Biclique Cryptanalysis of Full Round AES-128 Based Hashing Modes

Donghoon Chang, Mohona Ghosh(✉), and Somitra Kumar Sanadhya

Indraprastha Institute of Information Technology, Delhi (IIIT-D), New Delhi, India
{donghoon,mohonag,somitra}@iiitd.ac.in

Abstract. In this work, we revisit the security analysis of hashing modes instantiated with AES-128. We use biclique cryptanalysis as the basis for our evaluation. In Asiacrypt'11, Bogdanov et al. had proposed biclique technique for key recovery attacks on full AES-128. Further, they had shown application of this technique to find preimage for compression function instantiated with AES-128 with a complexity of $2^{125.56}$. However, this preimage attack on compression function cannot be directly converted to preimage attack on hash function. This is due to the fact that the initialization vector (IV) is a publically known constant in the hash function settings and the attacker is not allowed to change it, whereas the compression function attack using bicliques introduced differences in the chaining variable. We extend the application of biclique technique to the domain of hash functions and demonstrate second preimage attack on all 12 PGV modes.

The complexities of finding second preimages in our analysis differ based on the PGV construction chosen - the lowest being $2^{126.3}$ and the highest requiring $2^{126.6}$ compression function calls. We implement C programs to find the best biclique trails (that guarantee the lowest time complexity possible) and calculate the above mentioned values accordingly. Our security analysis requires only 2 message blocks and works on full 10 rounds of AES-128 for all 12 PGV modes. This improves upon the previous best result on AES-128 based hash functions by Sasaki at FSE'11 where the maximum number of rounds attacked is 7. Though our results do not significantly decrease the attack complexity factor as compared to brute force but they highlight the actual security margin provided by these constructions against second preimage attack.

Keywords: AES · Block ciphers · Hash functions · Cryptanalysis · Biclique · Second preimage attack

1 Introduction

Block ciphers have been favored as cryptographic primitives for constructing hash functions for a long time. In [17], Preneel et al. proposed 64 basic ways to construct a n-bit compression function from a n-bit block cipher (under a n-bit key). Black et al. [5] analyzed the security of such constructions and showed

D. Lin et al. (Eds.): Inscrypt 2015, LNCS 9589, pp. 3–21, 2016.
DOI: 10.1007/978-3-319-38898-4_1

12 of them to be provably secure. These modes are commonly termed as PGV hash modes. The three most popularly used modes are Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP) modes.

AES (Advanced Encryption Standard), standardized by the US NIST in October 2000 and widely accepted thereafter has been considered a suitable candidate for block cipher based hash functions in the cryptographic community. ISO standardized Whirlpool [3] is a popular example of the same. Infact, in the recently concluded SHA-3 competition also, several AES based hash functions were submitted, e.g., LANE [11], ECHO [4], Grøstl [9] etc. A significant progress has been made in the field of block cipher based hash function security. Spearheaded by rebound attacks alongwith other cryptanalytic techniques, several AES as well as other block cipher based dedicated hash functions have been reviewed and cryptanalyzed [12,14–16,18,19,21]. But all of the analysis that has been done has been performed on round-reduced versions of block ciphers. Specifically, if we refer to the previous best result on AES-128 based hash modes performed by Sasaki [18], the maximum number of rounds attacked is 7.

The reason behind this restriction was the fact that AES-128 itself was resistant to full 10 rounds attack for a considerable period of time since its advent. Until few years ago, there was no single key model attack known which could break full AES-128 better than brute force. In Asiacrypt'11, Bogdanov et al. [7] proposed a novel idea called biclique attack which allowed an attacker to recover the AES secret key 3–5 times faster than exhaustive search. Subsequently, this technique was applied to break many other block ciphers such as PRESENT [1], ARIA [22], HIGHT [10] etc. As block cipher and block cipher based hash function security are inter-related, it is imperative to analyse the hash function security against biclique technique.

Biclique cryptanalysis is a variant of meet-in-the-middle attack, first introduced by Khovratovich et al. in [13] for preimage attacks on hash functions Skein and SHA-2. The concept was taken over by Bogdanov et al. to successfully cryptanalyze full rounds of all AES variants. The biclique attack results on AES in [7] were further improved in [6,20]. Bogdanov et al. in [7] also showed conversion of biclique key recovery attack on AES-128 to the corresponding preimage attack on AES-128 instantiated compression function. The current best complexity of this attack as reported in [6] is $2^{125.56}$.

1.1 Motivation

The above biclique based preimage attack on AES-128 instantiated compression function cannot be converted to preimage attack on the corresponding hash function (and hence second preimage attack as discussed in Sect. 5 later). This is due to the fact that in the preimage attack on compression function shown in [6,7], the attacker needs to modify the chaining variable (CV) value and the message input to obtain the desired preimage. However, in hash function settings, the initialization vector (IV) is a publically known constant which cannot be altered by the attacker. Hence, the biclique trails used in the preimage attack on

AES-128 based compression function in [6,7] cannot be adopted to find preimage for the corresponding AES-128 based hash function. This can be explained as discussed below.

Let us consider Matyas-Meyer-Oseas (MMO) mode and Davies-Meyer (DM) mode based compression functions as shown in Fig. 1(a) and (b). In case of MMO mode, the chaining variable acts as the key input to the underlying block cipher AES (as shown in Fig. 1(a)). If the chaining variable is used as the IV (in hash function settings) then it is fixed and cannot be modified. This means that the value of the key input to the block cipher should not change. However, the type of biclique trails used in [7] (as shown in Fig. 2) for compression function introduce a change both in the key input as well as all the intermediate states including the plaintext input ensuring that the final chaining variable so obtained after the attack will not be the desired IV. Hence, the kind of biclique trails we are interested in should only affect the intermediate states (an example of which is given in Fig. 3) and not the key input.

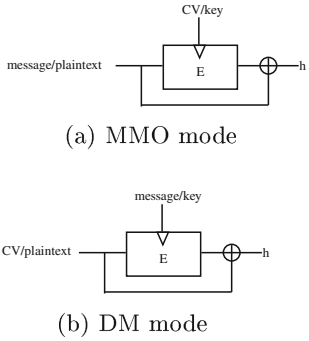


Fig. 1. Compression function in MMO and DM mode respectively.

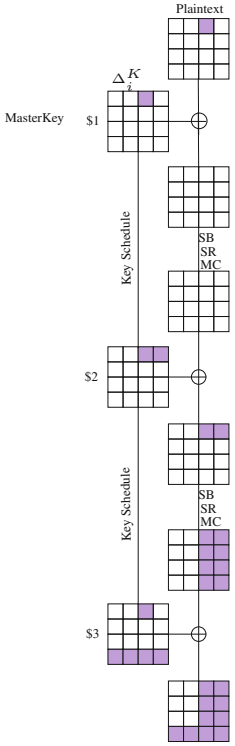


Fig. 2. An example of the trail used in [7] for preimage attack on AES-128 instantiated compression function.

Similarly, in the DM mode, the chaining variable acts as the plaintext input to the underlying block cipher (as shown in Fig. 1(b)). Therefore, if the chaining

variable under consideration is the IV then the chosen biclique trails should not inject any difference in the plaintext input of the block cipher (an example of the same is shown in Fig. 4). Again, the biclique trails adapted for preimage attack on AES-128 instantiated compression function do not satisfy this condition (as seen in Fig. 2).

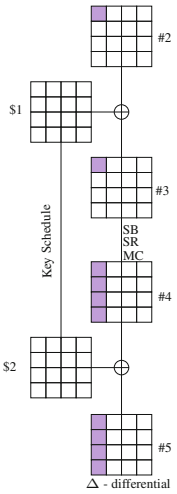


Fig. 3. An example of the desired trails that will work for attacking MMO based hash function. It is to be noted only the plaintext input and subsequent intermediate states are affected in the trail considered whereas the key input is a fixed constant.

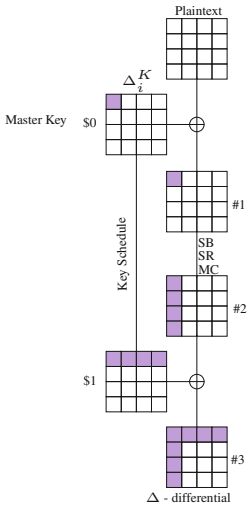


Fig. 4. An example of the desired trail that will work for attacking DM based hash function. It is to be noted here that the plaintext input is not affected by the differential trail so chosen and is a fixed constant.

The examples discussed above warrant searching of new biclique trails which can be used to launch second preimage attack on AES-128 based hash functions. Moreover, searching these trails manually may not give the best results as demonstrated in [2,6]. Hence, automated search process is required. In this work, we implemented our restrictions in C programs to enumerate the best biclique trails which guarantee the lowest possible attack complexities. We then apply biclique technique to evaluate the security of AES-128 based hash functions against second preimage attack.

1.2 Our Contributions

The contributions of this paper are as follows:

– We re-evaluate the offered security of full 10 rounds AES-128 based hash functions against second preimage attack. The previous best result could only work on 7 rounds.

– Our analysis works on all 12 PGV modes of the hash function constructions.
– The complexities of the biclique based analysis differ depending upon the PGV construction chosen. For MP and MMO mode it is $2^{126.3}$ whereas for DM mode it is $2^{126.67}$.
– We propose new biclique trails to achieve the above results.
– All the trails have been obtained by implementing C programs which ensure that they yield the best attacks (lowest possible time complexity).

The results of our security evaluation against second preimage attack on all 12 PGV based modes are given in Table 1.

Table 1. Summary of the results obtained. In this table, we assume hash function to be instantiated with block cipher E, h is the chaining variable, m is the message input and $h \oplus m = w$.

S.No	Hash Function Modes	Second Preimage Complexity	Succ. Prob	Brute Force Complexity	Succ. Prob
1	$E_h(m) \oplus m$ - MMO	$2^{126.3}$	0.632	2^{128}	0.632
2	$E_h(m) \oplus w$ - MP	$2^{126.3}$	0.632	2^{128}	0.632
3	$E_m(h) \oplus h$ - DM	$2^{126.6}$	0.632	2^{128}	0.632
4	$E_h(w) \oplus w$ - similar to MMO	$2^{126.3}$	0.632	2^{128}	0.632
5	$E_h(w) \oplus m$ - similar to MMO	$2^{126.3}$	0.632	2^{128}	0.632
6	$E_m(h) \oplus w$ - similar to DM	$2^{126.6}$	0.632	2^{128}	0.632
7	$E_m(w) \oplus h$ - similar to DM	$2^{126.6}$	0.632	2^{128}	0.632
8	$E_m(w) \oplus w$ - similar to DM	$2^{126.6}$	0.632	2^{128}	0.632
9	$E_w(h) \oplus h$ - similar to DM	$2^{126.6}$	0.632	2^{128}	0.632
10	$E_w(h) \oplus m$ - similar to DM	$2^{126.6}$	0.632	2^{128}	0.632
11	$E_w(m) \oplus h$ - similar to MP	$2^{126.3}$	0.632	2^{128}	0.632
12	$E_w(m) \oplus m$ - similar to MMO	$2^{126.3}$	0.632	2^{128}	0.632

2 Preliminaries

In this section we give a brief overview of the key concepts used in our cryptanalysis technique to facilitate better understanding.

2.1 AES-128

AES-128 is a block cipher with 128-bit internal state and 128-bit key K. The internal state and the key is represented by a 4×4 matrix. The plaintext is xor'ed with the key, and then undergoes a sequence of 10 rounds. Each round consists of four transformations: nonlinear bytewise SubBytes, the byte permutation ShiftRows, linear transformation MixColumns, and the addition with a subkey AddRoundKey. MixColumns is omitted in the last round.

For the sake of clarity, we will follow the same notation used for description of AES-128 as used in [7]. We address two internal states in each round as follows: #1 is the state before SubBytes in round 1, #2 is the state after MixColumns in round 1, #3 is the state before SubBytes in round 2, ..., #19 is the state before SubBytes in round 10, #20 is the state after ShiftRows in round 10. The key K is expanded to a sequence of keys $K^0, K^1, K^2, \ldots, K^{10}$, which form a 4×44 byte array. Then the 128-bit subkeys $\$0, \$1, \$2, \ldots, \$10$ come out of the sliding window with a 4-column step. We refer the reader to [8] for a detailed description of AES.

2.2 Biclique Key Recovery Attack

In this section, we briefly discuss the independent biclique key recovery attack for AES-128. For a more detailed description of bicliques, one can refer to [7]. In this attack, the entire key space of AES-128 is first divided into non-overlapping group of keys. Then, a subcipher f that maps an internal state S to a ciphertext C under a key K, i.e. $f_K(S) = C$ is chosen. Suppose f connects 2^d intermediate states $\{S_j\}$ to 2^d ciphertexts $\{C_i\}$ with 2^{2d} keys $\{K[i,j]\}$. The 3-tuple of sets $[\{S_j\}, \{C_i\}, \{K[i,j]\}]$ is called a d-dimensional biclique, if: $\forall i,j \in \{0, ..., 2^d - 1\} : C_i = f_{K[i,j]}(S_j)$.

Each key in a group can be represented relative to the base key of the group, i.e., $K[0,0]$ and two key differences Δ_i^k and ∇_j^k such that: $K[i,j] = K[0,0] \oplus \Delta_i^k \oplus \nabla_j^k$. For each group we choose a base computation i.e., $S_0 \xrightarrow[f]{K[0,0]} C_0$. Then C_i and S_j are obtained using 2^d forward differentials Δ_i, i.e., $S_0 \xrightarrow[f]{K[0,0] \oplus \Delta_i^k} C_i$ and 2^d backward differentials ∇_j, i.e., $S_j \xleftarrow[f^{-1}]{K[0,0] \oplus \nabla_j^k} C_0$. If the above two differentials do not share active nonlinear components for all i and j, then the following relation: $S_0 \oplus \nabla_j \xrightarrow[f]{K[0,0] \oplus \Delta_i^k \oplus \nabla_j^k} C_0 \oplus \Delta_i$ is satisfied [7]:

Once a biclique is constructed for an arbitrary part of the cipher, meet-in-the middle (MITM) attack is used for the remaining part to recover the key. During the MITM phase, a partial intermediate state is chosen as the matching state v. The adversary then precomputes and stores in memory 2^{d+1} times full computations upto a matching state v: $\forall i, P_i \xrightarrow{K[i,0]} \overrightarrow{v}$ and $\forall j, \overleftarrow{v} \xleftarrow{K[0,j]} S_j$.

Here, plaintext P_i is obtained from ciphertexts C_i through the decryption oracle[1]. If a key in a group satisfies the following relation: $P_i \xrightarrow[h]{K[i,j]} \overrightarrow{v} = \overleftarrow{v} \xleftarrow[g^{-1}]{K[i,j]} S_j$, then the adversary proposes a key candidate. If a right key is not found in the chosen group then another group is chosen and the whole process is repeated. The full complexity of independent biclique attacks is calculated as:

$$C_{full} = 2^{k-2d}(C_{biclique} + C_{precompute} + C_{recompute} + C_{falsepos}),$$

[1] Under hash function settings decryption oracle is replaced by feed-forward operation.

where, $C_{precompute}$ is the cost complexity for calculating v for 2^{d+1}, $C_{recompute}$ is the cost complexity of recomputing v for 2^{2d} times and $C_{falsepos}$ is the complexity to eliminate false positives. As mentioned in [7], the full key recovery complexity is dominated by $2^{k-2d} \times C_{recomp}$[2].

3 Notations

To facilitate better understanding, we use the following notations in the rest of the paper.

$\mathbf{CV}$	: Chaining Variable
$\mathbf{IV}$	: Initialization Vector
$(\mathbf{CV}, \mathbf{message})$	: Input tuple to hash function/compression function
$(\mathbf{key}, \mathbf{plaintext})$	: Input tuple to underlying block cipher
$\mathbf{n}$	: Input message/key size (in bits)
$\mathbf{A_b}$	: Base State
$\mathbf{m_b}$	: Base Plaintext
$\mathbf{K_b}$	: Base Key
$\mathbf{K[i,j]}$	: Keys generated by Δ_i and ∇_j modifications
$\mathbf{M[i,j]}$	: Messages generated by Δ_i and ∇_j modifications
$\mathbf{Nbr}$	: Number of AES rounds called
$\mathbf{E_{enc/dec}}$	: One Round of AES encryption/decryption
$\mathbf{E(x,y)}$	: Full AES encryption under y-bit key and x-bit message
$\mathbf{E^{-1}(x,y)}$	: Full AES decryption under y-bit key and x-bit message

4 Biclique Based Preimage Attack on AES-128 Instantiated Compression Function

In this section, we examine how biclique technique discussed in Sect. 2.2 can be applied to find preimage for block cipher based compression function. This preimage attack on compression function will then be used to evaluate second preimage resistance of AES-128 based hash functions under different PGV modes as discussed in Sect. 5.

Let us consider an AES-128 based compression function (as shown in Fig. 5). To find the preimage for h, the attacker needs to find a valid (CV, message) pair which generates h. In terms of the underlying block cipher E which is instantiated with AES-128, this problem translates to finding a valid (plaintext, key) pair where both the key and the plaintext are of 128-bits size. To guarantee the existence of a preimage for h (with probability 0.632), the attacker needs to test 2^{128} distinct (key, plaintext) pairs.

[2] C_{recomp} in turn is measured as: 2^{128} (#S-boxes recomputed in MITM phase/#Total S-boxes required in one full AES encryption) $\implies$ 2^{128} (#S-boxes recomputed in MITM phase/200).

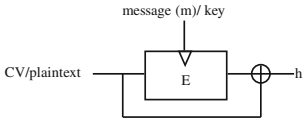


Fig. 5. AES-128 instantiated compression function in DM mode.

When biclique methodology is applied on AES-128 to recover the secret key [7], full key space, i.e., 2^{128} keys are divided into 2^{112} groups of 2^{16} size each and tested[3]. These 2^{112} groups are generated from 2^{112} base key values where each base value defines one group. However, the same biclique approach when extended to hash functions warrants the need of testing 2^{128} (key, plaintext) pairs. These 2^{128} (key, plaintext) pairs will be generated from 2^{112} (key, plaintext) base states. Hence, under hash function settings, alongwith the *base key* we introduce the term "*base message*". Let K_b denote the base key value and A_b denote the base message value. If we apply the original biclique approach [7] on compression function, then 2^{128} (key, plaintext) pairs are generated from a combination of $2^{112}(K_b, A_b)$ as shown in (Fig. 6). Here, a single A_b is chosen and repeated across all the groups whereas 2^{112} different $K_b's$ are used. The biclique algorithm for the attack is shown in Fig. 7. In Algorithm 1, the specific (i,j) tuple for which a match is found gives us the corresponding $K[i,j]$ and $M[i,j]$ as the desired inputs for compression function. The complexity of this attack when applied for searching preimages in AES-128 instantiated compression function is $2^{125.56}$ [6].

$(K_b^{(1)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$(K_b^{(2)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$(K_b^{(3)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$\vdots$
$(K_b^{(2^{112})}, A_b) \longrightarrow 2^{16}$ (key, message) pairs

Fig. 6. Generation of groups in original attack [7]

Algorithm 1 :

Fix a base state A_b
for *each* 2^{112} *base keys* $(K_b's)$ *and the fixed chosen* A_b **do**
  Generate 2^{16} (Δ_i^k, ∇_j^k) combinations
  Generate corresponding $2^{16}K[i,j]$
  Construct a biclique structure using these $2^{16}K[i,j]$
  for *each* $2^{16}K[i,j]$ **do**
    1. Generate $M[i,j]$ (where $M[i,j]$ = Nbr $E_{enc/dec}(K[i,j], A_b)$)
    2. Perform meet-in-the-middle attack in the rest of the rounds

Fig. 7. Steps of the original biclique attack in [7] using the base key K_b and the base message A_b.

[3] Here, bicliques of dimension $d = 8$ are constructed. In our attacks, we also construct bicliques of dimension 8.

In the procedure described above, it can be seen that the attacker generates a chaining value ($M[i,j]$) of her own along with the preimage ($K[i,j]$). However, as already discussed, the IV value is a public constant in the hash function setting and cannot be altered by the attacker. In the subsequent section, we show how to utilize variants of the above framework for launching second preimage attack on AES-128 based hash functions in different PGV modes with IV being fixed.

5 Second Preimage Attack on Hash Functions

In this section, we examine the feasability of extending the biclique cryptanalysis technique for second preimage attack on AES-128 instantiated hash functions for all 12 PGV modes.

5.1 PGV Construction 1 - Matyas-Meyer-Oseas (MMO) Mode: $E_h(\mathbf{m}) \oplus \mathbf{m}$

Consider MMO based hash function as shown in Fig. 8. Here, the (chaining variable, message block) tuple act as the (key, plaintext) inputs respectively to block cipher E. In this case, the attacker is given m = $(m_0 \,||\, m_1 \,||\, pad)$ and its corresponding hash value h_2. Her aim is to find another different message, m' that will produce the same h_2. To achieve so, the attacker can consider m' as - $(m'_0 \,||\, m_1 \,||\, pad)$ where the second half of $m' = m$ while for the first half, the attacker has to carry a biclique attack. For the first half, i.e., $h_1 := E_{IV}(m'_0)$, the attacker knows h_1 and IV. Her aim is now to find a preimage m'_0 which produces h_1 under the given IV. The attack steps are as follows:

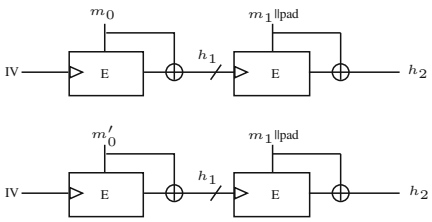
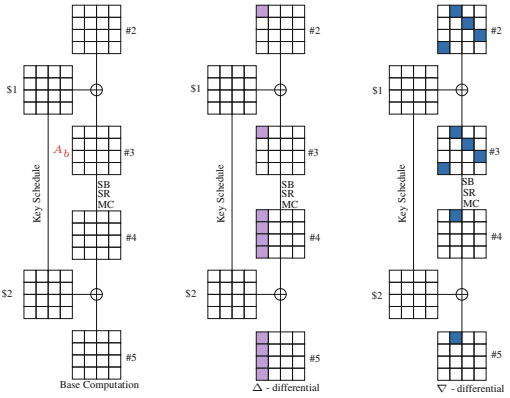


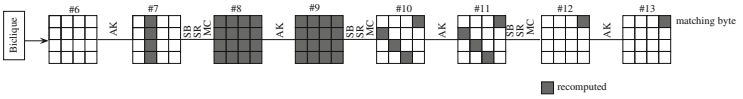
Fig. 8. Second preimage attack on MMO based hash function

1. The attacker fixes IV as the key input to the block cipher E & chooses a 128-bit base message A_b.
2. ***Choice of biclique structure***. Here, the key input to the block cipher (i.e., IV) is fixed. The attacker has to choose a biclique structure such that the Δ_i and ∇_j trails only modify the message states and not the key states (since IV cannot change) plus the biclique attack should have lowest search complexity. All the existing biclique trails in literature allow modification in the keys states as well, therefore, we construct new biclique trails to suit our needs.

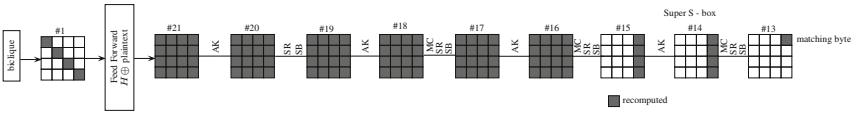
3. We represent the Δ and ∇ trails as Δ_i^m and ∇_j^m respectively. The biclique structure satisfying the above requirements is as shown in Fig. 9(a).
4. For the above biclique, she divides the 128-bit message space into 2^{112} groups each having 2^{16} messages with respect to intermediate state #3 as shown in Fig. 9(a). The base messages are all 16-byte values with two bytes (i.e., bytes 0 and 4) fixed to 0 whereas the remaining 14-bytes taking all possible values (shown in Fig. 10). The messages in each group ($M[i, j]$) are enumerated with respect to the base message by applying difference as shown in Fig. 11. The proof for the claim that this base message (with the corresponding Δ_i and ∇_j differences) uniquely divides the message space into non-overlapping groups is given in Appendix A.1.
5. The biclique covers 1.5 rounds (round 2 and round 3 upto *Shift Rows* operation). Δ_i^m trail activates byte 0 whereas ∇_j^m trail activates bytes 3,4,9 and 14 of #3 state.
6. Meet-in-the-middle attack is performed on the rest 8.5 rounds. In the MITM phase, partial matching is done in byte 12 of state #13. In the backward direction, Δ_i^m trail activates 4 bytes in the plaintext i.e., byte 0, 5, 10 and 15



(a) Biclique over 1.5 rounds



(b) Forward Recomputations



(c) Backward Recomputations

Fig. 9. Biclique structure for MMO mode when key/IV is known

whereas ∇_j^m activates all bytes. As such, during the recomputation phase, the 4 bytes of plaintext affected by both Δ_i^m and ∇_j^m trails need to be recomputed. Similar explanation can be provided for other bytes shown to be recomputed in Fig. 9(b) and (c). In the forward propagation (starting from round 4), 4+16+4 = 24 S-boxes and in the backward propagation (starting from round 1), 4+16+16+4+1= 41 S-boxes are recomputed. Thus, a total of 65 S-boxes are involved in the recomputation process. One full AES encryption requires 200 S-box computations. As each group has 2^{16} messages, $C_{recomp} = 2^{16} \times \frac{65}{200} = 2^{14.3}$. Hence, $C_{full} = 2^{112} \times 2^{14.3} = 2^{126.3}$.

7. For the specific (i, j) value which produces a match in the middle, the corresponding $M[i, j]$ i.e., xoring of #3 states in base computation, Δ_i and ∇_j trails (in Fig. 9(a)) yields the plaintext m'_0 for the block cipher E. The biclique algorithm, i.e., Algorithm 2 is as shown in Fig. 12.

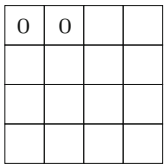


Fig. 10. Base message

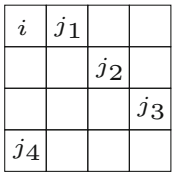


Fig. 11. Δ_i and ∇_j differences

Thus with a time complexity of $\mathbf{2^{126.3}}$, the attacker is able to find a (IV, m'_0) pair which produces hash value h_1 and $m' = (m'_0 \mid\mid m_1 \mid\mid pad)$ forms a valid second preimage.

PGV Construction 2 - Miyaguchi-Preneel Mode (MP) Mode: $\mathbf{E_h(m) \oplus m \oplus h}$ - The MP mode is an extended version of MMO mode. The only difference between the two constructions is the fact that output of block cipher is xor'ed both with the plaintext input as well the chaining variable input. However, this does not demand any extra attack requirements and the second preimage attack on MP mode is exactly the same as that described on MMO mode.

5.2 PGV Construction 3 Davies-Meyer (DM) Mode: $E_m(\mathbf{h}) \oplus \mathbf{h}$

In the DM based hash function (as shown in Fig. 13), the (chaining variable, message block) tuple act as the (plaintext, key) inputs respectively to block cipher E. We again inspect a similar scenario as described in Sect. 5.1, i.e., for a message m = $(m_0 \mid\mid m_1 \mid\mid pad)$, the attacker is given its corresponding hash value h_2. Her aim is to find another different message m' that will produce the same h_2. Consider the hash function as concatenation of two compression functions - $E_{m_0}(IV)$ and $E_{m_1 \mid\mid pad}$ (h_1). To get a valid second preimage, the attacker chooses m' as - $(m'_0 \mid\mid m_1 \mid\mid pad)$ i.e., she focuses on the first compression

Algorithm 2:

Fix a base state A_b
for *each* 2^{112} *base messages* ($A_b's$) *and a single base key i.e., IV* **do**
  Generate 2^{16} (Δ_i^m, ∇_j^m) combinations
  Generate corresponding $2^{16} M[i,j]$
  Construct a biclique structure using these $2^{16} M[i,j]$
  for *each* $2^{16} M[i,j]$ **do**
    1. Perform meet-in-the-middle attack in the rest of the rounds

Fig. 12. Steps of the new biclique attack when key input to the underlying block cipher is fixed and cannot be modified by the attacker for MMO mode.

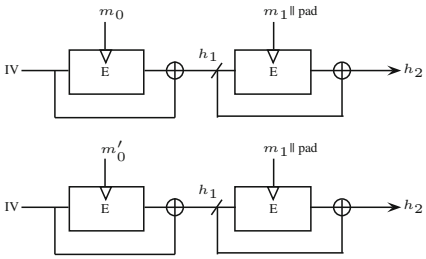
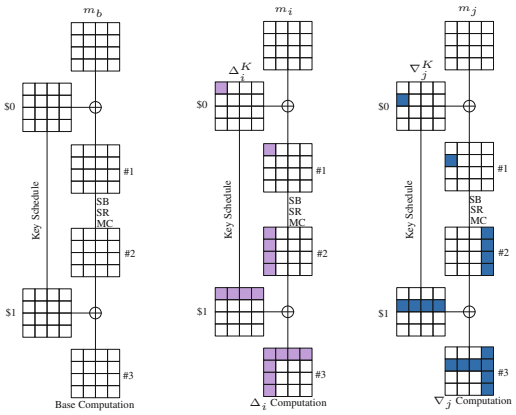


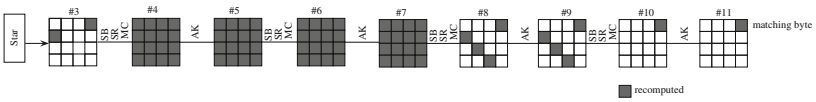
Fig. 13. Second preimage attack on DM based hash function

function and her aim is to find m'_0 such that $E_{m'_0}(IV) = h_1$ when IV and h_1 are known to the attacker. The attack steps are as follows:

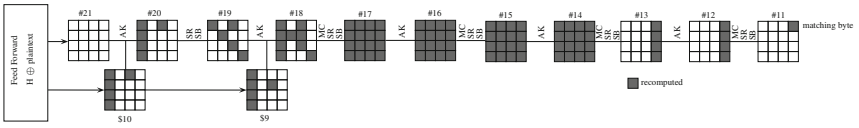
1. The attacker fixes the IV as the plaintext input to the block cipher.
2. ***Choice of biclique structure***. Under the given attack scenario, since the message input, i.e., IV is fixed, the attacker has to choose a biclique structure such that the Δ_i and ∇_j trails do not modify the plaintext state and the biclique attack has lowest search complexity. The biclique structure satisfying the above requirements is given in Fig. 14(a).
3. For the above biclique, she divides the 128-bit key space into 2^{112} groups, each having 2^{16} keys with respect to subkey \$0 i.e., the master key as the base key as shown in Fig. 14(a). The base keys are all 16-byte values with two bytes (i.e., bytes 0 and 1) fixed to 0 whereas the remaining 14-bytes taking all possible values (shown in Fig. 15). The keys in each group ($K[i,j]$) are enumerated with respect to the base key by applying difference as shown in Fig. 16. It can be easily verified that this base key uniquely divides the key space into non-overlapping groups.
4. The biclique covers the first round. Δ_i trail activates byte 0 of \$0 subkey whereas ∇_j trail activates byte 1 of \$ 0 subkey.



(a) Biclique over first round



(b) Forward Recomputations



(c) Backward Recomputations

Fig. 14. Biclique structure for DM mode when IV/message input is known to the attacker

5. The attacker then performs meet-in-the-middle attack on the rest of the 9 rounds. In the MITM phase, partial matching is done in byte 12 of state #11. In the forward propagation (starting from round 2), 2+16+16+4 = 38 S-boxes and in the backward propagation (starting from round 10), 5+16+16+4+1 = 42 S-boxes need to be recomputed (as shown in Fig. 14(b) and (c)). 2 S-box recomputations in the key schedule are also required. Thus a total of 82 S-boxes are involved in recomputation process. One full AES encryption requires 200 S-box computations. As each group has 2^{16} keys, $C_{recomp} = 2^{16} \times \frac{82}{200} = 2^{14.6}$. Hence, $C_{full} = 2^{112} \times 2^{14.6} = 2^{126.6}$.
6. For the specific (i, j) value which produces a match in the middle, the corresponding $K[i, j]$ forms the key (m_0) for the block cipher E. The biclique algorithm, i.e., Algorithm 3 is given in Fig. 17.

Thus with a time complexity of $\mathbf{2^{126.6}}$, the attacker is able to find a (IV, m'_0) pair which produces hash value h_1 and $m' = (m'_0 \mid\mid m_1 \mid\mid pad)$ forms a valid second preimage. The attack procedure on two block message for other

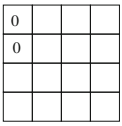


Fig. 15. Base message

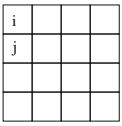


Fig. 16. Δ_i and ∇_j differences

Algorithm 3 :

for *each* 2^{112} *base keys* ($K_b's$) *and the fixed chosen IV* **do**
- Generate 2^{16} (Δ_i^k, ∇_j^k) combinations
- Generate the corresponding $2^{16}K[i,j]$
- Construct a biclique structure using these $2^{16}K[i,j]$
- **for** *each* $2^{16}K[i,j]$ **do**
  1. Perform meet-in-the-middle attack in the rest of the rounds

Fig. 17. Steps of the new biclique attack when message input is fixed and known to the attacker under DM mode

constructions is similar to those discussed in Sects. 5.1 and 5.2. Their results are given in Table 1.

6 Second Preimage Attack on Hash Functions Extended to Messages with Message Length ≥3

The second preimage attack discussed in above sections can be extended to messages of any length >2 with same complexity as obtained for 2-block messages. To demonstrate the same, consider a MMO-based hash function with 3-block message as shown in Fig. 18. In this case, the attacker is given a message $m = (m_0 \mid\mid m_1 \mid\mid m_2 \mid\mid pad)$ and its corresponding hash value h_3. Her aim is to find another message m', such that $H(m') = H(m)$. The attacker knows IV and the compression function E. She will choose any m_0 of her own choice, e.g., let $m_0 = 0$, and then calculate $h_1 = E_{IV}(0)$. Once she knows h_1, the setting is reduced to the case discussed in Sect. 5.1, i.e., h_1 and h_2 are known to the attacker and her aim is to find m_1' such that $m' = (0 \mid\mid m_1' \mid\mid m_2 \mid\mid pad)$ forms a valid second preimage. This can be found with a complexity of $2^{126.3}$ which is same as that shown for a 2-block message. Similarly, the attack can be applied on other long messages for all other PGV modes.

7 Conclusions

In this paper, we evaluate the security of AES-128 based hash modes against second preimage attack. Specifically, we examine the applicability of biclique

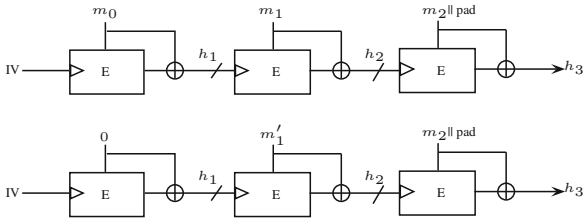


Fig. 18. MMO base hash function with $|m| = 3$

attack on all 12 PGV modes when instantiated with AES-128 and show that best biclique attack for finding preimages in AES-128 instantiated compression function does not translate to best attack for second preimage search under AES-128 based hash function settings. A natural research extension to this work would be to apply the ideas discussed in this paper to hash functions instantiated with other block ciphers. Another research direction can be to extend the methodology to carry out collision attacks on hash functions.

A Proofs

In this section, we will prove how the base structure which we chose for bicliques in Sect. 5.1 produce non-overlapping keys/messages within a same group and between groups.

A.1 Biclique Structure When *IV* Is Known and Acts as the Message Input to Block Cipher E

For the base message (shown in Fig. 10) that is used for the biclique structure in Fig. 9(a), our aim is to prove that when Δ_i and ∇_j differences are injected in this base message (as shown in Fig. 19), we are able to partition the message space into 2^{112} groups with 2^{16} messages in each and the inter and intra group messages generated are non-overlapping. The ∇j_1, ∇j_2, ∇j_3 and ∇j_4 are differences produced from ∇j as shown in Fig. 20.

Here, $b_{i,j}$ and $c_{i,j}$ $(0 \leq \text{i,j} \leq 3)$ represent the base values of corresponding bytes in the intermediate states #B and #C respectively as shown in Fig. 21. #B and #C are #3 and #4 states in Fig. 9(a).

Aim: Given any two base messages B, B', any two Δ_i differences i, i', any two ∇_j differences j, j' $(0 \leq \text{i,j} \leq 2^8)$, we want to prove that $\text{B[i,j]} \neq B[i', j']$ i.e., messages generated are non-overlapping. We will prove this statement case-by-case. Cases (1–4) cover inter group messages whereas Cases (5–7) cover within group messages. For all the proofs discussed below, we will refer to Figs. 22, 23 and 24 for better understanding.

Case 1. Given $B \neq B'$, $i = i', j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, to show: $B[i, j] \neq B'[i', j']$

i	j_1		
		j_2	
			j_3
j_4			

Fig. 19. Δ_i and ∇_j differences in base message

$$\begin{pmatrix} b_{01} \oplus j_1 \\ b_{12} \oplus j_2 \\ b_{23} \oplus j_3 \\ b_{30} \oplus j_4 \end{pmatrix} = ISB, ISR, IMC \begin{pmatrix} c_{01} \oplus j \\ c_{12} \\ c_{23} \\ c_{30} \end{pmatrix}$$

Fig. 20. Relation between $\nabla j, \nabla j_1, \nabla j_2, \nabla j_3, \nabla j_4$

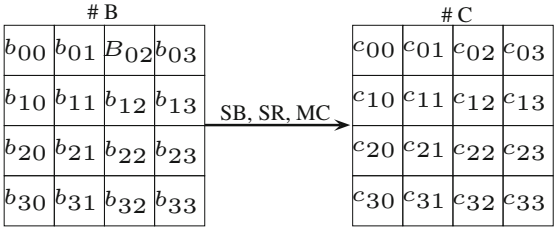


Fig. 21. Relation between #B and #C states

Proof: We will prove this setting by 'proof by contraposition', i.e., if $B[i,j] = B'[i',j']$, $i = i', j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, $\implies$ $B = B'$

In Fig. 24, if $B[i,j] = B'[i',j'] \implies C[i,j] = C'[i',j'] \implies c_{0,2} = c'_{0,2}$, $c_{0,3} = c'_{0,3}$, $c_{1,1} = c'_{1,1}$, $c_{1,2} = c'_{1,2}$, $c_{1,3} = c'_{1,3}$, $c_{2,1} = c'_{2,1}$, $c_{2,2} = c'_{2,2}$,$c_{2,3} = c'_{2,3}$, $c_{3,1} = c'_{3,1}$, $c_{3,2} = c'_{3,2}$ and $c_{3,3} = c'_{3,3}$. Since $C[i,j] = C'[i',j'] \implies c_{0,1} \oplus j = c'_{0,1} \oplus j'$. As $j = j' \implies c_{0,1} = c'_{0,1}$. Hence, **12** bytes in state C and corresponding bytes in state C' share equal values. This relation automatically transcends to related byte positions in B and B' after application of *InvMixColumns*, *InvShiftRows* and *InvSubBytes* (as shown in Fig. 22), i.e., $b_{0,1} = b'_{0,1}$, $b_{0,2} = b'_{0,2}$, $b_{0,3} = b'_{0,3}$, $b_{1,0} = b'_{1,0}$, $b_{1,2} = b'_{1,2}$, $b_{1,3} = b'_{1,3}$, $b_{2,0} = b'_{2,0}$, $b_{2,1} = b'_{2,1}$, $b_{2,3} = b'_{2,3}$, $b_{3,0} = b'_{3,0}$, $b_{3,1} = b'_{3,1}$ and $b_{3,2} = b'_{3,2}$, 12 bytes in B and B' respectively also have same base values). As we have assumed $B[i,j] = B'[i',j'] \implies b_{1,1} = b'_{1,1}$, $b_{2,2} = b'_{2,2}$ and $b_{3,3} = b'_{3,3}$ as these base values are not affected by Δ_i and ∇_j differences (as seen in Fig. 24). Since in states B and B', $b_{0,0} = b'_{0,0} = 0$, hence all **16** byte positions in B and corresponding byte positions in B' share same base values. Hence $B = B'$. This proves that our initial proposition is correct.

Case 2. Given $B \neq B'$, $i = i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$

Proof: We will prove this setting by 'proof by contradiction', i.e., let us assume if $B \neq B'$, $i = i'$, $j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, $\implies$ $B[i,j] = B'[i',j']$

In Fig. 24, if $B[i,j] = B'[i',j'] \implies C[i,j] = C'[i',j'] \implies c_{0,1} \oplus j = c'_{0,1} \oplus j'$. Since j $\neq$ j' $\implies$ $c_{0,1} \neq c'_{0,1}$. As a result after applying *InvMixColumns* and *InvSubBytes* on them the bytes generated i.e., $b_{0,1}$ and $b'_{0,1}$ should also satisfy the relation - $b_{0,1} \neq b'_{0,1}$. But $b_{0,1} = b'_{0,1} = 0$ (as seen in Fig. 21). Hence, a

contradiction arises implying our assumed proposition is wrong. Therefore, our initial proposition is correct.

Case 3. Given $B \neq B'$, $i \neq i'$, $j = j'$, $b_{00}{=}b_{01}{=}b'_{00}{=}b'_{01}{=}0$, to show: $B[i,j] \neq B'[i',j']$

Proof: In this setting since $i \neq i'$, hence $B[i,j] \neq B'[i',j']$ always as they will always differ at zeroth byte position (Fig. 24).

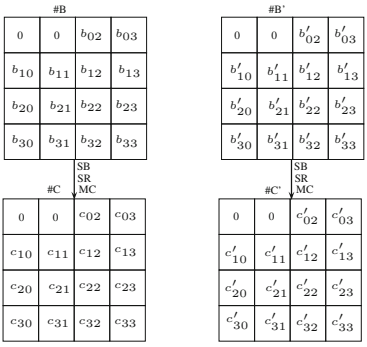


Fig. 22. Relation between base states B and C. The labels inside each box denote the base values of the corresponding byte positions

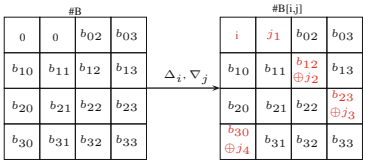


Fig. 23. Modification of state $\#B$ after applying Δ_i and ∇_j differences. Same relation exists between $\#B'$ and $\#B'[i,j]$

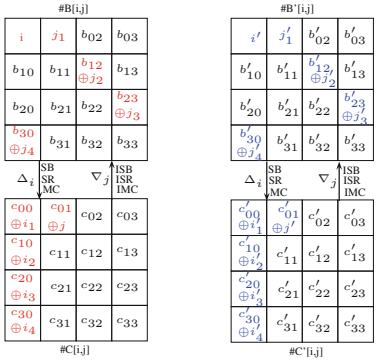


Fig. 24. Relation between states $\#B[i,j]$, $\#C[i,j]$ and $\#B'[i,j]$, $\#C'[i,j]$

Case 4. Given $B \neq B'$, $i \neq i'$, $j \neq j'$, $b_{00}{=}b_{01}{=}b'_{00}{=}b'_{01}{=}0$, to show: $B[i,j] \neq B'[i',j']$

Proof: Proof similar to as discussed in *Case 3.*

Case 5. Given $B = B'$, $i \neq i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$

Proof: Proof similar to as discussed in *Case 3.*

Case 6. Given $B = B'$, $i \neq i'$, $j = j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$

Proof: Proof similar to as discussed in *Case 3.*

Case 7. Given $B = B'$, $i = i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$

Proof: Since $B = B' \implies C = C' \implies c_{0,1} = c'_{0,1}$. As $j \neq j' \implies c_{0,1} \oplus j \neq c'_{0,1} \oplus j' \implies C[i,j] \neq C'[i',j']$ always as they will everytime differ at fourth byte position (Fig. 24). As a result $B[i,j] \neq B'[i',j']$ always due to bijection relation between states B and C.

Hence we proved that in all cases $M[i,j]$'s so generated are non-overlapping.

References

1. Abed, F., Forler, C., List, E., Lucks, S., Wenzel, J.: Biclique cryptanalysis of the PRESENT and LED lightweight ciphers. IACR Cryptology ePrint Archive, 2012:591 (2012)
2. Abed, F., Forler, C., List, E., Lucks, S., Wenzel, J.: A framework for automated independent-biclique cryptanalysis. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 561–582. Springer, Heidelberg (2014)
3. Barreto, P.S.L.M., Rijmen, V.: Whirlpool. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, 2nd edn, pp. 1384–1385. Springer US, New York (2011)
4. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (2008)
5. Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An analysis of the blockcipher-based hash functions from PGV. J. Cryptology **23**(4), 519–545 (2010)
6. Bogdanov, A., Chang, D., Ghosh, M., Sanadhya, S.K.: Bicliques with minimal data and time complexity for AES. In: Lee, J., Kim, J. (eds.) ICISC 2014. LNCS, vol. 8949, pp. 160–174. Springer, Heidelberg (2011)
7. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002)
9. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl - a SHA-3 candidate. In: Symmetric Cryptography, Dagstuhl Seminar Proceedings, Dagstuhl, Germany (2009)
10. Hong, D., Koo, B., Kwon, D.: Biclique attack on the full HIGHT. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 365–374. Springer, Heidelberg (2012)
11. Indesteege, S.: The LANE Hash Function. Submission to NIST (2008)
12. Jean, J., Naya-Plasencia, M., Schläffer, M.: Improved analysis of ECHO-256. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 19–36. Springer, Heidelberg (2011)

13. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: attacks on skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)
14. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The rebound attack and subspace distinguishers: application to whirlpool. IACR Cryptology ePrint Archive, 2010:198 (2010)
15. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full Lane compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 106–125. Springer, Heidelberg (2009)
16. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attacks on the reduced `Grøstl` hash function. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 350–365. Springer, Heidelberg (2010)
17. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
18. Sasaki, Y., Yasuda, K.: Known-key distinguishers on 11-round feistel and collision attacks on its hashing modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 397–415. Springer, Heidelberg (2011)
19. Schläffer, M.: Subspace distinguisher for 5/8 rounds of the `ECHO`-256 hash function. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 369–387. Springer, Heidelberg (2011)
20. Tao, B., Wu, H.: Improving the biclique cryptanalysis of AES. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 39–56. Springer, Heidelberg (2015)
21. Wu, S., Feng, D., Wu, W.: Cryptanalysis of the LANE hash function. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 126–140. Springer, Heidelberg (2009)
22. Chen, S.Z., Xu, T.M.: Biclique attack of the full ARIA-256. IACR Cryptology ePrint Archive, 2012:11 (2012)