Biclique Cryptanalysis of full round AES-128 based hashing modes

Donghoon Chang, Mohona Ghosh, Somitra Sanadhya

Indraprastha Institute of Information Technology
donghoon, mohonag,somitra@iiitd.ac.in

Abstract. In this work, we revisit the security analysis of AES-128 instantiated hash modes. We use biclique cryptanalysis technique as our basis for the attack. The traditional biclique approach used for key recovery in AES (and preimage search in AES based compression function) cannot be applied directly to hash function settings due to restrictions imposed on message input due to padding. Under this criteria, we show how to translate biclique technique to hash domain and demonstrate preimage and second preimage attack on all 12 PGV modes. Our preimage attack complexity for all PGV modes stands at $2^{127.4}$. The second preimage attack complexities differ based on the PGV construction chosen - the lowest being $2^{126.3}$ and the highest being $2^{126.67}$ complexity. We also show how to model our attacks under different settings, e.g., when message is padded/ not padded, when chaining variable is known/not known, when full message or key space is available/ not available to the attacker etc. Our attacks require only 2 message blocks with padding included and works on full 10 rounds of AES-128 for all 12 PGV modes. In our attacks, the IV is assumed to be a known constant which is a practical assumption but knowledge of other chaining variables is not required for the attacker. Considering these, our results can be termed as the best so far in literature. Though our attack 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.

Keywords: AES, block ciphers, hash functions, cryptanalysis, biclique, preimage attack.

1 Introduction

Block ciphers have been favored as cryptographic primitives for constructing hash functions for a long time. In [1], 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. [2] analyzed the security of such constructions and showed 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 [4], ECHO [5], GR∅STL [6] 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 [7,8,9,10,11,12,13]. 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 [7], 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 recently, there was no single key model attack known which could break full AES-128 better than brute force attack. In Asiacrypt'11, Bogdanov et al. [14] proposed a novel idea called biclique attack which allowed an attacker to recover AES secret key up to 3-5 times faster than exhaustive search. Since then the technique has garnered considerable interest amongst cryptographic community. This approach, which is a variant of meet-in-the-middle attack, was first introduced by Khovratovich et al. in [15]

for preimage attacks on hash functions Skein and SHA-2. The concept was taken over by Bogdanov et al. to successfully cryptanalyze full round AES and has been subsequently adopted to break many other block ciphers [16,17,18,19,20]. As block cipher and block cipher based hash function security are inter-related, it is imperative to analyse hash function security against biclique technique.

Motivation. In [14], Bogadanov et al. have mentioned that with a computational complexity of $2^{125.83}$, biclique key recovery attack on AES-128 can be converted to preimage attack on AES-128 based compression function in MP mode. However, this biclique attack on compression function cannot be directly extended to block cipher based hash functions primarily for two reasons:

- Firstly, under hash function settings message padding norms are exercised, i.e., often message inputs are padded in order to make them conform to a certain length (as shown in Fig. 1). Once a message is padded, its padding bits are fixed and cannot change. The biclique attack on compression function in [14] does not consider this restriction and allows modification of padding bits, i.e., the preimage so generated need not fulfill the padding criteria.
- Secondly, the biclique attack methodology in [14] warrants utilization of full key space to work successfully, i.e., if the block cipher uses a k-bit key then the attack requires availability of all 2^k keys to the attacker. However, under hash function settings this might not be possible in all cases especially when the key input to underlying block cipher acts as message input for the hash function so built (as seen in Fig. 1). Here, since the length of padding bits are fixed, the attacker does not have control on all key bits, i.e., if r bits are fixed for padding, then attacker has only $(k-r)$ degrees of freedom and can work with only 2^{k-r} keys.

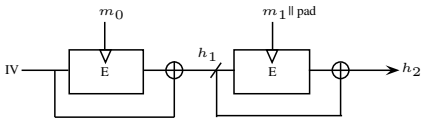


Fig. 1. Here, m_1 has been padded to satisfy the message length requirements. m_0 and $m_1||$pad which are message inputs to the hash functions are infact key inputs to the underlying block cipher E.

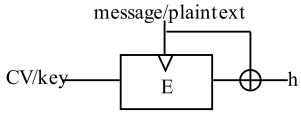


Fig. 2. AES-128 instantiated compression function in DM mode. This figure is used in § 4

Under such scenarios, in our work, we demonstrate how biclique cryptanalysis can still be utilised to successfully launch preimage and second preimage attacks on AES-128 based hash functions and show that the biclique structure which yields best results in compression function cryptanalysis do not deliver the best results under the corresponding hash function settings. Moreover, in our attack we assume IV to be a public constant but all other chaining variables if involved are unknown to the attacker. The previous best result [7] works only under known key settings where IV and all the chaining variables are known to the attacker. As a consequence, their preimage attack and second preimage attacks can target only specific PGV constructions and most of them require long messages (message block ≥ 3). Our attacks, don't have such restrictions. Principally, we suggest a framework for biclique cryptanalysis on hash functions and show how it can be applied to attack PGV modes under different conditions.

Our contribution: The contributions of this paper are as follows:

- We propose preimage and second preimage attacks on full 10 rounds of AES-128 based hash modes. The attacks work on all 12 PGV hash function constructions.
- The preimage attack complexities for all the three frequently used modes, i.e., DM, MMO and MP modes is $2^{127.38}$, $2^{127.45}$ and $2^{127.45}$ respectively.
- The second preimage attack complexity similarly differs based on PGV construction chosen. For MP and MMO mode it is $2^{126.3}$ whereas for DM mode it is $2^{126.67}$.

– All the above attacks are performed on **2** block messages (with pad) and can be easily extended to messages of length $\geq$ **3** with the same attack complexities.
– We also show how our attacks can be modelled under all possible scenarios, e.g., when message is padded/not padded, when chaining variable is known/not known, when full message/key space is available/not available to the attacker etc.

The results of cryptanalytic attacks on all 12 PGV based modes is given in Table 1.

S.No.	Hash Function Modes	Preimage Complexity	Second Preimage Complexity
1	$E_h(m) \oplus m$ - MMO	$2^{127.45}$	$2^{126.3}$
2	$E_h(m) \oplus w$ - MP	$2^{127.45}$	$2^{126.3}$
3	$E_m(h) \oplus h$ - DM	$2^{127.38}$	$2^{126.67}$
4	$E_h(w) \oplus w$	$2^{127.45}$	$2^{126.3}$
5	$E_h(w) \oplus m$ - similar to Constr.4	$2^{127.45}$	$2^{126.3}$
6	$E_m(h) \oplus w$ - similar to DM	$2^{127.38}$	$2^{126.67}$
7	$E_m(w) \oplus h$ - similar to DM	$2^{127.38}$	$2^{126.67}$
8	$E_m(w) \oplus w$ - similar to DM	$2^{127.38}$	$2^{126.67}$
9	$E_w(h) \oplus h$	$2^{127.56}$	$2^{126.67}$
10	$E_w(h) \oplus m$ - similar to Constr.9	$2^{127.56}$	$2^{126.67}$
11	$E_w(m) \oplus h$ - similar to MP	$2^{127.45}$	$2^{126.3}$
12	$E_w(m) \oplus m$ - similar to MMO	$2^{127.45}$	$2^{126.3}$

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 inpuut and $h \oplus m = w$

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 [14]. 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 [21] 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 [14]. 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 [14]:

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 she 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:

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

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 [14], 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.

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

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

[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)

4 Biclique Modus Operandi for Preimage Attack on Hash Function

In this section, we examine how biclique technique discussed in Subec 2.2 can be applied on hash functions. Let us consider an AES-128 based compression function(as shown in Fig. 2). To find a preimage h, the attacker needs to find a valid (CV, message) pair which generates h. In terms of underlying block cipher E which is instantiated with AES-128, this problem translates to finding a valid (key, plaintext) pair where both key and 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 precisely.

When biclique methodology is applied on AES-128 to recover the secret key [14], 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 [14] 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. 3). Here, a single A_b is chosen and repeated across

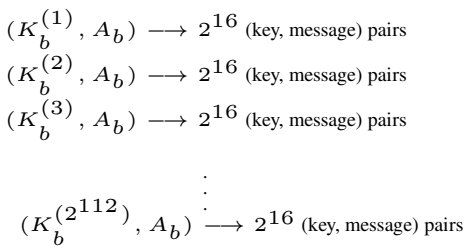


Fig. 3. Generation of groups in original attack [14]

Algorithm 1 :

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

Fig. 4. Steps in the original biclique attack in [14]

all the groups whereas 2^{112} different $K_b's$ are used. The biclique algorithm for the attack is shown in Fig.4. In *Algorithm1*, 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 $2^{125.83}$ [14].

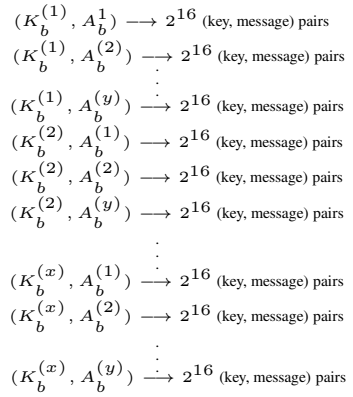


Fig. 5. Generation of groups under hash function settings

Algorithm 2 :

for *each* 2^x *base keys* ($K_b's$) **do**
  for *each* 2^y *base messages* (A_b) **do**
    for 2^{16} (Δ_i^k, ∇_j^k) *combinations* **do**
      Generate K[i,j]
      Construct biclique structure
      Generate M[i,j] (where M[i,j] = Nbr $E_{enc/dec}$(K[i,j], A_b))
      Perform meet-in-the-middle attack

Fig. 6. Steps of the modified biclique attack under hash function settings

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

However as discussed in § 1, cases may arise where an attacker does not have the freedom to use all 2^{128} keys. Let us suppose, due to message padding, the attacker has only 2^r keys with her (where $r<128$). Let us further assume that these 2^r keys are generated from 2^x base key values, i.e., $2^{r-16} = 2^x$ base keys. If the attacker fixes a base message A_b of her own choice and only varies K_b, then she can generate only 2^x (K_b, A_b) pairs (where $x<112$). In such scenarios, to reach her threshold of 2^{112} base states, the attacker will then choose 2^y base messages (A_b's) such that $2^x \times 2^y = 2^{112}$. Now the $2^{112}(K_b, A_b)$ base value pairs will be generated as shown in Fig. 5. The corresponding modified biclique algorithm is shown in Fig. 6. The attack complexity is now calculated as : $2^{x+y}(2^{16} \times C_{recomp})$.

In § 1, it was also discussed that preimage search for hash functions using biclique technique is favorable when message inputs obey padding rules. In the subsequent sections, we will show how this constraint is handled depending on the hash mode considered and apply the modified algorithm (shown in Fig. 6) to launch successful attacks.

5 Preimage Attack on Hash Functions

In this section, we examine the feasability of extending the biclique attack on AES-128 instantiated hash functions under all 12 PGV modes and discuss how bicliques can be applied when padding of messages is involved. We adopt the standard padding convention for all messages, i.e., append to a message m (represented in bits) a single 1-bit followed by zero or more 0-bits. A length field of 64 bits is appended at last to specify the bit-length of the original message. To give maximum freedom to the attacker, we omit the intermediate 0-bits in padding and directly append the length field after '1' bit, i.e., **65** bits are fixed for padding. We discuss 4 specific PGV models, rest 8 follow the same attack procedure as applied to above four. The attacks are demonstrated on 2-block messages ($m_0||(m_1||\text{pad})$) and in all the cases final hash value and initialization vector(IV) which is a public constant are known to the attacker. In our attack demonstration, since 2-block messages are considered and padding requires 65 bits, the attacker will always try to generate a 191-bit messages (128+128-65 = 128+63) i.e., the messages will be of the form -

$$message(m_0||m_1||pad) = \underbrace{m_0}_{\text{128-bit}} \;||\; \underbrace{m_1}_{\text{63-bit}} \;||\; \underbrace{1||\;\overbrace{00000....00010111111}^{\text{binary representation of 191 in 64 bits}}}_{\text{65-bit pad}}$$

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

In the MMO based hash function (as shown in Fig. 7), the (chaining variable, message block) tuple

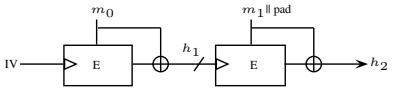


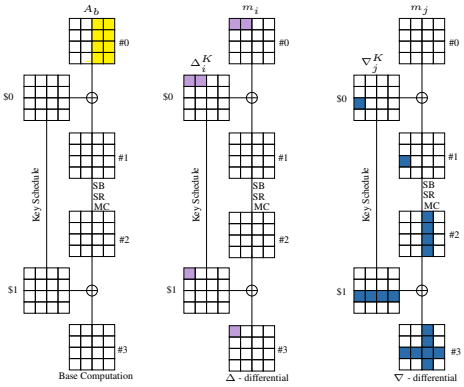
Fig. 7. Hash function based on MMO mode

act as the (key, plaintext) inputs respectively to block cipher E. For better understanding we divide the hash function into 2 compression functions - each function being governed by its (key, plaintext) pair, i.e., $E_{IV}(m_0)$ and $E_{h1}(m_1||pad)$. We look at both cases separately and then merge them together to attack the hash function as a whole.

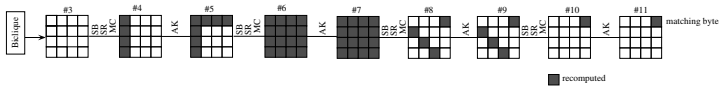
Case A. h2 := $\mathbf{E_{h1}(m_1||pad)}$ - Under this scenario, attacker knows h_2 and her aim is to find a valid (h_1, m_1||pad) pair. The key h_1 is unknown to her. The attack steps are as follows:

1. The attacker chooses a 128-bit base message A_b such that its last 65 bits follow the padding rule (i.e., the value of last 65 bits is fixed).

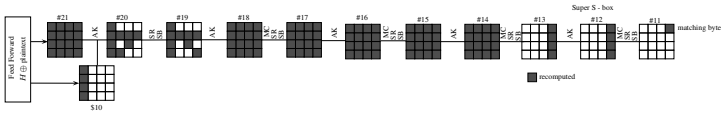
2. *Choice of biclique structure*. The attacker has to choose a biclique structure such that Δ_i^K and ∇_j^K trails do not modify the last 65 padding bits of the base message. Hence, he strives to choose a structure which not only has the lowest search complexities but also allow non-alteration of padding bits. The biclique structure satisfying the above two requirements is as follows (Fig. 8(a)):
3. The biclique covers the first round. Δ_i trail activates bytes 0 and 4 of \$0 sub key whereas ∇_j trail activates byte 2 of \$0 subkey. \$0 is taken as the base key (K_b) and #0 is taken as the base state (A_b).



(a) Biclique over first round . In this biclique, the boxes colored in yellow show the padding bits. As can be seen, Δ_i and ∇_j trails do not modify these padding bits.



(b) Forward Recomputations



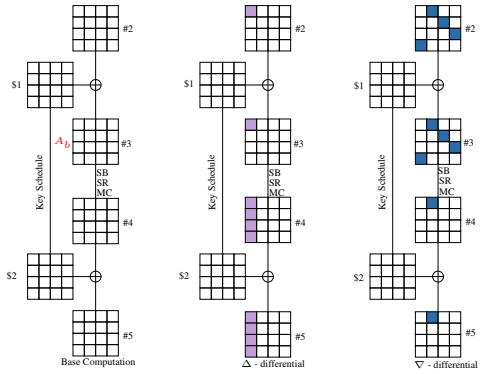
(c) Backward Recomputations

Fig. 8. Biclique structure for MMO mode when padding in message is necessary.

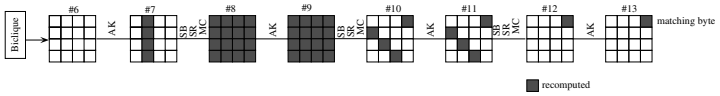
4. The attacker then performs meet-in-the-middle attack on the rest of the 9 rounds. In the MITM phase, partial matching is done in byte 12 of state #11. In the forward propagation (starting from round 2), 7+16+4 = 27 S-boxes and in the backward propagation (starting from round 10), 9+16+16+4+1 = 46 S-boxes need to be recomputed (in Figs. 8(b), 8(c)). Thus, a total of 73 S-boxes are involved in recomputation process. As each group has 2^{16} keys, therefore, for each group $C_{recomp} = 2^{16} \times \frac{73}{200} = 2^{14.54}$. Since we match on 1 byte, i.e., 8 bits in v, we have 2^8 false positives on an average. Similarly, $C_{biclique} = 2^{5.67} \approx 2^9 \times \frac{1}{10}$ and $C_{precomp} = 2^{7.68}$. Hence, $C_{full} = 2^{112} \times (2^{5.67} + 2^{7.68} + 2^{14.54} + 2^8) = 2^{126.54}$. The biclique algorithm followed is same as shown in Fig. 4.
5. For the specific (i,j) value which produces a match in the middle, the corresponding xoring of m_i, m_j and A_b (as shown in Fig. 8(a)) yields the message input ($m1$||pad) for the compression function and $K[i,j]$ - the desired h_1. Thus with a time complexity of $\mathbf{2^{126.54}}$, the attacker is able to find a (h_1, $m1$||pad) pair which produces hash value h_2.

Case B. h1 := $\mathbf{E_{IV}(m_0)}$ - In this case, the attacker has already obtained the value of h_1 from the above mentioned attack. Public constant IV is known to her. Her aim is now to find a preimage m_0 which produces h_1 under the given value of IV. The attack steps are as followed:

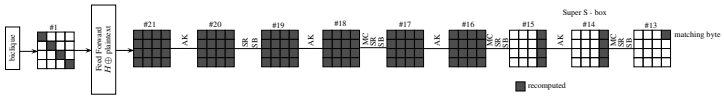
1. The attacker fixes IV as the key input to the block cipher & chooses a 128-bit base message A_b.
2. *Choice of biclique structure*. Here, the key input to the block cipher (i.e., IV) is fixed. The attacker has to choose a biclique structure such that the Δ_i and ∇_j trails only modify the message states and not the key states plus the biclique attack should have lowest search complexity. 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).
3. 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



(a) Biclique over 1.5 rounds



(b) Forward Recomputations



(c) Backward Recomputations

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

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

0	0		

Fig. 10. Base Message

i	j_1		
		j_2	
			j_3
j_4			

Fig. 11. Δ_i and ∇_j differences

Algorithm 3:

Fix a base state A_b
for *each* 2^{112} *base keys* ($A_b's$) *and key as IV* **do**
  for 2^{16} (Δ_i^m, ∇_j^m) *combinations* **do**
    (a) Generate M[i,j]
    (b) Construct biclique structure
    (c) Perform meet-in-the-middle attack

Fig. 12. Steps of the modified biclique attack

4. The biclique covers 1.5 rounds (round 2 and round 3 upto *Shift Rows* operation). Δ_i^m trail activates byte 0 ∇_j^m trail activates bytes 3,4,9 and 14 of #3 state.
5. Meet-in-the-middle attack is performed on the rest 8.5 rounds. In the MITM phase, partial matching is done in byte 0 of state #13. 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 (in Figs. 9(b), 9(c)). Thus, a total of 65 S-boxes are involved in recomputation process. Hence $C_{full} \approx 2^{126.37}$.
6. 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 (Fig. 7). The biclique algorithm, i.e., *Algorithm 3* is shown in Fig. 12.

When the above mentioned two attacks are merged together (i.e., case A followed by case B) the total attack complexity is - $2^{126.54} + 2^{126.37} = 2^{127.45}$. Thus, with a complexity of $\mathbf{2^{127.45}}$, the attacker is able to successfully launch preimage attack on MMO based hash function.

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 demands any extra attack requirements and the attack on MP mode is exactly 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 the hash function as concatenation of two compression functions - $E_{m_0}(IV)$ and $E_{m_1||pad}(h1)$.

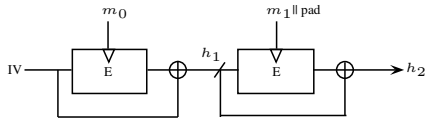
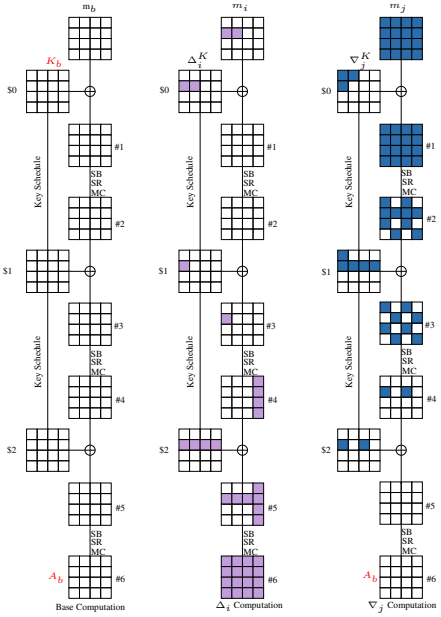


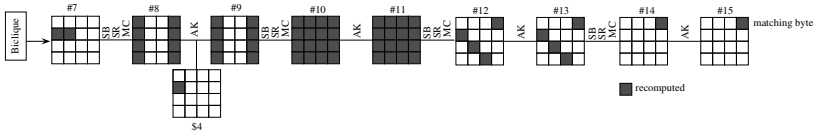
Fig. 13. Hash function based on DM mode

CaseA. $\mathbf{h_2 := E_{m_1||pad}(h1)}$ - Here, the attacker knows h_2 and his aim is to find valid (m_1||pad, h_1) pair. Here h_1 (plaintext input) and m_1 (key input) both are unknown to the attacker. The attack steps are as follows:

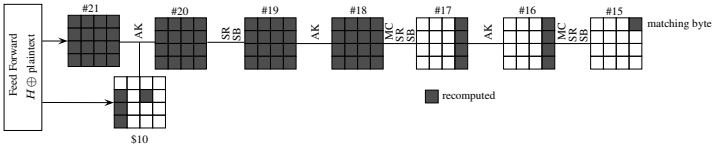
1. The attacker chooses and fixes a 128-bit base message A_b.
2. She is interested in those keys for which the last 65 bits follow the padding rule, hence total number of such keys are $2^{128-65} = 2^{63}$ i.e., the attacker has limited number of key choices.
3. *Choice of biclique structure.* Under the given attack scenario, the attacker has to choose a biclique structure such that the Δ_i and ∇_j trails do not modify the last 65 padding bits of the base key. Secondly, she has to consider the \$0 subkey (i.e., the master key) as her base key to ensure the integrity of the padding bits. Thirdly, the biclique attack should have minimal complexity. The biclique structure satisfying the above requirements is as shown in (Fig. 14(a)). The resulting biclique structure covers first 3 rounds of AES-128.



(a) Biclique over first 3 rounds



(b) Forward Recomputations



(c) Backward Recomputations

Fig. 14. Biclique structure for DM mode when padding in key is necessary

4. For constructing this biclique, \$0 subkey is chosen to be the base key (K_b) as shown in Fig. 15.The keys (K[i,j]) in each group are enumerated by applying difference as shown in Figs. 16 and 17 [4] Δ_i trail activates byte 1 and 5 of \$0 subkey whereas ∇_j trail activates bytes 0,1 and 4 of \$0 subkey. As per the biclique attack rule, the keys in one group should not overlap with each other as well as with keys of all other groups. To ensure the non-overlapping property of keys, the attacker only generates 2^{31} $K_b's$ seen in Fig. 18 (as against $2^{63-16} = 2^{47}$ base keys). In Fig. 18, the 31 bits marked in green generate the 2^{31} $K_b's$ which define the 2^{31} groups respectively.
5. For the attacker to find atleast one preimage with success probability 0.632, she should have 2^{112} (K_b, A_b) pair inputs. At present, the attacker can generate only 2^{31} (K_b, A_b) pairs with $2^{31} K_b's$. Hence, to fulfill her threshold of 2^{112} pairs, she considers 2^{81} base messages A_b (as shown in Fig. 18). The proof for the claim that 2^{128} (key, plaintext) pairs so generated (shown in Fig. 19) are distinct and non-overlapping is given in Appendix A.2.
6. In the MITM phase, partial matching is done in byte 12 of state #15. In the forward propagation, 2+8+16+4 = 30 S-boxes and in the backward propagation, 16+4+1= 21 S-boxes are required to be recomputed (in Figs. 14(b), 14(c)). Thus a total of 51 S-boxes are involved in the recomputation process. Hence $C_{full} \approx 2^{126.02}$.

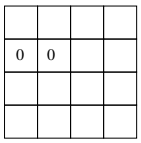


Fig. 15. Base Key

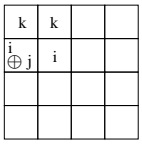


Fig. 16. Δ_i and ∇_j differences

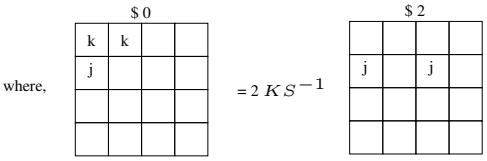


Fig. 17. Influence of ∇_j difference injected in \$2 on base key \$0

7. Thus with a time complexity of $\mathbf{2^{126.02}}$, the attacker is able to find a (m_1||pad, h_1) pair which produces hash value h_2. For the specific (i,j) value which produces a match in the middle, the key K[i,j] forms m_1//*pad* and the corresponding M[i,j] forms h_1 for the block cipher E (see Fig. 13). The biclique algorithm, i.e., *Algorithm 4* is shown in Fig. 20.

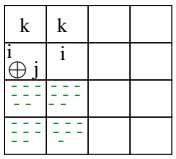


Fig. 18. Base keys are defined based on bits marked in green

$2^{31} K_b's \times 2^{81} A_b's \longrightarrow 2^{112}$ (K_b, A_b) pairs

1(K_b, A_b) pair $\longrightarrow 2^{16}$ (K[i,j], M[i,j]) pairs

2^{112} (K_b, A_b) pairs $\longrightarrow 2^{128}$ (K[i,j], M[i,j]) pairs

Fig. 19. Generation of K_b, A_b pairs

Algorithm 4 :

for *each* 2^{31} *base keys* ($K_b's$) **do**
  for *each* 2^{81} *base messages* ($A_b's$) **do**
    Generate $m_b = 3E_{dec}(K_b, A_b)$
    for 2^{16} (Δ_i^k, ∇_j^k) *combinations* **do**
      1. Generate K[i,j]
      2. Construct biclique structure
      3. Generate M[i,j] (where M[i,j] = $m_b \oplus m_i \oplus m_j$)
      4. Perform meet-in-the-middle attack

Fig. 20. Steps of the modified biclique attack

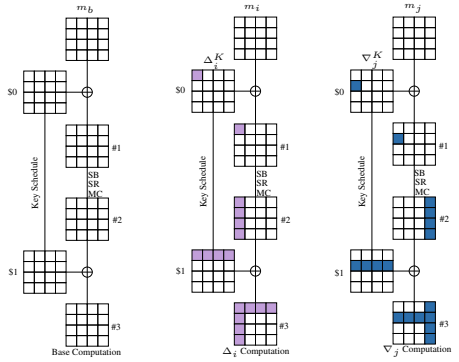
Case B. $\mathbf{h1 := E_{m_0}(IV)}$ - Here, the attacker knows the value of h_1. Public constant IV is already known to her. Her aim is now to find a preimage m_0 which produces the chaining variable h_1 under the given value of IV. The attack steps are as follows:

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

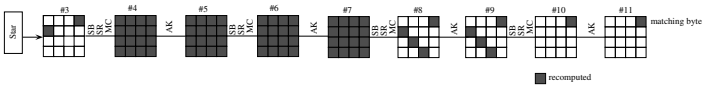
[4] Here KS^{-1} represents inverse key schedule algorithm of AES-128.

the plaintext state and the biclique attack has lowest search complexity. The biclique structure satisfying the above requirements is given in 21(a).

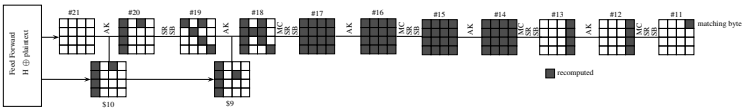
3. The biclique covers the first round. Δ_i trail activates byte 0 of \$0 sub key whereas ∇_j trail activates byte 1 of \$ 0 subkey. The biclique algorithm, i.e., *Algorithm 6* is given in Appendix B.
4. The attacker then performs meet-in-the-middle attack on the rest of the 9 rounds. In the MITM phase, 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 Figs. 21(b) and 21(c)). Thus a total of 80 S-boxes are involved in recomputation process. Thus, $C_{full} \approx 2^{126.67}$.
5. 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. Thus with a time complexity of $\mathbf{2^{126.67}}$, the attacker is able to find a (IV, m_0) pair which produces hash value h_1.



(a) Biclique over first round



(b) Forward Recomputations



(c) Backward Recomputations

Fig. 21. Biclique structure for DM mode when IV/message input is known to the attacker

When the above mentioned two attacks are merged together (i.e., case A followed by case B) the total attack complexity is - $2^{126.02} + 2^{126.67} = 2^{127.38}$. Thus, with a complexity of $\mathbf{2^{127.38}}$, the attacker is able to successfully launch preimage attack on DM based hash function.

5.3 PGV Construction 4 - $E_h(w) \oplus w$, where w = h ⊕ m

In the PGV construction 4 based compression function (as shown in Fig. 22), the chaining variable acts as the key input to the block cipher E. The xor'ed output of chaining variable and message i.e., (x/y) goes as as plaintext input to E. The output ciphertext is xored with x/y to produce the next hash value. Here also the attacker is given h_2 and IV value is known to him. We again inspect the hash function as concatenation of two compression functions - $E_{IV}(x)$ and $E_{h_1}(y)$.

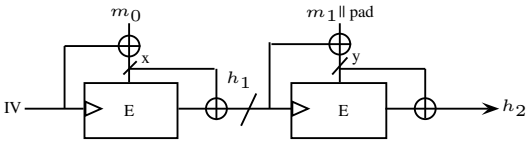


Fig. 22. Hash function based on PGV construction 4 mode

Algorithm 5 :

for *each* 2^{112} *base keys* ($K_b's$) **do**
  for 2^{16} (Δ_i^k, ∇_j^k) *combinations* **do**

1. Generate K[i,j]
2. Choose A_b such that $(A_b)_{64.....128} \oplus (K_b)_{64.....128} = pad$
3. Construct biclique structure
4. Generate M[i,j] (where M[i,j] = $A_b \oplus m_i$)
5. Perform meet-in-the-middle attack

Fig. 23. Steps of the modified biclique attack

Case A. h2 $:= \mathbf{E_{h_1}(y)}$ - Under this setting, the attacker knows h_2 and his aim is to find valid (y, h_1) pair. Here y (plaintext input) and h_1 (key input) both are unknown to the attacker. The y message so found will then be xor'ed with h_1 to generate $m_1||pad$. The attack steps are as followed:

1. *Choice of biclique structure.*Under this scenario, the attacker has to choose h_1 and y such that when h_1 and y are xor'ed, the last 65 bits of the xor'ed output follow the padding convention (since xor'ed output is $m_1||$pad). Let us suppose the attacker utilizes all 2^{128} keys. Then based on each of the 2^{128} keys i.e., K[i,j], corresponding y's i.e., M[i,j] will be computed such that $K[i,j]_{64.....128} \oplus M[i,j]_{64.....128} = pad$. The attacker has to choose a biclique structure such that Δ_i^K and ∇_j^K trails do not affect the the last 65 bits of h_1 and y (since their xoring produces the actual pad that attacker is interested in). The attacker also has to choose \$0 subkey as the base K_b to ensure non-alteration of last 65 bits of h_1. Lastly, attacker will try to choose a biclique structure that yields minimal complexities. The biclique structure satisfying all the requirements is same as shown in Fig. 8(a). The biclique algorithm is as shown in Fig. 23, Algorithm 5.
2. The complexity of this attack is $\mathbf{2^{126.54}}$. The y (i.e., $M[i,j]$) so found will be xor'ed with h_1 to get the actual $m_1||pad$.

Case B. h1 $:= \mathbf{E_{IV}(x)}$ - The attack procedure to find x when IV and h_1 are known to the attacker is exactly the same as discussed in § 5.1, Case B. Hence attack complexity is $2^{126.3}$.

When the above mentioned two attacks are merged together (i.e., case A followed by case B) the total attack complexity is - $2^{126.54} + 2^{126.3} = 2^{127.45}$. Thus, with a complexity of $\mathbf{2^{127.45}}$, the attacker is able to successfully launch preimage attack on PGV construction 4 based hash function.

The attack procedure on two block message for other constructions is similar to those discussed in § 5.1-§ 5.3. Their results are given in Table 2.

S.No.	Compression Function	Case A	Case B	Complexity
1	$E_h(m) \oplus m$ - MMO	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$
2	$E_h(m) \oplus w$ - MP	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$
3	$E_m(h) \oplus h$ - DM	$2^{126.02}$	$2^{126.67}$	$2^{127.38}$
4	$E_h(w) \oplus w$	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$
5	$E_h(w) \oplus m$ - similar to Constr.4	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$
6	$E_m(h) \oplus w$ - similar to DM	$2^{126.02}$	$2^{126.67}$	$2^{127.38}$
7	$E_m(w) \oplus h$ - similar to DM	$2^{126.02}$	$2^{126.67}$	$2^{127.38}$
8	$E_m(w) \oplus w$ - similar to DM	$2^{126.02}$	$2^{126.67}$	$2^{127.38}$
9	$E_w(h) \oplus h$			$2^{127.59}$
	Case A - similar to Constr.4, case A	$2^{126.54}$		
	Case B - similar to DM, case B		$2^{126.67}$	
10	$E_w(h) \oplus m$ - similar to Constr.9	$2^{126.54}$	$2^{126.67}$	$2^{127.59}$
11	$E_w(m) \oplus h$ - similar to MP	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$
12	$E_w(m) \oplus m$ - similar to MMO	$2^{126.54}$	$2^{126.3}$	$2^{127.45}$

Table 2. Results of preimage attack (case-wise) on all PGV modes of hash functions

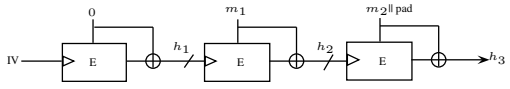


Fig. 24. MMO base hash function with |m|=3

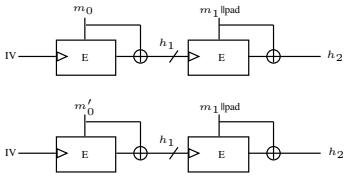


Fig. 25. Second Preimage Attack

6 Other Attacks

Preimage attack on hash functions extended to messages with message length $\geq$ ***3.*** The 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. 24. In this case the attacker knows IV and the compression function E. She'll 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 § 5.1, i.e., preimage $(0||m_1||m_2||\text{pad})$ can be found with complexity $2^{127.45}$. Similarly, the attack can be applied on other long messages under different PGV modes.

Second Preimage Attack on Hash Functions. Consider again MMO based hash function in Fig. 25. In this case the attacker is given m = $(m_0||m_1||\text{pad})$ and 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 - $(m'_0||m_1||\text{pad})$ where the second half of m'=m while for the first half, the attacker has to carry a biclique attack. Since attacker knows IV and h_1, she can find m'_0 with a search complexity of $2^{126.3}$ (similar to case B in MMO mode in Section 6.1). Thus , overall a second preimage m' for MMO can be found with a complexity of $2^{126.3}$. The second preimage attack results for other PGV constructions are given in Table 1.

7 Conclusion

In this paper, we discuss the challenges in applying the traditional biclique technique on block cipher based hash functions when message padding is involved and suggest solutions to overcome them. Specifically, we examine preimage and second preimage attacks on all 12 PGV modes when instantiated with AES-128 and show that best biclique attack for key recovery attack in AES-128 doesn't translate to best attack for 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.

References

1. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.
2. John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An Analysis of the Blockcipher-Based Hash Functions from PGV. *J. Cryptology*, 23(4):519–545, 2010.
3. Paulo S. L. M. Barreto and Vincent Rijmen. Whirlpool. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1384–1385. Springer, 2011.
4. Sebastiaan Indesteege. The LANE hash function. Submission to NIST, 2008.
5. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST, 2008.
6. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Thomsen Søren S. Grøstl - a SHA-3 candidate. In *Symmetric Cryptography*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009.
7. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013.
8. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolic, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full Lane Compression Function. In *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2009.
9. Shuang Wu, Dengguo Feng, and Wenling Wu. Cryptanalysis of the LANE Hash Function. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2009.
10. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. *IACR Cryptology ePrint Archive*, 2010:198, 2010.
11. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 350–365. Springer, 2010.
12. Martin Schläffer. Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. In *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 369–387. Springer, 2010.

13. Jérémy Jean, María Naya-Plasencia, and Martin Schläffer. Improved analysis of echo-256. In *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2011.
14. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *Asiacrypt 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
15. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *Fast Software Encryption 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.
16. Shao zhen Chen and Tian min Xu. Biclique Attack of the Full ARIA-256. *IACR Cryptology ePrint Archive*, 2012:11, 2012.
17. Hamid Mala. Biclique Cryptanalysis of the Block Cipher SQUARE. *IACR Cryptology ePrint Archive*, 2011:500, 2011.
18. Mustafa Çoban, Ferhat Karakoç, and Özkan Boztas. Biclique Cryptanalysis of TWINE. *IACR Cryptology ePrint Archive*, 2012:422, 2012.
19. Deukjo Hong, Bonwook Koo, and Daesung Kwon. Biclique Attack on the Full HIGHT. In *ICISC 2011*, volume 7259 of *Lecture Notes in Computer Science*, pages 365–374. Springer, 2011.
20. Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique Cryptanalysis of the PRESENT and LED Lightweight Ciphers. *IACR Cryptology ePrint Archive*, 2012:591, 2012.
21. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

A Proofs

In this section, we will prove how the base structure which we chose for bicliques in §. 5.1, *Case B* and § 5.2, *Case A* 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. 8(a), our aim is to prove that when Δ_i and ∇_j differences are injected in this base message (as shown in Fig. 26), 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. 27.

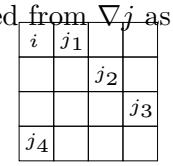


Fig. 26. Δ_i and ∇_j differences in base message

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

Fig. 27. Relation between $\nabla j, \nabla j_1, \nabla j_2, \nabla j_3, \nabla j_4$

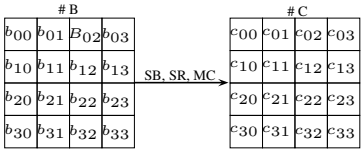


Fig. 28. Relation between #B and #C states

Here, $b_{i,j}$ and $c_{i,j}$ ($0 \leq$ i,j ≤ 3) represent the base values of corresponding bytes in the intermediate states #B and #C respectively as shown in Fig. 28. #B and #C are #3 and #4 states in Fig. 8(a).

Aim: Given any two base messages B, B', any two Δ_i differences i, i', any two ∇_j differences j, j' ($0 \leq$ i,j $\leq 2^8$), we want to prove that 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'll refer to Fig. 29, 30, 31 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']

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. 31, 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]

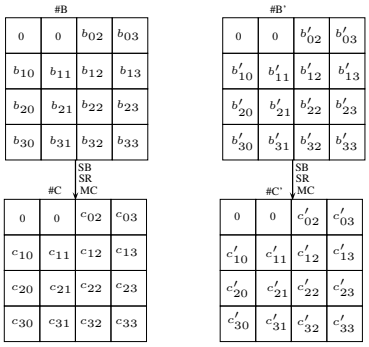


Fig. 29. Relation between base states B and C. The labels inside each box denote the base values of the corresponding byte positions

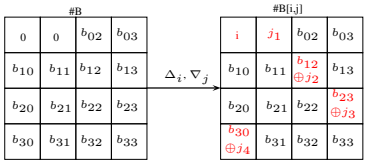


Fig. 30. Modification of state #B after applying Δ_i and ∇_j differences. Same relation exists between #B' and #B'[i,j]

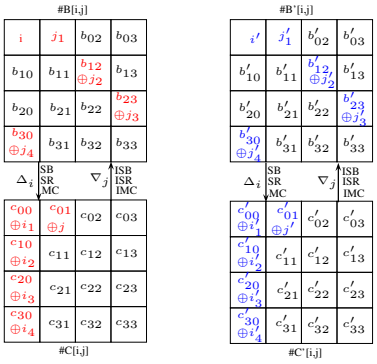


Fig. 31. Relation between states #B [i,j], #C [i,j]and #B' [i,j], #C' [i,j]

= 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. 29), 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 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. 31). 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. 31, 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. 28). 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. 31).

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 ≠ j', b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: B[i,j] ≠ B'[i',j']
Proof: Since B = B' $\implies$ C = C' $\implies$ $c_{0,1} = c'_{0,1}$. As j ≠ j' $\implies$ $c_{0,1} \oplus j \neq c'_{0,1} \oplus j'$ $\implies$ C[i,j] ≠ C'[i',j'] always as they will everytime differ at fourth byte position (Fig. 31). As a result B[i,j] ≠ 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.

A.2 Biclique Structure when (message,chaining variable) tuple acts as (key, plaintext) input to block cipher E

In Fig. 14(a), for the biclique structure we had chosen, the base key K was given as shown in Fig. 32. We had then shown that when Δ_i and ∇_j differences are injected in the base key (as shown in Fig. 33), we were able to generate only 2^{31} base key values. However to find a preimage with high probability 2^{128} (key, plaintext pairs) are needed. To achieve the same we had chosen 2^{81} base messages and then generated 2^{112} groups each defined by 2^{112} (base key, base message) pairs which ultimately produced 2^{128} (key, plaintext pairs). Our goal is to prove that the 2^{128} tuples so produced are distinct and non-overlapping. Let A_b and K_b denote the base message and base key values respectively. Let M[i,j] = $3E_{dec}$(K[0,j], A_b) $\oplus\Delta_i^K$ (see Fig. 14(a)).

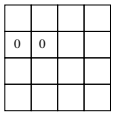


Fig. 32. Base Message

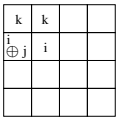


Fig. 33. Δ_i and ∇_j differences

Aim: Given any two base keys K_b, K'_b, any two base messages A_b, A'_b, any two Δ_i^K differences i, i', any two ∇_j^K differences j, j' ($0 \leq$ i,j $\leq 2^8$), we want to prove that (K[i,j], M[i,j]) ≠ (K'[i',j'], M'[i',j']).

Case1. Given $K_b \neq K'_b$, to prove (K[i,j], M[i,j]) ≠ (K'[i',j'], M'[i',j'])
Proof. Since $K_b \neq K'_b \implies K[i,j] \neq K'[i',j'] \implies (K[i,j], M[i,j]) \neq$ (K'[i',j'], M'[i',j']) always.

Case2. Given $K_b = K'_b$, $A_b \neq A'_b$, i=i', j=j', to prove (K[i,j], M[i,j]) ≠ (K[i',j'], M'[i',j'])
Proof. Since i = i', j = j' $\implies$ key part in any two pairs is same, i.e., tuples are of the form [(K[i,j], M[i,j]), (K[i,j], M'[i',j'])]. Now, $M[i,j] = 3E_{dec}(K[0,j], A_b) \oplus \Delta_i^K$ (1)
$M'[i',j'] = 3E_{dec}(K[0,j], A'_b) \oplus \Delta_i^K$ (2)
Since $A_b \neq A'_b$, from equations (1) and (2) $\implies$ M[i,j] ≠ M'[i',j'] (since L.H.S of both equations are same except base message part).

Case3. Given $K_b = K'_b$, $A_b \neq A'_b$, i≠i', j=j', to prove (K[i,j], M[i,j]) ≠ (K[i',j'], M'[i',j'])
Proof. Since i≠i' and $K_b = K'_b \implies K[i,j] \neq K[i',j] \implies (K[i,j], M[i,j]) \neq$ (K[i',j'], M'[i',j'])

Case4. Given $K_b = K'_b$, $A_b \neq A'_b$, j≠j', i=i', to prove (K[i,j], M[i,j]) ≠ (K[i',j'], M'[i',j'])
Proof. Since j≠j' and $K_b = K'_b \implies K[i,j] \neq K[i,j'] \implies (K[i,j], M[i,j]) \neq$ (K[i',j'], M'[i',j'])

Case5. Given $K_b = K'_b$, $A_b \neq A'_b$, j≠j', i≠i', to prove (K[i,j], M[i,j]) ≠ (K[i',j'], M'[i',j'])
Proof. Since i≠i', j≠j' and $K_b = K'_b \implies K[i,j] \neq K[i',j'] \implies (K[i,j], M[i,j]) \neq$ (K[i',j'], M'[i',j'])

Hence we proved that in all cases tuples (K[i,j], M[i,j]) so generated are non-overlapping.

B Biclique Algorithm for DM mode when IV/message is known

In this section, we describe the biclique attack algorithm for finding preimage when IV and hash output are known to the attacker under DM mode.

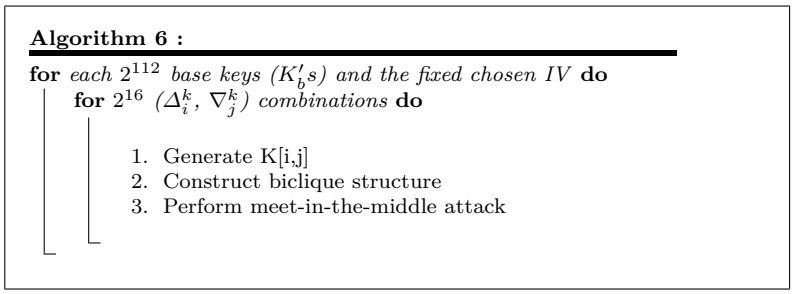


Fig. 34. Steps of the biclique attack when message input is fixed and known to the attacker under DM mode