the input is set to 64 bits.

The constant used to initialize state element $S_{0,2}$ consists of all 1. This constant is rotation invariant for any arbitrary rotation distance. Similarly, the constant used to initialize state element $S_{0,3}$ consists of all 0s. This constant is also rotation invariant for any arbitrary rotation distance.

3.3. Rotational Properties of MORUS State Contents

This section investigates the rotational properties in the MORUS state contents. The contents of MORUS state elements were updated using the state update function $Update(S^t, M^t)$. Rotational properties of the operations and constants used in the state update function $Update(S^t, M^t)$ are discussed in Sections 3.1 and 3.2. In our investigation, we used these properties to determine the propagation of rotational pairs in MORUS state contents.

We first investigated the probability of preserving the rotational pairs in the MORUS state elements after one step of the initialization phase. We then extended our analysis for more than one step of the initialization phase.

We started our experiment by defining the key K and initialization vector V in terms of variables. We then generated equations in terms of K and V to represent the state contents of n-step MORUS. After that, we generated the equations in terms of rotated versions of the key $\overleftarrow{K}^{r}$ and initialization vector $\overleftarrow{V}^{r}$ to represent the state contents of n-step MORUS. Finally, we investigated the rotational pair in the state elements $(S, \overleftarrow{S}^{r})$ to determine whether they preserve the rotational pair. Preserving the rotational pair in the state elements means that the rotational pair in the output keystream is also preserved.

If the rotational pair is preserved in the output pair, then adversary can observe the output pair and use this to distinguish the MORUS output from a randomly generated output. This can be considered as a distinguisher under the related key-IV model.

3.3.1. Rotational Properties of MORUS-640 State Contents

MORUS-640 uses the operations XOR, AND, bitwise left rotation and the operation $Rotl\_128\_32(x, b)$ in its state update function. As illustrated in Theorems 1 and 2, bitwise XOR and AND operation will preserve the rotational pair for any arbitrary number of rotations. However, for the $Rotl\_128\_32(x, b)$ operation, as described in Theorem 3, the rotation distance in the input needs to be a multiple of 32 bits—i.e., 32, 64 or 92—for MORUS-640, to preserve rotational pairs in the output. Thus, in our analysis of MORUS-640

state contents, the distance of the rotation r was set to 32. We conducted our analysis with the rotation distance of 32 bits, since that preserved the rotational pairs in maximum number of bits after performing the XOR operation with the constant $const_0$ (see Section 3.2.1).

Rotational Pairs in MORUS-640 with 1-Step Initialization Phase

For a 32 bit rotated input, rotational pairs are preserved in 74 bits of the state element $S_{0,0}$ with probability 1. Rotational pairs are preserved in the rest of the 54 bits of the state element $S_{0,0}$ with probability 0—i.e., the corresponding bits are inverted. This is because of the the XOR of $const_0$ with the contents of state element $S_{0,0}$, at the beginning of the initialization phase. As shown in Section 3.2.1 for $r = 32$ bit rotation distance, $const_0$ has 74 bits which are rotation invariant. Thus, this introduces 54 inverted bits.

For state element $S_{0,1}$, rotational pairs are preserved in 62 bits with probability 1. Rotational pairs are preserved in the rest of the 66 bits of the state element $S_{0,1}$ with probability 0; i.e., the corresponding bits are inverted. Note that $\overleftarrow{\overleftarrow{const_0}^{32}}^{32} \oplus const_1$ is XOR-ed with the contents of state element $S_{1,1}$. As shown in Section 3.2.1, constant $\overleftarrow{\overleftarrow{const_0}^{32}}^{32} \oplus const_1$ has 62 rotation invariant bits. Thus, this introduces $128 - 62 = 66$ inversion in the resulting rotational pair.

Rotational pairs are preserved in 72 bits of the state element $S_{0,2}$ with probability 1. Rotational pairs are preserved in the remaining 56 bits of the state element $S_{0,2}$ with probability 0; i.e., the corresponding bits are inverted. We also observe that the constant $\overleftarrow{(\overleftarrow{const_0}^{32} \otimes \overleftarrow{const_1}^{64}) \oplus Rotl\_128\_32(const_0, 5)}^{32}$ is XOR-ed with the contents of $S_{0,2}$. As illustrated in Section 3.2.1, constant $\overleftarrow{(\overleftarrow{const_0}^{32} \otimes \overleftarrow{const_1}^{64}) \oplus Rotl\_128\_32(const_0, 5)}^{32}$ has 72 rotation invariant bits. Thus, this introduces $128 - 72 = 56$ inversion in the resulting rotational pair.

Rotational pairs are preserved with probabilities 1, 0.5 and 0 in 36, 64 and 28 bits of $S_{0,3}$, respectively. Finally, rotational pairs are preserved with probabilities 1, 0.5 and 0 in 14, 94 and 20 bits of $S_{0,4}$, respectively. Table 2 summarizes the probability of preserving rotational pairs in the state elements of 1-step MORUS-640.

Table 2. Probability of preserving rotational pairs after one step of the initialization of MORUS-640.

State Element	$p = 1$	$p = 0.5$	$p = 0$
$S_{0,0}$	74	0	54
$S_{0,1}$	62	0	66
$S_{0,2}$	72	0	56
$S_{0,3}$	36	64	28
$S_{0,4}$	14	94	20
	258	158	224

As shown in Table 2, after one step there are 482 known differences in rotational pairs based on 32-bit rotations of the state elements of MORUS. Therefore, observing these known differences in the specific output bit of a rotational pair, an adversary can distinguish the keystream of 1-step MORUS-640 from a randomly generated output.

Rotational Pairs in MORUS-640 with Initialization Phase beyond One Step

We extended our experiments to determine the probability of preserving rotational pairs in more than one step of the initialization phase of MORUS-640. For two steps of the initialization phase, we found there are only 21 known differences in rotational pairs of the state contents of MORUS-640. In particular, we found only 14 bits and 7 bits are preserved in state elements S_0 and S_1, respectively, with a probability of 1 or 0. This is not

sufficient to determine a distinguisher because there are no known differences in the rest of the state elements.

For more than two steps of the initialization phase, Sage fails to generate the equations. This is due to the fact that the equations get very complicated after two steps of the initialization phase and the software Sage runs out of memory to perform the necessary computations.

As indicated above, after the two steps of the initialization phase, there are not enough rotational pairs in the state elements to construct a distinguisher and such situation becomes worse with more steps. Therefore, it is unlikely for an adversary to construct a distinguisher for the full version of MORUS-640 based on observing the rotational pairs.

3.3.2. Rotational Properties of MORUS-1280 State Contents

MORUS-1280 uses the XOR, AND, bitwise left rotation and the $Rotl\_256\_64(x, b)$ operations in its state update function. As illustrated in Theorems 1 and 2, bitwise XOR and AND operation will preserve the rotational pair for any arbitrary number of rotations. However, for the $Rotl\_256\_64(x, b)$ operation as described in Theorem 3, the rotation distance in the input needs to be a multiple of 64 bits, i.e., 64, 128 or 192, for MORUS-1280, to preserve rotational pairs in the output. Thus, in our analysis of MORUS-1280 state contents, the distance of the rotation r is set to 64. We conducted our analysis with the rotation distance of 64 bits, since it preserved the rotational pairs in the maximum number of bits after performing the XOR operation with the constant $const_0$ (see Section 3.2.2).

Rotational Pairs in MORUS-1280 with 1-Step Initialization Phase

Rotational pairs are preserved in all of the 256 bits of the state element $S_{0,0}$ with probability 1. This is because the constant XOR-ed with the contents of this state element are all zeroes and so rotation invariant for arbitrary rotation distance r.

Rotational pairs are preserved in 138 bits of the state element $S_{0,1}$ with probability 1. Rotational pairs are preserved in the remaining 118 bits of the state element $S_{0,1}$ with probability 0; i.e., the corresponding bits are inverted. Observe that $const_0||const_1$ is XOR-ed with the contents of the state element $S_{0,1}$. As described in Section 3.2.2, $\overleftarrow{const_0||const_1}^{64}$ has 138 rotation invariant bits. Thus, XOR-ing of this constant introduces 118 bits of inversion in the rotational pair.

Rotational pairs are preserved in all of the 256 bits of the state element $S_{0,2}$ with probability 1. This is because the constant XOR-ed with the contents of this state element is $const_0 \otimes 0^{128}$, e.g., all zeroes, and rotation invariant for arbitrary rotation distance r.

Rotational pairs are preserved with probabilities 1, 0.5 and 0 in 73, 118 and 65 bits of $S_{0,3}$, respectively. Rotational pairs are preserved with probabilities 1, 0.5 and 0 in 78, 118 and 60 bits of $S_{0,4}$, respectively.

Table 3 summarizes the probability of preserving rotational pairs in the state elements of 1-step MORUS-1280. We found that rotational pairs are preserved with probability 1 in 801 bits of the 1-step MORUS-1280. After one step of the initialization phase, there are only 236 bits which are unknown with probability 0.5. Therefore, based on these known differences in the specific output bit of a rotational pair, an adversary can distinguish the keystream of 1-step MORUS-1280 from a randomly generated output.

Table 3. Probability of preserving rotational pairs after one step of the initialization of MORUS-1280.

State Element	$p = 1$	$p = 0.5$	$p = 0$
$S_{0,0}$	256	0	0
$S_{0,1}$	138	0	118
$S_{0,2}$	256	0	0
$S_{0,3}$	73	118	65
$S_{0,4}$	78	118	60

Rotational Pairs in MORUS-1280 with Initialization Phase beyond One Step

We extended our experiments to determine the probability of preserving rotational pairs in more than one step of the initialization phase of MORUS-640. For two steps of the initialization phase, we found only 30 bits and 17 bits are preserved in state elements S_0 and S_1, respectively, with a probability of 1 or 0. This is not sufficient to determine a distinguisher because there are no known differences in the rest of the state elements.

For more than two steps of the initialization phase, Sage fails to generate the equations for MORUS-1280. This is due to the fact that the equations get very complicated after two steps of the initialization phase and the software Sage runs out of memory.

As indicated above, after the two steps of the initialization phase, there are not enough rotational pairs in the state elements to construct a distinguisher; and said situation becomes worse with more steps. Therefore, it is unlikely an adversary will be able to construct a distinguisher for the full version of MORUS-1280 based on observing the rotational pairs.

4. Conclusions

We investigated the feasibility of rotational cryptanalysis on different variants of MORUS. Our investigation showed that all the operations used in MORUS preserve the rotational pairs when the rotation distance is set to a multiple of 32 or 64 for MORUS-640 and MORUS-1280, respectively. We have also verified that an adversary can build a distinguisher for the full version of MORUS if rotational-invariant constants are used in the state update function of MORUS. However, the constants used in MORUS are not rotational-invariant, which makes it infeasible to build the distinguisher for more than one step. Due to the non-invariant constants used in the state update function of MORUS, we found that rotational cryptanalysis can distinguish the MORUS output for only one step of the initialization phase. For more than one step of the initialization phase, the probability of preserving rotational pair becomes 0.5 in most of the bits. This makes it infeasible to apply the distinguisher for more than one round.

Funding: This research was funded by Xiamen University Malaysia Research Fund (grant number XMUMRF/2019-C3/IECE/0005).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Wu, H.; Huang, T. The Authenticated Cipher MORUS (v1). CAESAR Competition. Available online: https://competitions.cr.yp.to/round1/morusv1.pdf (accessed on 14 October 2021).
2. Wu, H.; Huang, T. The Authenticated Cipher MORUS (v2). CAESAR Competition. Available online: https://competitions.cr.yp.to/round3/morusv2.pdf (accessed on 14 October 2021).
3. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available online: https://competitions.cr.yp.to/index.html (accessed on 14 October 2021).
4. Mileva, A.; Dimitrova, V.; Velichkov, V. Analysis of the authenticated cipher MORUS (v1). In *Cryptography and Information Security in the Balkans—BalkanCryptSec 2015*; Lecture Notes in Computer Science; Pasalic, E., Knudsen, L.R., Eds.; Springer: Cham, Switzerland, 2015; Volume 9540, pp. 45–59. [CrossRef]
5. Dwivedi, A.D.; Klouček, M.; Morawiecki, P.; Nikolić, I.; Pieprzyk, J.; Wójtowicz, S. SAT-based cryptanalysis of authenticated ciphers from the CAESAR Competition. In Proceedings of the 14th International Conference on Security and Cryptography—SECRYPT 2017, Madrid, Spain, 24–26 July 2017; pp. 237–246. [CrossRef]
6. Dwivedi, A.; Morawiecki, P.; Wójtowicz, S. Differential and rotational cryptanalysis of round-reduced MORUS. In Proceedings of the 14th International Conference on Security and Cryptography—SECRYPT 2017, Madrid, Spain, 24–26 July 2017; pp. 275–284. [CrossRef]
7. Salam, I.; Simpson, L.; Bartlett, H.; Dawson, E.; Pieprzyk, J.; Wong, K.K. Investigating cube attacks on the authenticated encryption stream cipher MORUS. In Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications—TrustCom 2017, Sydney, Australia, 1–4 August 2017; pp. 961–966. [CrossRef]

8. Kales, D.; Eichlseder, M.; Mendel, F. Note on the robustness of CAESAR candidates. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 1137.
9. Vaudenay, S.; Vizar, D. Under pressure: Security of CAESAR candidates beyond their guarantees. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 1147.
10. Salam, I.; Simpson, L.; Bartlett, H.; Dawson, E.; Wong, K.K. Fault attacks on the authenticated encryption stream cipher MORUS. *Cryptography* **2018**, *2*, 4. [CrossRef]
11. Nozaki, Y.; Yoshikawa, M. Power analysis attack for a fast authenticated encryption MORUS. In Proceedings of the International Conference on Applied System Innovation—ICASI 2017, Sapporo, Japan, 13–17 May 2017; pp. 365–368. [CrossRef]
12. Ashur, T.; Eichlseder, M.; Lauridsen, M.; Leurent, G.; Minaud, B.; Rotella, Y.; Sasaki, Y.; Viguier, B. Cryptanalysis of MORUS. In *Advances in Cryptology—ASIACRYPT 2018*; Lecture Notes in Computer Science; Peyrin, T., Galbraith, S., Eds.; Springer: Cham, Switzerland, 2018; Volume 11273, pp. 35–64. [CrossRef]
13. Li, Y.; Wang, M. Cryptanalysis of MORUS. *Des. Codes Cryptogr.* **2019**, *87*, 1035–1058. [CrossRef]
14. Shi, T.; Guan, J. Real-time state recovery attack against MORUS in nonce-misuse setting. *Sci. China Inf. Sci.* **2019**, *62*, 39109. [CrossRef]
15. Ye, T.; Wei, Y.; Meier, W. A new cube attack on MORUS by using division property. *IEEE Trans. Comput.* **2019**, *68*, 1731–1740. [CrossRef]
16. Shi, D.; Sun, S.; Sasaki, Y.; Li, C.; Hu, L. Correlation of quadratic Boolean functions: Cryptanalysis of all versions of full MORUS. In *Advances in Cryptology—CRYPTO 2019*; Lecture Notes in Computer Science; Boldyreva, A., Micciancio, D., Eds.; Springer: Cham, Switzerland, 2019; Volume 11693, pp. 180–209. [CrossRef]
17. Chen, S.; Xiang, Z.; Zeng, X.; Zhang, S. Cube attacks on round-reduced MORUS and GIMLI. *Sci. China Inf. Sci.* **2021**, *65*, 119101. [CrossRef]
18. He, Y.; Wang, G.; Li, W.; Ren, Y. Improved cube attacks on some authenticated encryption ciphers and stream ciphers in the internet of things. *IEEE Access* **2020**, *8*, 20920–20930. [CrossRef]
19. Khovratovich, D.; Nikolić, I. Rotational cryptanalysis of ARX. In *Fast Software Encryption—FSE 2010*; Lecture Notes in Computer Science; Hong, S., Iwata, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6147, pp. 333–346. [CrossRef]
20. Biham, E. New types of cryptanalytic attacks using related keys. *J. Cryptol.* **1994**, *7*, 229–246. [CrossRef]