IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 4, pp. 318–347. DOI:10.13154/tosc.v2019.i4.318-347

Improved Meet-in-the-Middle Preimage Attacks against AES Hashing Modes

Zhenzhen Bao[1,2], Lin Ding[3], Jian Guo[1], Haoyang Wang[1(✉)] and Wenying Zhang[1,4]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
{zzbao,guojian}@ntu.edu.sg,wang1153@e.ntu.edu.sg
[2] Strategic Centre for Research in Privacy-Preserving Technologies and Systems, Nanyang Technological University, Singapore, Singapore
[3] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
dinglin@sjtu.edu.cn
[4] School of Information Science and Engineering, Shandong Normal University, Jinan, China
zhangwenying@sdnu.edu.cn

Abstract. Hashing modes are ways to convert a block cipher into a hash function, and those with AES as the underlying block cipher are referred to as AES hashing modes. Sasaki in 2011, introduced the first preimage attack against AES hashing modes with the AES block cipher reduced to 7 rounds, by the method of meet-in-the-middle. In his attack, the key-schedules are not taken into account. Hence, the same attack applies to all three versions of AES. In this paper, by introducing neutral bits from the key, extra degree of freedom is gained, which is utilized in two ways, *i.e.*, to reduce the time complexity and to extend the attack to more rounds. As an immediate result, the complexities of 7-round pseudo-preimage attacks are reduced from 2^{120} to 2^{104}, 2^{96}, and 2^{96} for AES-128, AES-192, and AES-256, respectively. By carefully choosing the neutral bits from the key to cancel those from the state, the attack is extended to 8 rounds for AES-192 and AES-256 with complexities 2^{112} and 2^{96}. Similar results are obtained for Kiasu-BC, a tweakable block cipher based on AES-128, and interestingly the additional input tweak helps reduce the complexity and extend the attack to one more round. To the best of our knowledge, these are the first preimage attacks against 8-round AES hashing modes.

Keywords: AES · MITM · preimage · hashing mode · key-schedule

1 Introduction

The Advanced Encryption Standard (AES). Designed by Daemen and Rijmen in 1998 [AES01], and later formally standardized by the U.S. National Institute of Standards and Technology (NIST) in 2001, AES becomes the most widely deployed block cipher nowadays in the world among both industry and government agencies, for its long-standing security against massive cryptanalysis and efficiency in both software and hardware. There are three variants, and according to the key sizes and hence security level in bits from the set $\{128, 192, 256\}$, they are named as AES-128, AES-192, and AES-256, respectively.

The PGV Hashing Modes. Hashing modes are ways to convert block ciphers to compression functions, and then to hash functions under some domain extensions. Compared

Received: 2019-09-01, Accepted: 2019-11-01, Published: 2020-01-31

with designs from scratch, hashing modes enjoy both inherited security and performance efficiencies from the underlying block ciphers. Security proofs of hashing modes, which deduce the security, such as collision resistance of a hash function to the security of the underlying block cipher, ensure that no attack against the hash function could be possible before an attack against the underlying block cipher is found. This removes the hassle of intensive cryptanalysis required by hash functions designed from scratch, when the hashing mode is instantiated by a secure block cipher such as AES. For environments like resource-constrained hardware where a block cipher is already implemented, a hashing mode could be the most economical way to achieve a hash function for purposes like digest and signature, since in most of the cases implementing a hashing mode on top of an existing block cipher costs much lesser than that of a standalone hash function.

In [PGV94], Preneel, Govaerts, and Vandewalle summarized that there are 12 secure ways to convert a block cipher to a compression function, and these constructions are referred to as PGV modes nowadays after the name of the authors. Out of them, there are three modes named Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO), Miyaguchi-Preneel (MP) commonly used in practice. For instance, all the hash functions in the MD-SHA hash family, including MD4, MD5, SHA-1, and SHA-2, can be viewed as DM modes.

Use-Cases of AES Hashing Modes. Using block cipher MMO-mode to build hash functions is one of ISO/IEC standardized methods [ISO10], which specifies how to use an n-bit block cipher building a hash function with an n-bit digest. In [ISO10], such a block cipher-based hash function (Hash function 1) is instantiated using AES as an example. Besides, the instantiation of DM-mode by AES was once submitted to NIST as a formal proposal under the name AES-Hash [CL01]. Furthermore, there are cryptographic protocols with rich functionality using hash functions as building blocks, e.g., Symmetric Searchable Encryption (SSE), which is designed to achieve the best trade-offs between security, efficiency, and functionality. In [Bos16, BMO17], secure SSE schemes are proposed, which use hash functions with a requirement of preimage resistance. In particular, it is crucial that the hash functions are pre-image resistant for the security of the schemes [Bos16]. And, in the implementation of the proposed SSE scheme named Diana, the hash function is the AES block cipher used in Matyas-Meyer-Oseas mode [BMO17]. [1] Due to the standardization, the well-studied security of PGV-mode, and the AES-NI instructions in modern CPUs, hash functions built based on AES PGV-mode may have also attracted other real-life cryptographic applications that require loose but finely controllable security. A hash function with a 128-bit internal state has potential applications in cases where it requires (second-)preimage resistance but explicitly does not require collision resistance (e.g., hashing with short-input for signature schemes [KLMR16]).

The MITM Preimage Attack. The preimage attack against a hash function is to find a message whose digest equals to the given value. The most naive way is to randomly select a message, evaluate its digest and check against the given value. This brute-force method costs 2^n hash evaluations for an n-bit hash function. A method running faster than 2^n is considered as an attack. Sasaki and Aoki introduced the Meet-in-the-Middle (MITM) preimage attack in 2008 [SA08], and the technique was extended and used to break the theoretical preimage security claims of MD4 [GLRW10a], MD5 [SA09], Tiger [WS10, GLRW10a], HAVAL [SA08, GSY15] and round-reduced variants of many other hash functions such as SHA-0 and SHA-1 [AS09a, KK12, EFK15], SHA-2 [AGM+09], BLAKE [EFK15], HAS-160 [HKS10], RIPEMD and RIPEMD-160 [WSK+11],

[1] Corresponding open-source libraries are released, and one can find the implementation of the hash function built using AES MMO-mode via https://gitlab.com/sse/crypto/blob/master/src/block_hash.cpp, and https://github.com/jgharehchamani/SSE/blob/master/third_party/crypto/src/block_hash.cpp. We found that in the newest repository moved to GitHub, the hash function used has been replaced with BLAKE2b and SHA-512, see https://github.com/OpenSSE/crypto-tk/blob/master/src/hash.cpp.

Stribog [AY14a], and Whirlwind [AY14b]. It is interesting to see that the idea of MITM preimage attacks also leads to the progress of collision attacks against reduced SHA-2 [LIS12]. These preimage attacks are generally converted from MITM pseudo-preimage attacks by using a generic algorithm, where pseudo-preimage attacks are attacks with unconstrained IVs (one allows free choice of IVs). Thus, formally, denote the hash function by $\mathcal{H}$, for pseudo-preimage attacks, given a target T, the goal is to find an initialization vector or chaining value V and a message M such that $\mathcal{H}(V, M) = T$.

MITM Preimage Attack against AES Hashing Modes. AES hashing modes refer to the hashing modes instantiated by AES block cipher. In 2011, Sasaki [Sas11] introduced the first attack against AES hashing modes with the underlying AES reduced to 7 rounds and the last round without the MixColumns operation. The complexity of the attack was sightly improved by Wu *et al.* [WFW+12] in 2012. To the best of our knowledge, there is no more public progress on this topic since then.

The general idea of MITM preimage attack is to split the cipher (or compression function) into two independent chunks, which can be computed independently from each other with respect to some *neutral bits*. Technically, the source of neutral bits of most previous works is the key bits of the block cipher or the message bits of the compression function. However, in [Sas11], the neutral bits are chosen from the state while the key bits are not used and fixed to some random constants. The key bits are not used because finding neutral bits in the key is difficult due to the key-schedule which diffuses all key bits quickly. It is then natural to ask whether it is possible to eventually find neutral bits from key to either improve the attack complexity or to extend the number of attacked rounds. In this paper, we achieve both.

Our Contributions. On one hand, additional neutral bits from the key improves the attack in two directions, *i.e.*, improving the time complexity directly due to more neutral bits and extending the attack to more rounds since local collisions of neutral bits from the encryption state and the key state are possible. On the other hand, to avoid dealing with the quick diffusion of the key-schedule of AES, we choose neutral bits from the key state for one chunk v.s. for both chunks. This is possible thanks to the improvement of the attack in [WFW+12], where the unbalanced 8 and 32 bits neutral bits are found for the two chunks. The additional neutral bits from key makes it closer to the balanced 32 and 32 bits, which improves the time complexity of the final attack by a factor of at most $2^{32-8} = 2^{24}$.

Larger key sizes allow more degrees of freedom for the choices of neutral bits, and also AES with a larger key size comes with a slower key diffusion. These factors lead us to a higher number of attacked rounds and lower time complexities for AES-192 and AES-256, compared with the previous attacks against AES-128 in [Sas11, WFW+12]. The details of attacks, including the number of attacked rounds and time/memory complexities, compared with the previous works, are summarized in Table 1.

Organization. The rest of the paper is organized as follows. Section 2 gives the preliminaries of AES and the PGV hashing modes, followed by a general description of the MITM preimage attack in Section 3. The main techniques of MITM attack on AES hashing modes are summarized in Section 4, including those used in previous attacks and in our attacks. Results of 7 and 8 rounds are given in Section 5 and 6, respectively. The applicability of the presented pseudo-preimage attacks on the PGV modes and on conversion into full (second-)preimage attacks are discussed in Section 7. Section 8 concludes the paper.

Table 1: Summary of our improved pseudo-preimage attacks against the round-reduced compression function of AES hashing modes, compared with previous works

Target	#Rounds	Time-1	Time-2	Memory	(d_1, d_2, m)	Source
AES-128	7	2^{120}	2^{125} *	2^8	$(8, 8, 32)$	[Sas11]
	7	$2^{120-\min(t,24)}$	2^{123} *	$2^{8+\min(t,24)}$	$(8, 32, 32)$	[WFW+12]
	7	$\mathbf{2^{104}}$	$\mathbf{2^{117}}$	$\mathbf{2^{24}}$	$\mathbf{(24, 32, 24)}$	**Section 5.1**
AES-192	7	2^{120}	2^{125} *	2^8	$(8, 8, 32)$	[Sas11]
	7	$\mathbf{2^{96}}$	$\mathbf{2^{113}}$	$\mathbf{2^{32}}$	$\mathbf{(32, 32, 32)}$	**Section 5.2**
	8	$\mathbf{2^{112-\min(t,16)}}$	$\mathbf{2^{116}}$	$\mathbf{2^{16+\min(t,16)}}$	$\mathbf{(16, 32, 32)}$	**Section 6.3**
AES-256	7	2^{120}	2^{125} *	2^8	$(8, 8, 32)$	[Sas11]
	8	$\mathbf{2^{96}}$	$\mathbf{2^{113}}$	$\mathbf{2^{32}}$	$\mathbf{(32, 32, 32)}$	**Section 6.2**
Kiasu-BC	**7**	$\mathbf{2^{104-\min(t,8)}}$	$\mathbf{2^{117}}$	$\mathbf{2^{24+\min(t,8)}}$	$\mathbf{(24, 32, 32)}$	**Section 5.3**
	8	$\mathbf{2^{120-\min(t,24)}}$	$\mathbf{2^{123}}$	$\mathbf{2^{8+\min(t,24)}}$	$\mathbf{(8, 32, 32)}$	**Section 6.4**

– Time-1 is the time complexity of pseudo-preimage. Here, 2^t is the number of available targets for multi-target pseudo-preimage attacks. In the full (second-)preimage attacks, that can be the number of available nodes in the unbalanced-tree-based conversion from multi-target pseudo-preimage to preimage attacks (refer to Sect. 3.2), or the number of blocks of given message for second-preimage attacks.

– Time-2 is the complexity of using the (multi-target) pseudo-preimage attacks to do (second-)preimage attacks when requiring an upper layer of meet-in-the-middle procedure of conversion (refer to Sect. 3.2) for some PGV modes, such as DM-mode, and here a single target is given.

– * Note that for some PGV modes, such as MMO and MP modes, converting the pseudo-preimage attacks in [Sas11, WFW+12] to second-preimage attacks will not require the upper-layer meet-in-the-middle procedure, and the time complexities will be the same as that of the pseudo-preimage attacks (refer to [Sas11] for details). Whereas, the presented attacks in this paper always require an upper-layer meet-in-the-middle procedure to be converted into (second-)preimage attacks, because they require that both of the inputs to the encryption and to the key-schedule be unfixed.

2 Preliminaries

2.1 Description of AES

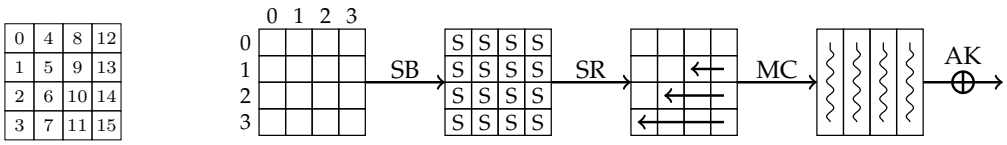
AES is an iterated block cipher that encrypts 128-bit plaintext with a secret key of sizes 128, 192, and 256 bits. AES with 128-bit (192, 256) master keys is denoted by AES-128 (192, 256). AES-128, AES-192, and AES-256 share the same round function with a different number of rounds: 10, 12, and 14, respectively. The rounds are numbered $0, \cdots, N_r - 1$, where $N_r \in \{10, 12, 14\}$ is the number of rounds. Its internal state can be represented as a 4×4 matrix whose elements are byte value (8 bits) in a finite field of GF(2^8). The round function consists of four basic transformations in the following order:

- `SubBytes` (SB) is a nonlinear substitution that applies the same S-box to each byte of the internal state.
- `ShiftRows` (SR) is a cyclic rotation of i-th row by i bytes to the left, for $i = 0, 1, 2, 3$.
- `MixColumns` (MC) is a multiplication of each column with a Maximum Distance Separable (MDS) matrix over GF(2^8).
- `AddRoundKey` (AK) is an exclusive-or with the round-dependent key.

MDS guarantees that the sum of active bytes (a.k.a. non-zero bytes) in the input and output of the MixColumns operation is at least 5 unless all bytes are non-active (a.k.a. zeros). The matrices for the encryption and decryption are shown below. Note that $X[j]$ is the input value and $Y[j]$ is the updated value. Numbers in typewriter font, `e`, `b`, `d`, and `9`, are in hexadecimal.

$$\begin{pmatrix} Y[0] \\ Y[1] \\ Y[2] \\ Y[3] \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{pmatrix}, \begin{pmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{pmatrix} = \begin{pmatrix} \texttt{e} & \texttt{b} & \texttt{d} & \texttt{9} \\ \texttt{9} & \texttt{e} & \texttt{b} & \texttt{d} \\ \texttt{d} & \texttt{9} & \texttt{e} & \texttt{b} \\ \texttt{b} & \texttt{d} & \texttt{9} & \texttt{e} \end{pmatrix} \begin{pmatrix} Y[0] \\ Y[1] \\ Y[2] \\ Y[3] \end{pmatrix}$$

One round of AES is depicted in Figure 1 as follows:



(a) Byte order **(b)** AES round function

Figure 1: AES byte order and AES round function

At the very beginning of the encryption, an additional whitening key addition is performed, and the last round does not contain `MixColumns`.

The key-schedule of AES transforms the master key into $N_r + 1$ subkeys of 128 bits each. The master key is divided into N_k 32-bit words $(W[0], W[1], ..., W[N_k - 1])$, then $W[i]$ for $i = N_k, \cdots, 4 \cdot N_r + 3$ is computed as

$$W[i] = \begin{cases} W[i - N_k] \oplus \texttt{SB}(\texttt{RotByte}(W[i-1])) \oplus Rcon[i/N_k], & i \equiv 0 \bmod N_k; \\ W[i-8] \oplus \texttt{SB}(W[i-1]), & N_k = 8 \text{ and } i \equiv 4 \bmod 8; \\ W[i - N_k] \oplus W[i-1], & \text{otherwise.} \end{cases}$$

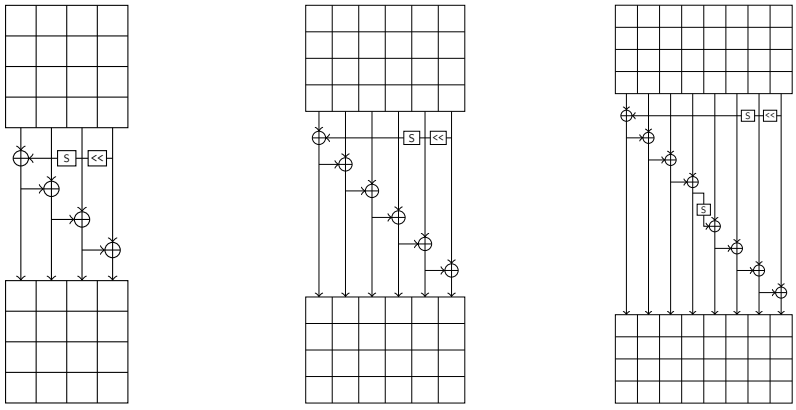


Figure 2: Round functions of key-schedules of AES-128, AES-192, and AES-256 [Jea16]

The i-th *round key* is the concatenation of 4 words $W[4i] \parallel W[4i+1] \parallel W[4i+2] \parallel W[4i+3]$. `RotByte` is a cyclic shift by one byte to the left, and *Rcon* is the round constant, for which we refer to [AES01] for details. The graphic representation of the key-schedules is depicted in Figure 2.

2.2 Description of Kiasu-BC

Kiasu-BC is the underlying tweakable block cipher (TBC) used in the authenticated encryption scheme Kiasu proposed by Jean *et al.* [JNP14a] alongside their TWEAKEY framework [JNP14b] at ASIACRYPT 2014. The TBC is almost identical to the AES-128 except for the additional input tweak, which renders it an attractive primitive for various modes of operation and applications requiring tweakable block ciphers. Therefore, studying how the additional tweak input affects the security strength compared to that of AES is highly valuable to gain trust for more adoptions.

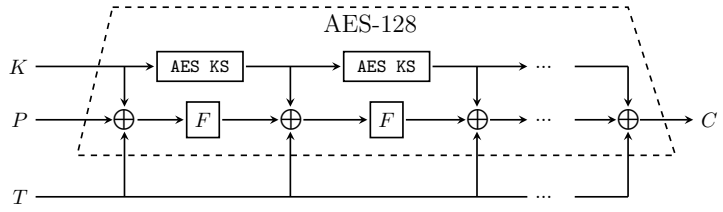


Figure 3: The Kiasu-BC tweakable block cipher based on AES-128

The TBC Kiasu-BC takes three inputs: a 64-bit tweak T, a 128-bit key K and a 128-bit plaintext P. It outputs a 128-bit ciphertext $C = E_K(T, P)$ as the encryption of P under the key K for the tweak value T. As depicted in Figure 3, Kiasu-BC is exactly the AES-128 cipher, but with a 64-bit tweak value XOR-ed to the first two rows of the internal state after each round key addition in the round function of the encryption, including after the pre-whitening key addition. There is no tweak schedule, *i.e.*, the same T is used every time in its original form. Kiasu-BC can actually be viewed as one of the simplest instances of the TWEAKEY framework based on AES.

3 The MITM Preimage Attack

The MITM Attack. In its early stages of development, the meet-in-the-middle approach proposed by Diffie and Hellman in [DH77] is mainly used as a generic time-memory trade-off technique to attack against encryption schemes with clear separations, *e.g.*, Double DES. That is because it is straightforward to divide the whole computation into two or multiple independent computational chunks. Thus, the whole 'for' loop in a brute force attack can be separated into two or multiple independent 'for' loops, which have a quite smaller and mutually balanced amount of computations, and which independently generate lists of candidates (of partial solutions). The independence between the smaller 'for' loops makes each element in one list be able to make a pair with any element in other lists to form a candidate solution. Such an effect of taking cartesian product between two sets enlarges the number of candidates of the correct computation dramatically. Then, according to the birthday paradox, say for two lists of 2^ℓ entries of n-bit values, to find a match with high probability, it is required $2^{(\ell+\ell)} \geq 2^n$, *i.e.*, $\ell \geq n/2$. Thus, the minimum time complexity of a simple MITM attack is $2^{n/2}$, together with $2^{n/2}$ memory.

3.1 Application to Pseudo-Preimage Attacks

Using the MITM approach in preimage finding on hash function can be seen in [AMM09, Leu08, IP07]. Aoki and Sasaki in [SA08] for the first time combined the MITM and local-collision approaches to devise preimage attacks on hash function HAVAL. Whereas, before that local-collision approach is mainly used in collision attacks on hash functions. Based

on these primary works, the MITM-based preimage attack on hash functions developed in a series of papers and advanced further.

3.1.1 Techniques Developed for MITM Preimage Attacks.

Techniques invented along the development of the attacks are as follows.

Splice-and-cut and neutral words. Aoki and Sasaki in [AS09b] invented the splice-and-cut MITM attack and proposed the concept of "neutral word." In the splice-and-cut MITM attack as depicted in Figure 4, the first and last steps of the attack target can be spliced to be consecutive steps by feed-forward mechanisms in the compression function of hashing modes (*e.g.*, DM-mode) or by querying the decryption in encryption schemes. The chain of computational steps of the attack target is cut starting from an internal step (named starting point), such that the chain is divided into two chunks of steps. Conventionally, the chunk of steps, which need to be computed forward (resp. backward) to reach the matching point, is named forward chunk (resp. backward chunk). The starting point is chosen so that each chunk includes at least one message (or key) word that is independent of the other chunk, where such message (or key) words are called "neutral words."

Initial-structure technique [SA09]: Initial-structure is a generalization of the local-collision technique that enables one to skip several steps at the beginning of chunks.

> "Initial structure is a few consecutive steps including at least two neutral words named m^{2nd} and m^{1st}, where steps after the initial structure (2nd chunk) can be computed independently of m^{1st} and steps before the initial structure (1st chunk) can be computed independently of m^{2nd} [SA09]."

Partial matching [AS09b, SA09]: In the primary MITM approach, the final phase of the attack involves matching between all-word in two states computed independently. Whereas, by executing only one-word (or several-words) matching instead of all-word matching, the required independent computations can be expanded by more steps, thus enables to attack more steps of the target.

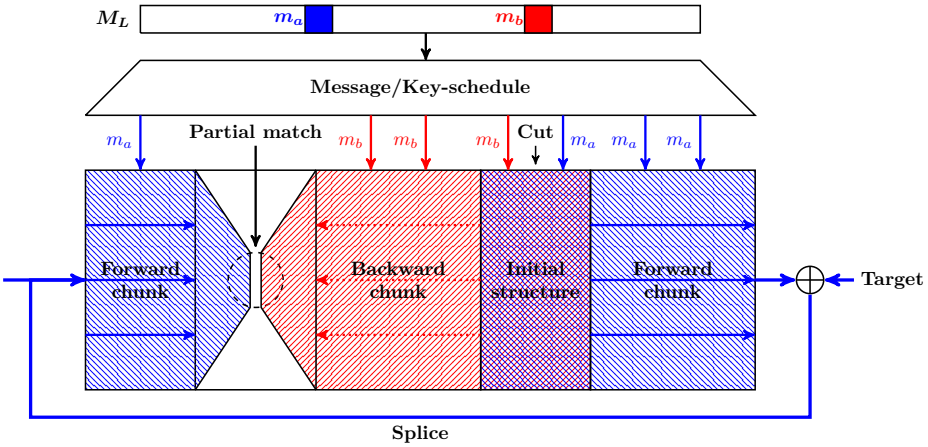
Partial-fixing [AS09b, SA09]: Fixing a part of the neutral words (*e.g.*, lower x-bits of a neutral word) enables one to partially compute more steps within a chunk even if in company with a neutral word for the other chunk.

Multi-targets [GLRW10a]: When incorporating multi-target scenarios into the MITM framework, the multiple available targets can directly provide additional freedoms to one computation chunk without influencing the other.

3.1.2 The Attack Framework.

As depicted in Figure 4, the attack framework of the splice-and-cut MITM attack using the initial structure and the partial match is as follows. Before the execution of the attack procedure, the configuration of the attack should be set up, which involves:

1. Chunk separation: by splicing and cutting, decide where the computation be the starting point of the forward/backward computation, and at which state, be the matching point. The principle of the chunk separation is to find the best balance between freedom degrees and the size of the matching point that the freedom degrees are fully used. That requires one to decide:



Let the space for both neutral words m_a and m_b be 2^ℓ, the time complexity is $2^{n-\ell}$, and memory complexity is 2^ℓ.

Figure 4: The advanced MITM pseudo-preimage attack on DM-mode [Sas11]

2. The neutral bytes for each chunk – the selection on the neutral bytes will determine the freedom degrees.

3. The bytes for matching – the derivation on the bytes for match also depends on the selection of neutral bytes and the computation rule of the attack target.

With the above configurations decided, the attack procedure goes as follows (Figure 4 illustrates the MITM pseudo-preimage attack integrating with these advanced techniques on Davies-Meyer mode): Denote the neutral words for the forward chunk and backward chunk by $N^{\mathtt{f}}$ and $N^{\mathtt{b}}$, respectively:

1. Fix all other words except for the neutral words $N^{\mathtt{f}}$ and $N^{\mathtt{b}}$ in the initial structure to arbitrary values.

2. For all possible values of $N^{\mathtt{f}}$, forward compute from the starting point to the matching point at the final state of the forward chunk to get a list $L^{\mathtt{f}}$ of candidate values indexed by the value of $N^{\mathtt{f}}$.

3. For all possible values of $N^{\mathtt{b}}$, backward compute from the starting point to the matching point at the final state of the backward chunk to get a list $L^{\mathtt{b}}$ of candidate values indexed by the value of $N^{\mathtt{b}}$.

4. Sorting the two lists $L^{\mathtt{f}}$ and $L^{\mathtt{b}}$ using hash tables, check whether there is a match/partial-match between them.

5. In case of partial-matching used in the above step, for the surviving pairs, check for a full match.

6. Repeat the whole procedure to find full state matches by changing the values of fixed words.

3.1.3 The Complexity Analysis.

Denote the size of the internal state by n, the freedom degrees in the forward and backward directions by d_1 and d_2 respectively, and the number of bits for the match by m.

1. Forward computing to get a list $L^{\mathtt{f}}$ of size 2^{d_1} requires 2^{d_1} computations of the forward chunk.

2. Backward computing to get a list $L^{\mathtt{b}}$ of size 2^{d_2} requires 2^{d_2} computations of the backward chunk.

3. Matching between $L^{\mathtt{f}}$ and $L^{\mathtt{b}}$ requires $2^{\max(d_1,d_2)}$ memory access which is usually ignored, compared with the $2^{\max(d_1,d_2)}$ computations of the compression function of the target in above steps.

4. $2^{d_1+d_2-m}$ pairs are left after partial matching, hence the same complexity is required for full-match checking.

5. Finding a full match requires $2^{m-(d_1+d_2)} \times 2^{n-m} = 2^{n-(d_1+d_2)}$ repetitions.

When d_1, d_2 are different, *i.e.*, unbalanced, we use $\max(2^{d_1}, 2^{d_2})$ to denote the sum of complexities for computing the two chunks. Note, when $d_1 = d_2$, the computation complexity is 2^{d_1} full target (= forward chunk + backward chunk). Hence, $\max(2^{d_1}, 2^{d_2})$ is used for all cases. The above complexity analysis gives

$$2^{n-(d_1+d_2)} \cdot (2^{\max(d_1,d_2)} + 2^{d_1+d_2-m}) = 2^{n-\min(d_1,d_2)} + 2^{n-m} \simeq 2^{n-\min(d_1,d_2,m)}. \quad (1)$$

From this formula, it can be seen that the critical point for the attack being optimized is to reach a balance between the freedom degrees for the forward chunk and the backward chunk. Because at last, it will only depend on the minimum between the two freedom degrees. Besides, the memory complexity can be $2^{\min(d_1,d_2)}$ by only storing the list computed by the direction with less freedom and make a match at once a candidate for the other direction being available. Accordingly, the larger the $\min(d_1, d_2)$ the lesser the time complexity and at the same time the larger the memory complexity. However, generally, in such MITM preimage attack on hash function, the $\min(d_1, d_2)$ is less than $n/2$, thus, compared with memory complexity, time complexity is the bottleneck. Therefore, generally, the larger the $\min(d_1, d_2)$ the better the attack. The second term 2^{n-m} is usually minor when $m > \min(d_1, d_2)$, *i.e.*, the number of matching bits is more than that of neutral bits.

3.2 Conversion from Pseudo-Preimages to Preimages

Note that, the above MITM attack only gives pseudo-preimage. Below we consider the methods to convert pseudo-preimages to preimages.

Converting pseudo-preimages to a preimage [MvV97, Fact9.99]: For n-bit narrow-pipe iterated hash function, using the unbalanced meet-in-the-middle approach, a pseudo-preimage attack with a computational complexity of 2^{ℓ} ($\ell < n - 2$) can be converted into a preimage attack with computational complexity of $2^{(n+\ell)/2+1}$. That is done by first finding $2^{(n-\ell)/2}$ pseudo-preimages (mapping $2^{(n-\ell)/2}$ random starting chaining values to the given target T), and then starting from the real initialization value vector IV, using $2^{(n+\ell)/2+1}$ random message blocks to get $2^{(n+\ell)/2+1}$ random chaining values. Among the $2^{(n+\ell)/2+1}$ chaining values mapped from the real IV and the $2^{(n-\ell)/2}$ starting chaining values mapped by the pseudo-preimages to the given target T, we can expect to find one match due to the birthday paradox, which leads a valid preimage. Figure 5a visualizes the above generic meet-in-the-middle procedure.

An improvement over the above meet-in-the-middle procedure of converting pseudo-preimage attacks to preimage attacks is made possible by Leurent in [Leu08], where he proposed to use multi-target pseudo-preimage attacks which form an unbalanced-tree, and require the assistance of the expandable message technique to overcome the length padding (see Figure 5b). The main idea is as follows. Like in the above meet-in-the-middle

procedure, one has to first launch many procedures of pseudo-preimage attack. But instead of using the starting chaining values of the pseudo-preimages attacks until the matching phase, those starting chaining values determined by previous procedures of pseudo-preimage attack are immediately used as targets for later procedures of pseudo-preimage attack. Suppose we have already launched i procedures of pseudo-preimage attack, there are then i targets for the $(i+1)$-th procedure of pseudo-preimage attack. Given i targets, the complexity of a pseudo-preimage attack is possible to be reduced from 2^ℓ to $2^\ell/i$.[2] As a result, to obtain 2^k starting chaining values in total by launching 2^k procedures of pseudo-preimage attack, it takes $2^\ell/1 + 2^\ell/2 + 2^\ell/3 + \cdots + 2^\ell/2^k \simeq k\ln(2) \cdot 2^\ell$. Then, by generating 2^{n-k} chaining values mapped by the expandable messages from the real IV, one match is expected, which leads to a real preimage. Taking the optimal $k = n - \ell$, the overall time complexity of this improved method is

$$((n-\ell)\cdot\ln(2)+1)\cdot 2^\ell = (\min(d_1,d_2)\cdot\ln(2)+1)\cdot 2^{n-\min(d_1,d_2)}, \tag{2}$$

where a pseudo-preimage attack costs 2^ℓ computations given a single target ($\ell = n - \min(d_1, d_2)$ in our notation), and $2^{\ell-t}$ computations given 2^t targets.

The assumption that 2^t-target pseudo-preimage attack costs $2^{\ell-t}$ computations is possible in our attacks when the target can be used as the additional source of neutral words. Without loss of generality, let us assume targets can be used as part of the forward chunk, hence the size of $L^{\mathtt{f}}$ list is increased from 2^{d_1} to 2^{d_1+t}, then Eq. (1) gives complexities

$$\text{Time: } 2^{n-\min(d_1+t,d_2,m)};\ \text{Memory: } 2^{\min(d_1+t,d_2,m)} \tag{3}$$

In other words, the complexity of pseudo-preimage attack reduces linearly with respect to the number of targets available, as long as $d_1 + t < d_2$ and $m > \min(d_1 + t, d_2)$.

When these conditions could not be met, the complexity of pseudo-preimage attack remains as $2^{n-\min(d_1,d_2)} + 2^{n-m} \simeq 2^{n-\min(d_1,d_2,m)}$ for $d_1 + t \geq d_2$ (*i.e.*, $t \geq d_2 - d_1$), for which the time complexity of preimage conversion will follow the original unbalanced MITM

$$2^{(n+\ell)/2+1} \simeq 2^{n+1-\min(d_1,d_2,m)/2}. \tag{4}$$

(a) Traditional Conversion

(b) Multi-Target Pseudo-Preimage

Figure 5: Converting Pseudo-Preimages to Preimages: circle denotes state, arrow denotes message block [GLRW10b]

3.3 Converting Block cipher to Compression Function

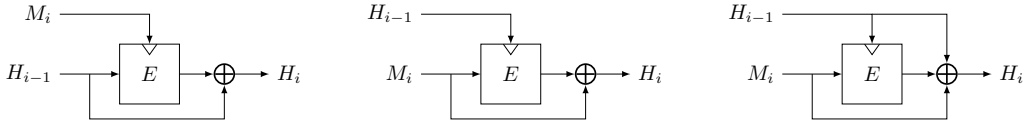
In [PGV94], Preneel *et al.* summarized 17 mode-of-operations to build a compression function for hash from a block cipher. Twelve of them are shown to be secure. Denote the compression function composed of a hash function by CF and the block cipher E with a key K by E_K. The twelve secure PGV constructions of CF can be expressed by formulas

[2] In our attacks, this is possible when there is a gap i between the freedom provided by neutral words of the forward computation and that of the backward computation, which can be filled by exploiting freedom provided by the multiple targets, as will be shown later.

Table 2: Twelve secure PGV constructions [PGV94, Sas11].

	No.	Computation	No.	Computation	No.	Computation	No.	Computation
Class 1	1	$E_{H_{i-1}}(M_i) \oplus M_i$	2	$E_{H_{i-1}}(X_i) \oplus X_i$	3	$E_{H_{i-1}}(M_i) \oplus X_i$	4	$E_{H_{i-1}}(X_i) \oplus M_i$
Class 2	5	$E_{M_i}(H_{i-1}) \oplus H_{i-1}$	6	$E_{M_i}(X_i) \oplus X_i$	7	$E_{M_i}(H_{i-1}) \oplus X_i$	8	$E_{M_i}(X_i) \oplus H_{i-1}$
Class 3	9	$E_{X_i}(M_i) \oplus M_i$	10	$E_{X_i}(H_{i-1}) \oplus H_{i-1}$	11	$E_{X_i}(M_i) \oplus H_{i-1}$	12	$E_{X_i}(H_{i-1}) \oplus M_i$

X_i represents $H_{i-1} \oplus M_i$.



(a) DM-mode (PGV No.5) $H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$

(b) MMO-mode (PGV No.1) $H_i = E_{H_{i-1}}(M_i) \oplus M_i$

(c) MP-mode (PGV No.3) $H_i = E_{H_{i-1}}(M_i) \oplus M_i \oplus H_{i-1}$

Figure 6: Illustrations for DM, MMO, and MP modes [Sas11, Jea16]

in Table 2, where H_{i-1} and H_i denote the chaining states before and after the update, M_i the message block, and X_i the XOR of H_{i-1} and M_i. These twelve PGV modes can be classified according to the material fed in through the key into three classes: Class 1 – chaining values are fed in through the key (row 1 in Table 2); Class 2 – messages are fed in through the key (row 2 in Table 2); Class 3 – XOR sum of message and the chaining values are fed in through the key (row 3 in Table 2). Three of these PGV modes, known as DM-mode, MMO-mode, and MP-mode, are used in practice (see Figure 6).

In the original pseudo-preimage attack on AES hashing mode by Sasaki [Sas11], the key is preset to random constants, which can be used by attacker, and the input state is determined by the attack, hence such pseudo-preimage can be converted into preimages for modes in Class 1 such as MMO and MP with the same complexity as pseudo-preimage. Other modes in Class 2 and 3 require conversion, and hence result in higher complexities for preimages as discussed in the subsection above.

Converting TBC to Compression Function. The tweakable block cipher, denoted here as $\widetilde{\mathsf{E}}(K_t, T_t, P)$, has an additional input tweak T, compared with block ciphers $\mathsf{E}(K, P)$. We consider here converting a TBC to a block cipher, then to a compression function through the modes above. The TBC-to-BC conversion can be divided into 3 types:

- Type-I: $\mathsf{E}(K, P) = \widetilde{\mathsf{E}}(K_t = K, T_t = C, P)$, where C is a constant.
- Type-II: $\mathsf{E}(K, P) = \widetilde{\mathsf{E}}(K_t = C, T_t = K, P)$, where C is a constant.
- Type-III: $\mathsf{E}(K, P) = \widetilde{\mathsf{E}}(K_t = K_1, T_t = K_2, P)$, where $K = K_1 || K_2$, *i.e.*, both key and tweak of $\widetilde{\mathsf{E}}$ are used as the key of the block cipher.

From the cryptanalyst's point of view, Type-III gives additional input to the attacker; hence it is likely more rounds could be attacked for Type-III, compared with the other two types.

4 Techniques of MITM Attack on AES Hashing Modes

4.1 The 7-Round Attack by Sasaki and Its Improvement by Wu *et al.*

Following the framework of splice-and-cut MITM attack depicted in Figure 4, and based on an important observation on the slow diffusion of a 4-round AES with the omission of

MixColumns in the second round, Sasaki [Sas11] invented a MITM preimage attack on AES hashing mode. In [Sas11], a basic splice-and-cut MITM attack on 4-round AES (with the omission of MixColumns in the last round as in the full version of AES) hashing mode was first proposed, which was then extended to 7-round using two main techniques that will be introduced next. In the basic 4-round attack, the computation flow is spliced using the feed-forward mechanism in the hashing mode, and it is cut exactly in the middle (at the beginning of the third round). In doing this, the round missing MixColumns becomes the second round viewing from the starting point at where the computation was cut. Due to the slow diffusion observed, importing two neutral bytes at the starting point, the influence of a neutral byte penetrating forwardly and undisturbedly through 3-round (across the sliced point) and the influence of a neutral byte penetrating backwardly and undisturbedly through 1-round can be matched at the end of the first round.

Different from previous MITM preimage attacks on hash functions such as MD5 and Tiger, neutral words in the attack on AES hashing mode in [Sas11] are not chosen from the message. Instead, they are from the internal state values of the compression function. This choice is mainly due to the fast diffusion of the AES key-schedule, *i.e.*, the key-schedule quickly diffuses any choice of key byte (as the neutral word) to all other key bytes in just a few rounds. As a result, in [Sas11] the key-input/message-input is fixed to arbitrary constant values. Although the number of rounds of AES is much smaller than that of many dedicated hash functions such as MD5 and SHA-1, the round function is relatively heavier and the computation has faster diffusion, which prevents the MITM attacks from penetrating many rounds. Thus, more techniques were invented in [Sas11] for attacking more rounds (up to 7 rounds, yet the results are still not for the full rounds, and currently leaves comfortable margins).

There are two main techniques used for attacking more rounds in the MITM pseudo-preimage attack in [Sas11]. In a nutshell, the first is a carefully crafted initial structure, and the second is matching through indirect but efficiently computable MixColumns relations.

Initial Structure (IS). Concretely, when constructing the initial structure, by restricting the values of neutral bytes to a special set, the initial structure can extend the number of rounds in each chunk by one round. These special values of the neutral bytes are chosen in such a way that their impacts on particular output bytes of the MixColumns (or InverseMixColumns) are known constants, *i.e.*, have no impact. This reduces the number of unknown bytes after (resp. before) the first MixColumns (resp. InverseMixColumns) in each chunk, which is the starting point of each chunk. For example, when 4 bytes $B^{\mathtt{b}}[0,1,2,3]$, *i.e.*, the bytes in the first column of the state $B^{\mathtt{b}}$, are chosen as the neutral bytes for the forward chunk, there are 2^{32} possible values. Among them, we can choose a small subset of size 2^8 in the following way: first, fix an arbitrary constant $C^{\mathtt{neut}}$ of 3 bytes, then, impose the restriction of $B^{\mathtt{f}} = \mathrm{MC}^{-1}(B^{\mathtt{b}}[0,1,2,3])$ [3]and $B^{\mathtt{f}}[1,2,3] = C^{\mathtt{neut}}$. Since $C^{\mathtt{neut}}$ is of 24 bits and $B^{\mathtt{b}}[0,1,2,3]$ is of 32 bits , there will be 2^8 solutions for $B^{\mathtt{b}}[0,1,2,3]$, which can be obtained by enumerating all possible values of $B^{\mathtt{b}}[0]$ and for each computing the unique value of the other 3 bytes according to the restriction imposed through $C^{\mathtt{neut}}$.

Matching Through MixColumns (MTM). In the second technique, when matching at the meeting point, properties of the MixColumns are used again: instead of directly matching values of the same bytes in a state, matching via deterministic relations between bytes in different states (specifically, the states immediately before and after the MixColumns). This can extend the attack by one more round. Concretely, the AES MixColumns has the following feature: knowledge of any 4 out of the 8 input/output bytes to the MC will determine all other bytes, and knowledge of any additional byte(s) (*i.e.*, more than

[3]In the sequel, we informally use $\mathrm{MC}(X)$ (resp. $\mathrm{MC}^{-1}(X)$) to represent the multiplication of a column X with the matrix (resp. its inverse) used in MixColumns.

4 bytes) can be used as an $8 \cdot (x-4)$-bit filter when a total of x bytes are known for $4 < x \leq 8$. With an example below, we demonstrate how this can be done in the way of meet-in-the-middle.

To efficiently check whether the paired values in the two lists match through the `MixColumns` operation, one can test the consistency between the bytes column-wise, individually for each column in sequence. Suppose the two states before and after the `MixColumns` are $B^{\mathtt{f}}$ and $B^{\mathtt{b}}$, of which some bytes have been obtained independently via the forward and backward chunks, and the candidate values have been stored in two lists, *e.g.*, $B^{\mathtt{f}}[0,2]$ and $B^{\mathtt{b}}[1,2,3]$ in the first column. These bytes are related through MC as follows:

$$B^{\mathtt{f}}[0] = (\mathtt{e},\mathtt{b},\mathtt{d},\mathtt{9}) \cdot (B^{\mathtt{b}}[0], B^{\mathtt{b}}[1], B^{\mathtt{b}}[2], B^{\mathtt{b}}[3])^T,$$
$$B^{\mathtt{f}}[2] = (\mathtt{d},\mathtt{9},\mathtt{e},\mathtt{b}) \cdot (B^{\mathtt{b}}[0], B^{\mathtt{b}}[1], B^{\mathtt{b}}[2], B^{\mathtt{b}}[3])^T.$$

Let us denote by $g'(B^{\mathtt{b}}[1,2,3]) = (\mathtt{b},\mathtt{d},\mathtt{9}) \cdot (B^{\mathtt{b}}[1], B^{\mathtt{b}}[2], B^{\mathtt{b}}[3])^T$ a linear mapping from 3 bytes to 1 byte, and by $g''(B^{\mathtt{b}}[1,2,3]) = (\mathtt{9},\mathtt{e},\mathtt{b}) \cdot (B^{\mathtt{b}}[1], B^{\mathtt{b}}[2], B^{\mathtt{b}}[3])^T$ another linear mapping. Then

$$B^{\mathtt{f}}[0] = \mathtt{e} \cdot B^{\mathtt{b}}[0] \oplus g'(B^{\mathtt{b}}[1,2,3]) \quad \text{and} \quad B^{\mathtt{f}}[2] = \mathtt{d} \cdot B^{\mathtt{b}}[0] \oplus g''(B^{\mathtt{b}}[1,2,3]).$$

Canceling out $B^{\mathtt{b}}[0]$ implies

$$\mathtt{d} \cdot B^{\mathtt{f}}[0] \oplus \mathtt{e} \cdot B^{\mathtt{f}}[2] = \mathtt{d} \cdot g'(B^{\mathtt{b}}[1,2,3]) \oplus \mathtt{e} \cdot g''(B^{\mathtt{b}}[1,2,3]).$$

For the sake of simplicity, denote by $M^{\mathtt{f}} = f(B^{\mathtt{f}}[0,2]) = \mathtt{d} \cdot B^{\mathtt{f}}[0] \oplus \mathtt{e} \cdot B^{\mathtt{f}}[2]$ a linear mapping from 2 bytes to 1 byte, and by $M^{\mathtt{b}} = g(B^{\mathtt{b}}[1,2,3]) = \mathtt{d} \cdot g'(B^{\mathtt{b}}[1,2,3]) \oplus \mathtt{e} \cdot g''(B^{\mathtt{b}}[1,2,3])$ another linear mapping from 3 bytes to 1 byte. Then find matches with $M^{\mathtt{f}} = M^{\mathtt{b}}$, which can be computed independently and stored in hash tables, then matched in the way of MITM. To be more general,

Property 1. When x out of the 8 input and output bytes of the `MixColumns` are known, there is a filter of $(x-4)$ bytes ($= 8 \cdot (x-4)$ bits), and such filtering can be done in the way of meet-in-the-middle.

This property is used in the matching of all attacks to be given in the rest of the paper, and for the sake of simplicity, the explicit expressions of $M^{\mathtt{f}}$ and $M^{\mathtt{b}}$, *i.e.*, the derived matching functions f and g from the input $B^{\mathtt{f}}$ and $B^{\mathtt{b}}$, are omitted. The total bits of filtering m is then the sum of 4 independent columns.

Attacks to be given in the rest of the paper inherit both those techniques and the splice-and-cut framework of the attack in [Sas11].

Exploiting the Freedom Lying in Multiple Targets (MT). In [WFW+12], authors observed that there is actually quite a great amount of additional degree of freedom for backward chunk in the attack in [Sas11], which are not utilized (set as constant). That is because, the bottleneck for the attack in [Sas11] lies in the lack of freedom for forward chunk. That is, for the forward chunk, the degree of freedom is much less than that of the backward chunk (*i.e.*, $d_1 \lll d_2$), which determines the time complexity. However, authors in [WFW+12] observed that in the setting of multi-target attack, the freedom that lies in the target can be integrated into the forward chunk and thus increase the degree of freedom for the forward chunk. In doing that, those additional neutral bytes for the backward chunk can be utilized more fully. As a result, the attack in [Sas11] was improved to be a more efficient multi-target pseudo-preimage attack, and the complexity of the overall preimage attack can be reduced.

4.2 Introducing Neutral Bytes in Key

Sasaki in [Sas11] has already discussed the possibility to improve the attack described in Sect. 4.1 in chosen-key setting, *i.e.*, introduce neutral bytes from key values. However, the conclusion is negative in consideration of the difficulties in adopting the splice-and-cut technique in the key-schedule function.

However, in this paper, we show that neutral bytes can be introduced from the key state to improve the computational complexity or increase the number of rounds that can be attacked. In our attacks, neutral bytes in the key are used as part of the forward chunk only, *i.e.*, none is used for the backward chunk. This enables us to avoid separating the key bytes in every round key generated by the key-schedule function, and in the meantime, to exploit freedom from the key to balance the computations of forward and backward.

Explicitly, we introduce neutral bytes in the key state in addition to the original neutral bytes in the encryption state. Those neutral bytes are all for the forward chunk, while the effect of neutral bytes in the encryption state and those in the key state are different: neutral bytes in the encryption state affect the candidate values at the matching point through the AES round functions in each direction, and neutral bytes in the key state affect through all subsequent round keys (via key-schedule). Thus, although they both affect the same bytes at the matching point, they both provide possible candidate values for the matching bytes. Moreover, because the independent constructions between the AES round function and the AES key-schedule, even the values that are chosen for the neutral bytes in encryption state and that in key state are within the same algebraic structure, *e.g.*, the same linear subspace, their effect on the matching values can be seen as independent.

At a high level, introducing neutral bytes in key state can play roles in various ways:

1. XORing neutral bytes in key state to neutral bytes in encryption state without any restriction on the values. In doing that, the lacking degree of freedom can be increased, and the amount of computation cost by the forward chuck and that cost by the backward chunk can be balanced, resulting in an improvement in the overall complexity. Abstractly, we represent this case as $\mathbf{A} \oplus \mathbf{A} = \mathbf{A}$, where $\mathbf{A}$ (the abbr. of 'All') represents an attribute of being fully free (can take any value) of neutral bytes.

2. XORing neutral bytes in key state to neutral bytes in encryption state with a restriction that the XOR sum is constant. In doing that, the impact caused by neutral bytes in one computation chunk on the other computation chunk can be canceled, providing an opportunity to extend the attack to more rounds. Abstractly, we represent this case as $\mathbf{A} \oplus \bar{\mathbf{A}} = \mathbf{C}$, where $\bar{\mathbf{A}}$ represents an attribute of being restricted by the value of other fully free neutral bytes, and $\mathbf{C}$ represents an attribute of being constant.

3. Combining neutral bytes in key state with neutral bytes in encryption state by linear equations (related with the linear diffusion layer of the cipher) such that their XOR sum can take any value, but after being operated by the linear layer of the cipher, part of the result is constant. In doing that, the impact caused by neutral bytes in one computation chunk on the other computation chunk can be partially canceled, at the same time additional freedom is imported, providing an opportunity to improve the overall complexity and/or extend the attack to more rounds. Abstractly, we represent this case as $\mathcal{L}(\mathbf{A}, \mathbf{A}) = \mathbf{A} \| \mathbf{C}$, where $\mathcal{L}(\mathbf{A}, \mathbf{A})$ represents an attribute of several fully free neutral bytes being restricted by linear equations.

In the next two sections, we first present our 7-round pseudo-preimage attacks on AES-128, AES-192, and Kiasu-BC hashing mode improved in terms of the attack complexity, and then present our 8-round attacks on AES-192, AES-256, and Kiasu-BC hashing mode improved in terms of the number of attacked rounds, which are all enabled by introducing neutral bytes in the key.

5 Reducing the Complexities of 7-Round Attacks

5.1 Improved Attack on 7-Round AES-128 Hashing Mode

As mentioned above, in [WFW+12], authors identified a great amount of freedom that can be imported by activating bytes that were set as constant previously in [Sas11] as neutral bytes for the backward chunk. The problem is then the lack of freedom for the forward chunk. In [WFW+12], the authors exploited the freedom that lies in the target in the setting where multiple targets are available. In our attack, we import freedom for the forward chunk by activating particular bytes that were set as constant previously in [Sas11] both in the encryption state and in the key state as neutral bytes. The introduced neutral bytes from the key can cancel out the diffused impact caused by (inverse) `MixColumns` operated on a neutral byte in the forward chunk, at the same time provide an additional degree of freedom.

Details of our improved pseudo-preimage attack are as follows (illustrated in Figure 7):

Attack 1: Improved pseudo-preimage attack on 7-round AES-128 hashing mode

Attack configuration

1. Initial structure: from $\#12$ – $\#19$ including k_3 – k_4

   - Neutral bytes for forward:
     - In encryption state: $\#12[0,5]$
     - In key state: $k_3[4,5,6,7]$, such that

$$\begin{cases} \mathrm{MC}^{-1}((0,\#12[5],0,0) \oplus k_3[4,5,6,7])[1,2,3] = C_1^{\mathtt{neut}} \\ k_3[0,1,2,3] \oplus k_3[4,5,6,7] = C_1^{\mathtt{key}} \end{cases}$$

     In the first equation, $C_1^{\mathtt{neut}}$ is a 3-byte constant, $\#12[5], k_3[4], k_3[5], k_3[6], k_3[7]$ are 1-byte variables. That is, the first equation essentially consists of three linear equations on five variables, *i.e.*,

$$\begin{bmatrix} \mathtt{9} & \mathtt{e} & \mathtt{b} & \mathtt{d} & \mathtt{e} \\ \mathtt{d} & \mathtt{9} & \mathtt{e} & \mathtt{b} & \mathtt{9} \\ \mathtt{b} & \mathtt{d} & \mathtt{9} & \mathtt{e} & \mathtt{d} \end{bmatrix} \times \begin{bmatrix} k_3[4] \\ k_3[5] \\ k_3[6] \\ k_3[7] \\ \#12[5] \end{bmatrix} = \begin{bmatrix} C_{1,0}^{\mathtt{neut}} \\ C_{1,1}^{\mathtt{neut}} \\ C_{1,2}^{\mathtt{neut}} \end{bmatrix}.$$

     Because the matrix used in AES `MixColumns` is MDS, the rows are pairwise independent. The rank of the above coefficient matrix is three. Thus, the number of solutions for these equations is $2^{(5-3)\times 8} = 2^{16}$. By solving the linear equations, one can efficiently obtain all the values of the neutral bytes fulfilling the constraints[a]. In short, the first equation brings 2 bytes degree of freedom for the forward computation, at the same time makes the impact of those neutral bytes on the backward computation of $\#11[5,6,7]$ be predefined constant $C_1^{\mathtt{neut}}$.

     Fix all other bytes in k_3, *i.e.*, $k_3[8..15]$ as constant. By the second equation and according to the key-schedule of AES-128, $k_4[0,1,2,3] \oplus k_3[0,1,2,3]$ and all other bytes in k_4 are constant. Denotes all constant materials in the key state as $C^{\mathtt{key}}$.

As a result, there is 3-byte degree of freedom brought by neutral bytes for the forward chunk, *i.e.*, $d_1 = 24$.

- Neutral bytes for backward:
  - In encryption state: $\#19[1,2,3;4,5,6;8,9,11;12,14,15]$, such that

$$\begin{cases} \mathrm{MC}((\quad 0, & \#19[1], & \#19[2], & \#19[3])) & = C_{2,0}^{\mathtt{neut}} \\ \mathrm{MC}((\quad \#19[4], & \#19[5], & \#19[6], & 0)) & = C_{2,1}^{\mathtt{neut}} \\ \mathrm{MC}((\quad \#19[8], & \#19[9], & 0, & \#19[11])) & = C_{2,2}^{\mathtt{neut}} \\ \mathrm{MC}((\quad \#19[12], & 0, & \#19[14], & \#19[15])) & = C_{2,3}^{\mathtt{neut}} \end{cases}$$

    That is, their impact on the forward computation of $\#20[0,2;5,7;8,10;13,15]$ equal to predefined constants $C_2^{\mathtt{neut}} = C_{2,0}^{\mathtt{neut}} \| C_{2,1}^{\mathtt{neut}} \| C_{2,2}^{\mathtt{neut}} \| C_{2,3}^{\mathtt{neut}}$, where $C_{2,i}^{\mathtt{neut}}$ for $0 \le i \le 3$ are 2-byte constants.

  As a result, there is 4-byte degree of freedom brought by neutral bytes for backward computation, *i.e.*, $d_2 = 32$.

2. Chunk separation:
   - Forward chunk: the computation following #20 – #28 – #0 – #7
   - Backward chunk: the computation following #11 – #8
3. Bytes for match: denote the equivalent sub-key of k_2 by uk_2, which can be computed via $\mathrm{MC}^{-1}(k_2)$:
   - $B^{\mathtt{f}} = (\#7 \oplus uk_2)[0,2;8,10;13,15]$, note the second column is not utilized.
   - $B^{\mathtt{b}} = \#8[1,2,3;8,9,10;12,13,14]$, note the second column is not utilized.

   As a result, there are equivalently 3 bytes for the match, *i.e.*, $m = 24$.

Attack procedure.

1. Fix constants:
   - $C^{\mathtt{key}}$: constant materials in the key state, including $C_1^{\mathtt{key}}$
   - $C_0^{\mathtt{neut}}$: 2 bytes values for $\#12[10,15]$
   - $C_1^{\mathtt{neut}}$: 3 bytes values for impacts from $\#12[5]$ and $k_3[4,5,6,7]$ on $\#11[5,6,7]$.
   - $C_2^{\mathtt{neut}}$: 8 bytes values for impacts from $\#19[1,2,3;4,5,6;8,9,11;12,14,15]$ on $\#20[0,2;5,7;8,10;13,15]$
2. Forward computation: for each of the $2^{d_1=24}$ values of the neutral bytes for forward – $\#12[0,5]$ and $k_3[4,5,6,7]$ s.t. their impact on $\#11[5,6,7] = C_1^{\mathtt{neut}}$:
   - Compute all required sub-key bytes from $k_3[4,5,6,7]$ and $C^{\mathtt{key}}$, note that $k_3[0,1,2,3] \oplus k_3[4,5,6,7]$, $k_4[0,1,2,3] \oplus k_3[4,5,6,7]$, and all other bytes in k_3 and k_4 are constants.
   - Compute in forward from #12 – #28 (blue cells in Figure 7)
     - xor state #28 with T, and compute #0 – #7 (blue cells in Figure 7)
     - compute $M^{\mathtt{f}}$ from $B^{\mathtt{f}}$, store results in $L^{\mathtt{f}}$

3. Backward computation: for each of the $2^{d_2=32}$ values of the neutral bytes for backward – $\#19[1,2,3;4,5,6;8,9,11;12,14,15]$ s.t. their impact on bytes $\#20[0,2;5,7;8,10;13,15] = C_2^{\texttt{neut}}$:
   - Compute in backward from #19 – #8 (red cells in Figure 7)
   - Compute M^{b} from B^{b}, store the results in L^{b}
4. Matching: find matches between L^{f} and L^{b}, for each such match of partial state, all bytes at the initial structure are fixed, and hence the full states of #7 and #8 can be tested for full-state matching. Output (H_{N-1}, M_N) if a full match is found, otherwise, go back to step 1 with some other values for the fixed constants at initial structure and repeat.

[a]Please refer to the example codes for obtaining the values of the neutral bytes via https://www.dropbox.com/sh/c5dm28821f2jmfl/AAC3iGXUtA6crWaLwdS9LC2Ca?dl=0.

Attack complexity. By directly applying the complexity analysis Eq. (1) and plugging $d_1 = 24$, $d_2 = 32$, and $m = 24$, we can immediately conclude that the time complexity of this attack is $2^{128-\min(24,32,24)} = 2^{104}$ and the memory complexity is 2^{24}.

In summary, in this attack, we introduce neutral bytes from the key in addition to two neutral bytes in the encryption state, and make them play a role in the third way, *i.e.*, $\mathcal{L}(\mathbf{A}, \mathbf{A}) = \mathbf{A}\|\mathbf{C}$. In doing that, the degree of freedom for the forward chunk is increased with a limited impact on the backward chunk. Note that the limited impact is the loss of 8 bits for the match. Nevertheless, the computations of the forward chunk and that of the backward chunk are more balanced. As a result, the computational complexity can be reduced from 2^{120} to 2^{104} given a single target compared with the attack in [Sas11], and changed from $2^{120-\min(t,24)}$ to 2^{104} given 2^t targets compared with the attack in [WFW+12].

5.2 Application to 7-Round AES-192 Hashing Mode

Unlike the 7-round attacks in [Sas11, WFW+12], the attack presented in above Sect. 5.1 is not quite general due to its dependence on key-schedule algorithms. Thus, it cannot be applied without modification for AES-192/256. However, the idea of introducing neutral bytes in the key can nevertheless be applied.

Compared with AES-128, the key-schedule of AES-192 has relatively slow diffusion. Consequently, it is possible to select more neutral bytes from the key to completely cancel out the impact caused by neutral bytes in the forward chunk on the backward chunk. Concretely, for AES-192, according to the key-schedule, the full key state k_3 can be fixed without dependence on neutral bytes in the first column of k_4 (as depicted in Figure 8b). Thus, the propagation of the influence brought by the neutral bytes in k_4 will not impact the backward computation which is relatively short.

An attack follows the same framework as previous attacks but with improved complexity can then be devised. The main different part is depicted in Figure 8. As can be seen in Figure 8, for forward chunk, when introducing neutral bytes $\#17[0,1,2,3]$ in encryption state and $k_4[0,1,2,3]$ in key state and restricting their XOR sum to be constant, their impact on the backward chunk is completely canceled; and at the same time, both the degree of freedom for the forward chunk and that for the backward chunk reach an amount of 32, *i.e.*, $d_1 = d_2 = 32$. The number of bits for match becomes even more, *i.e.*, $m = 64$. Applying Eq. (1) to the complexity analysis, we conclude that, for this attack on 7-round AES-192 hashing mode, the total computational complexity is $2^{128-\min(32,32,64)} = 2^{96}$ computations of 7-round AES. The memory complexity is 2^{32}.

Note that compared with the key-schedule of AES-192, the diffusion of the key-schedule

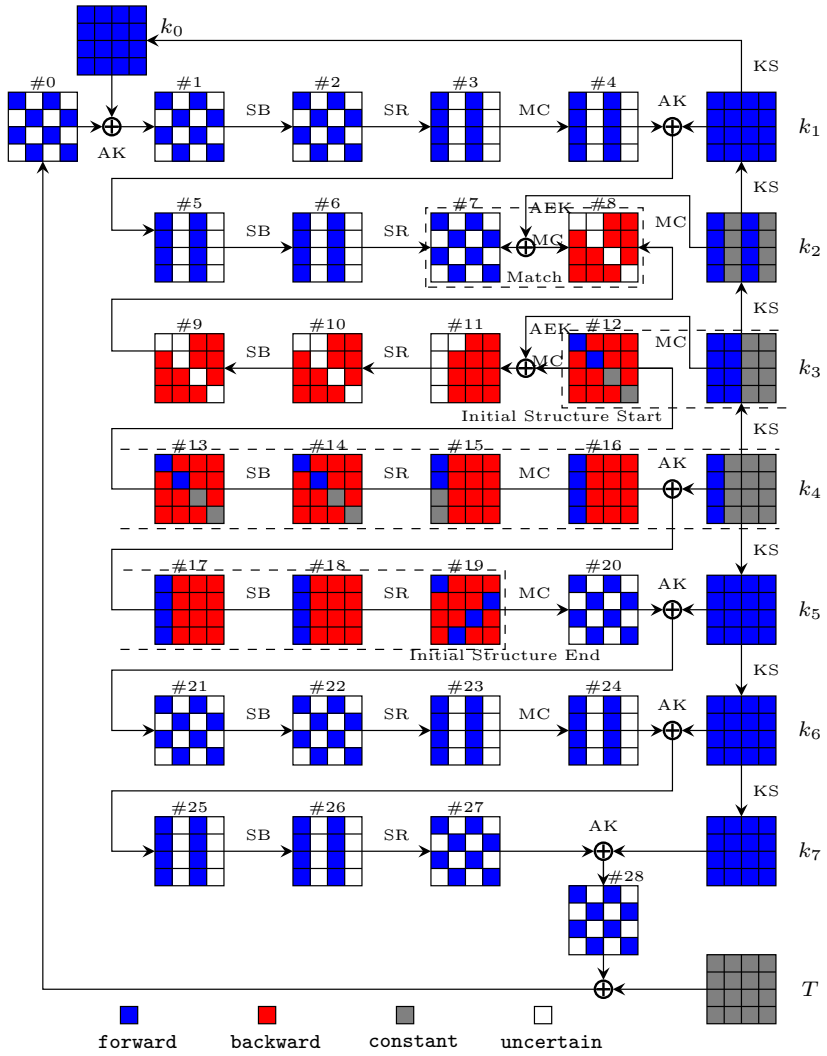


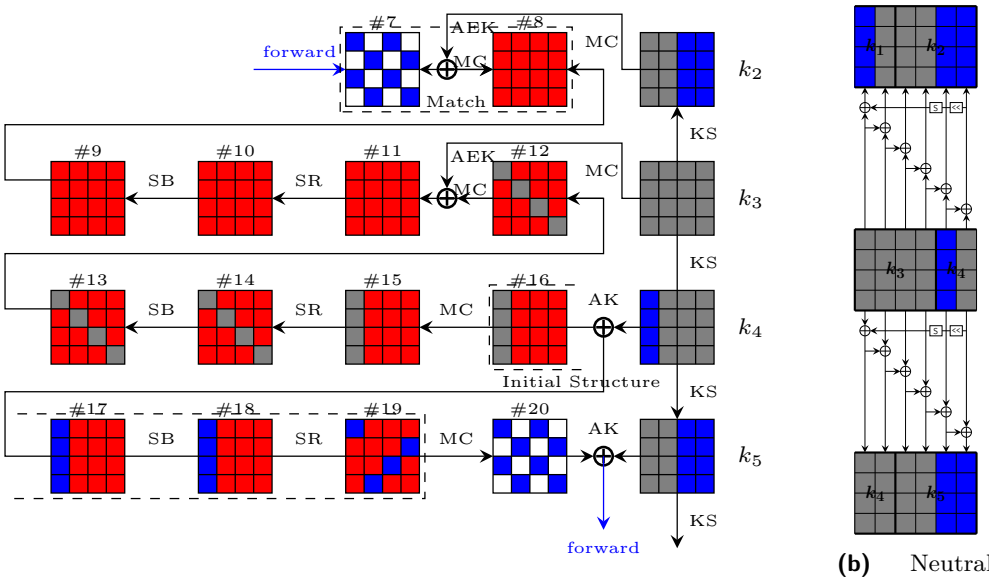
Figure 7: Introducing free bytes in the key to improve the 7-round preimage attack on AES-128 hashing mode

of AES-256 is even slower. Again, the full key state k_3 can be fixed as constant without being impacted by the neutral bytes in the first column of k_4. Thus, this attack directly applies to AES-256 hashing mode also.

In summary, in this attack, we introduce neutral bytes from the key and make them play a role in the second way, *i.e.*, $\mathbf{A} \oplus \bar{\mathbf{A}} = \mathbf{C}$. Due to the relatively low diffusion of the key-schedule, the introduced large number of neutral bytes from the key can completely cancel out the impact caused by the forward neutral bytes in the encryption state on the backward chunk, at the same time, leave an adequate degree of freedom for the forward chunk.

5.3 Application to 7-Round Kiasu-BC Hashing Mode

In the scenario where a tweakable block cipher is used in the PGV hashing mode and the tweaks can accept chosen inputs, freedom from this additional input might be exploited in similar attacks to the above ones.



(a) The attack configuration

(b) Neutral bytes in k_4

Figure 8: Introducing free bytes in the key to improve the 7-round preimage attack on AES-192 hashing mode

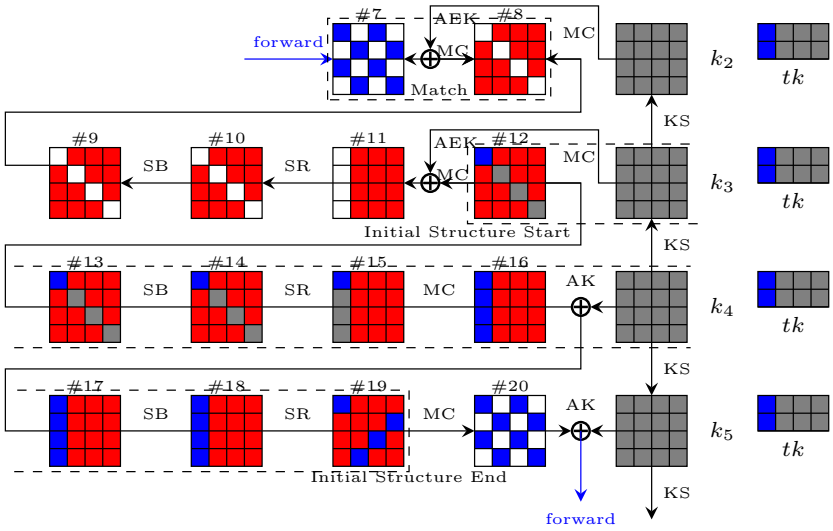


Figure 9: Introducing free bytes in the key to improve the 7-round preimage attack on Kiasu-BC hashing mode

Take Kiasu-BC for example whose encryption algorithm and key-schedule are exactly that of AES-128. The attack on 7-round AES hashing mode in [WFW+12] can be directly improved by introducing two additional neutral bytes for the forward computation from the tweak without any impact on the backward computation. Specifically, we choose $\#tk[0,1]$ as the neutral bytes for the forward chunk, which can take 2^{16} values and which has limited impact on the backward chunk (refer to Figure 9). In this way, together with the 2^8 freedom from neutral byte $\#12[0]$, we can finally obtain $2^{d_1=8+16=24}$ candidates in $M^{\mathtt{f}}$ without any cost (which is unlike in Attack 1 that reduces the number of bits for the match from 32 to 24). Corresponding to the complexity analysis equation Eq. (3) and compared with that of the attack in [WFW+12], the parameter d_1 increases from 8 to 24 and leaves all other parameters unchanged. Note that the number of bits for the match is still 32 which is unlike Attack 1. Thus, additional freedom in the target can still be utilized in the setting of a multi-target attack.

Consequently, for 7-round Kiasu-BC hashing mode, by introducing two neutral bytes from the tweak in addition to one neutral byte in the encryption state, and in the setting given 2^t targets, the attack will have a computational complexity $2^{128-\min(24+t,32)} = 2^{104-\min(t,8)}$ and a memory complexity $2^{24+\min(t,8)}$ (recall Eq. 3). Note that both the key and the tweak values are used here, so Type-III TBC-to-BC conversion is assumed here.

In summary, in this attack, we introduce neutral bytes from the tweak and make them play a role in the first way, *i.e.*, $\mathbf{A} \oplus \mathbf{A} = \mathbf{A}$. By adding neutral bytes from the tweak directly to neutral bytes in the encryption state without any restriction on the XOR sum, the gain of degree of freedom is completely free in the sense that the neutral bytes from the tweak brings no additional impact on the backward chunk (on top of the impact caused by the neutral bytes in the encryption state).

6 Extension to 8-Round Attacks

In this section, we extend the 7-round attacks of the previous section to 8-round attacks by introducing freedom from the key. We first explain the technique used in our extension, then provide its application to AES-256, AES-192, and Kiasu-BC.

6.1 Techniques for Attacking 8 Rounds

We add one more round to the backward chunk based on the previous 7-round attacks, and the new chunk separation is shown in Figure 10 (for a comparison, see Figure 7, 8, and 9). Notice that $\#17[0]$ is a neutral byte for the forward computation and can have 2^8 values, and it is denoted by x. We regard the whole key-schedule as a part of the forward chunk and set the value of $k_4[0]$ to be $x \oplus c$, where c is a constant value, and the left 15 bytes of k_4 equals to predefined constant values. In this way, the forward chunk is computed with the knowledge of neutral bytes from both key and internal states. While for the backward computation, the value of $\#16[0]$ is fixed to be c, thus states #13 to #15 can be computed deterministically. As for k_3, at most one column can be the forward neutral bytes, otherwise, #8 would have more than one unknown bytes in each column, which invalidates the matching step. And the neutral bytes in k_5 should not overlap with the neutral bytes for the backward chunk in state #20. The value of k_2 does not have any impact on the backward computation since the `MixColumns` is linear and we can instead find the match between $\#7 \oplus \mathrm{MC}^{-1}(k_2)$ and #8 through the `MixColumns`.

6.2 The 8-Round Attack against AES-256

Due to the slower diffusion in the key-schedule, we can apply the technique of Section 6.1 to the attack on the 8-round AES-256 hashing mode. The key path is initialized from the

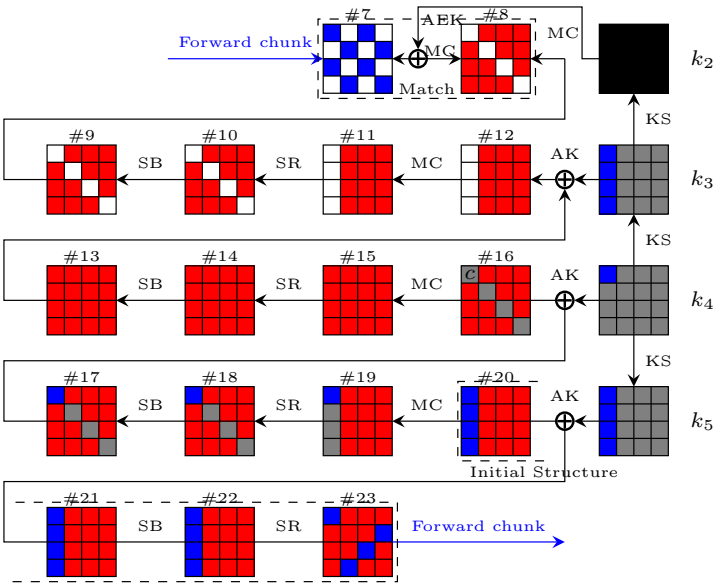
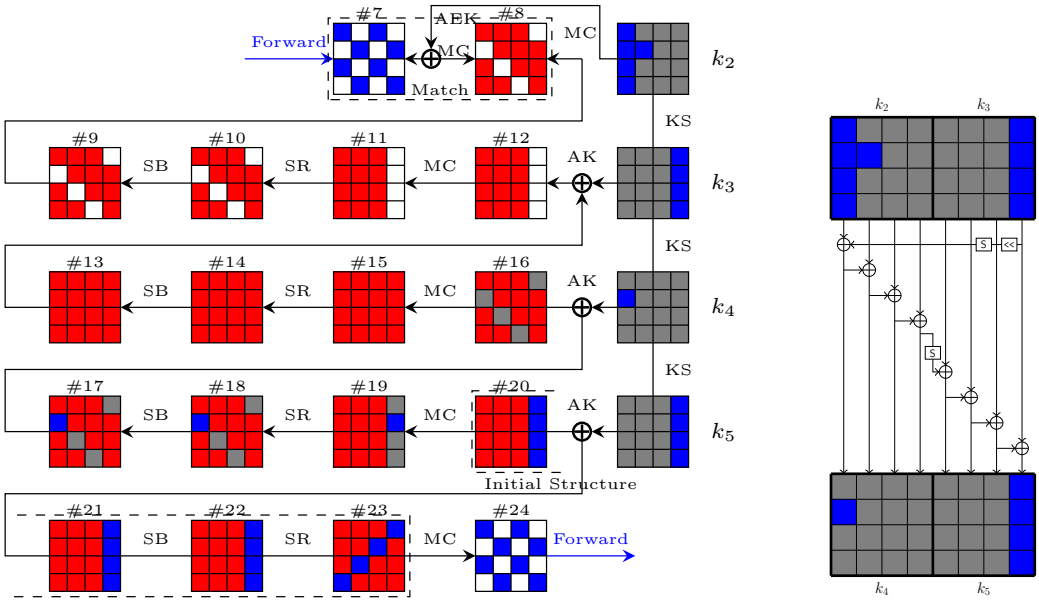


Figure 10: Basic idea of the extended 8-round attack, k_3 and k_5 can have at most one column of neutral bytes each.



(a) The attack configuration **(b)** Key states

Figure 11: Introducing free bytes in the key to launch an 8-round preimage attack on AES-256 hashing mode

256-bit state k_4 and k_5 in the way that the freedom in k_4 could neutralize the randomness in the backward chunk, and additional 2^{32} freedom degrees could be introduced in k_5, which is shown in Figure 11.

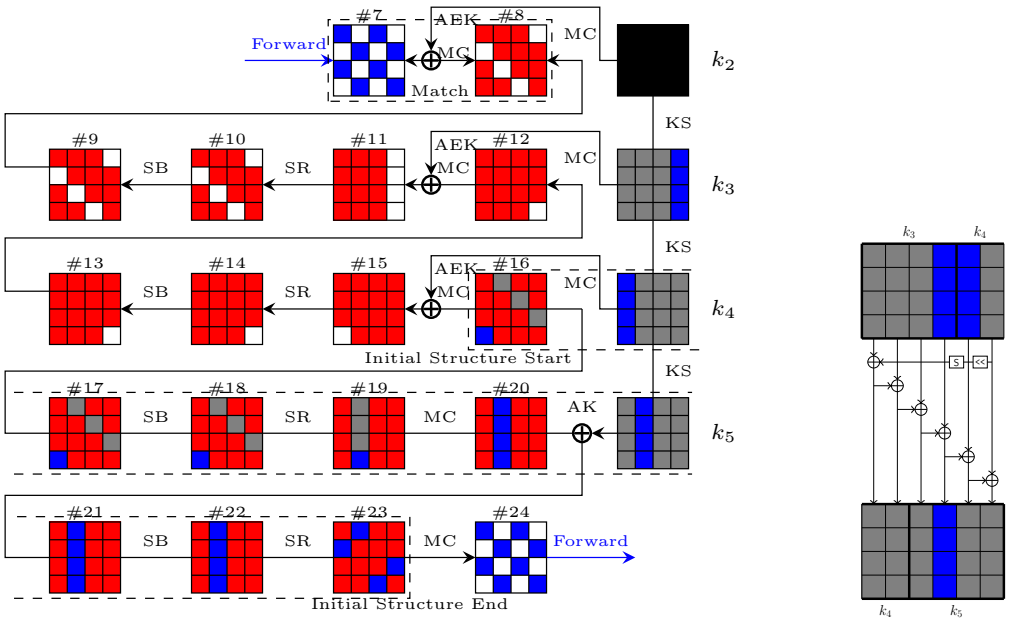
Attack 2: The pseudo-preimage attack on 8-round AES-256 hashing mode

Attack configuration

1. Initial structure:
   - Neutral bytes for forward (in encryption state): 2^8 possible values of #20[12,13,14, 15], s.t., #19[12,14,15] equals predefined constant C_0^{neut},
   - Neutral bytes for forward (in key state): 2^{32} possible values of $k_5[12, 13, 14, 15]$, s.t., their impacts on $\#19[12, 14, 15]$ equals predefined constant $C_1^{\mathtt{neut}}$. The value of $k_4[1]$ is chosen in the way so that its XOR difference with #17[1] equals to a predefined constant $C^{\mathtt{byte}}$, i.e., $k_4[1] = \#17[1] \oplus C^{\mathtt{byte}}$. As a result, for the backward chunk, states #13 to #16 can be computed deterministically.
     Other bytes in k_4 and k_5 equal to predefined constant $C^{\mathtt{key}}$.
   - Neutral bytes for backward: $(2^8)^4 = 2^{32}$ possible values of $\#23[0, 1, 2], \#23[4, 5, 7], \#23[8, 10, 11], \#23[13, 14, 15]$, s.t., their impacts on $\#24[0, 2], \#24[5, 7], \#24[8, 10], \#24[13, 15]$ equal predefined constant $C_2^{\mathtt{neut}} = C_{2,0}^{\mathtt{neut}} \| C_{2,1}^{\mathtt{neut}} \| C_{2,2}^{\mathtt{neut}} \| C_{2,3}^{\mathtt{neut}}$.
2. Chunk separation:
   - Forward chunk: the computation following #24 - #32 - #0 - #7
   - Backward chunk: the computation following #19 - #8
3. Bytes for match: Denote the equivalent sub-key of k_2 by uk_2, which can be computed via $\mathrm{MC}^{-1}(k_2)$:
   - $B^{\mathtt{f}} = (\#7 \oplus uk_2)[0, 2; 5, 7; 8, 10; 13, 15]$
   - $B^{\mathtt{b}} = \#8[0, 2, 3; 4, 5, 7; 8, 9, 10; 13, 14, 15]$

Attack procedure

1. Fix constants:
   - $C_0^{\mathtt{neut}}$: a value for 3-byte impact from neutral bytes in #20 on $\#19[12, 14, 15]$;
   - $C_1^{\mathtt{neut}}$: a value for 3-byte impact from $k_5[12, 13, 14, 15]$ on $\#19[12, 14, 15]$;
   - $C^{\mathtt{byte}}$: a value for byte #16[1];
   - $C^{\mathtt{key}}$: a 27-byte constant value in k_4 and k_5;
   - $C_2^{\mathtt{neut}}$: a value for 8-byte impact from neutral bytes in #23 on $\#24[0, 2, 5, 7, 8, 10, 13, 15]$.
2. Forward computation: for 2^8 values of $\#20[12, 13, 14, 15]$ s.t. their impact on $\#19[12, 14, 15]$ equals to $C_0^{\mathtt{neut}}$, and for 2^{24} out of 2^{32} values of $k_5[12, 13, 14, 15]$ s.t. their impact on $\#19[12, 14, 15]$ equals to $C_1^{\mathtt{neut}}$:
   - Compute the value of #17[1].
   - Determine the value of $k_4[1]$: $k_4[1] = \#17[1] \oplus C^{\mathtt{byte}}$.
   - Compute all sub-keys from k_4 and k_5 through the key-schedule.
   - Compute the forward chunk from #24 to #32, and from #0 to #7.



(a) The attack configuration

(b) Key states

Figure 12: Introducing free bytes in the key to launch an 8-round preimage attack on AES-192 hashing mode

- Compute $M^{\mathtt{f}}$ from $B^{\mathtt{f}}$, and store the results in a table $L^{\mathtt{f}}$.

3. Backward computation. For 2^{32} values of the neutral bytes #23[0, 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 15] s.t. their impact on bytes #24[0, 2, 5, 7, 8, 10, 13, 15] equals to $C_2^{\mathtt{neut}}$.
   - Compute the backward chunk from #19 to #8. The involved subkey bytes are constant in the backward computation.
   - Compute $M^{\mathtt{b}}$ from $B^{\mathtt{b}}$, and store the results in a table $L^{\mathtt{b}}$.
4. Matching: follow the same idea of the Attack 1.

Attack complexity The freedom degrees for both of the forward and backward chunk are 32, *i.e.*, $d_1 = d_2 = 32$, and the number of bits for matching comes from 32 bits ($m = 32$). Thus according to Eq. (1), the time complexity is 2^{96} and the memory complexity is 2^{32}.

In summary, in this attack, we introduce neutral bytes from the key and make them play the role in the second way, *i.e.*, $\mathbf{A} \oplus \bar{\mathbf{A}} = \mathbf{C}$.

6.3 The 8-Round Attack against AES-192

The main part of the attack is illustrated in Figure 12a. The data path in the internal state is similar to the attack on AES-256, the freedom for backward chunk remains $2^{d_2=32}$, but the neutral bytes for the forward in state #20 is changed to the second column. The way of introducing neutral bytes in the key is similar to the attack on 7-round AES-128: we introduce #16[3] as the neutral byte in encryption state and $k_4[0, 1, 2, 3]$ as neutral

bytes in the key state for the forward chunk, with the following constraints [4]

$$\begin{cases} \mathrm{MC}^{-1}((0,0,0,\#16[3]) \oplus k_4[0,1,2,3])[0,1,2] = C_1^{\texttt{neut}} \\ k_3[12,13,14,15] \oplus k_4[0,1,2,3] = C_1^{\texttt{key}} \end{cases}$$

Besides, all remaining bytes in the 192-bit subkey state (including the whole state of k_3 and the first two columns of k_4) are fixed as constant. According to the key-schedule of AES-192, we have $k_5[4,5,6,7] = k_3[12,13,14,15] \oplus \texttt{constant}$. In the constraints, the first equation ensures the impact caused by neutral bytes in the forward chunk on the backward chunk to be limited to a single byte (is constant in three bytes); the second equation together with the property of the key-schedule of AES-192 ensure that all the active bytes of the 192-bit subkey state are determined by predefined constants and the four neutral bytes without bringing further impact on backward chunk. Combining the neutral byte $\#16[3]$ in encryption state and neutral bytes $k_4[0,1,2,3]$ in the key state using the first linear equation, we obtain $d_1 = (5-3) \cdot 8 = 16$ bits of freedom for forward chunk.

Note that the number of bits for matching is $m = 32$ as can be seen in Figure 12a, which is larger than d_1. Thus, in a multi-target scenario, we can use freedom from the target to further balance the computation cost by the forward chunk and that by the backward chunk. Applying Eq. 3 to the complexity analysis, we conclude that this attack requires $2^{128-\min(16+t,32,32)} = 2^{112-\min(t,16)}$ computations of 8-round AES and $2^{\min(16+t,32,32)} = 2^{16+\min(t,16)}$ memory.

In summary, in this attack, we introduce neutral bytes from the key and make them play a role in the third way, *i.e.*, $\mathcal{L}(\mathbf{A},\mathbf{A}) = \mathbf{A}\|\mathbf{C}$. By introducing neutral bytes from the key, the impact caused by the forward neutral bytes on the backward chunk is partially canceled, and at the same time the freedom for the forward chunk is increased, which enables to extend the attack to more rounds with an improved complexity.

6.4 The 8-Round Attack against Kiasu-BC

Since Kiasu-BC adds the same tweak to the first two rows of internal state for each round, we can introduce neutral bytes from tweak instead of key values, and the attack configuration in Figure 10 can be directly adopted with the only modification in the key and the tweak: the key states are set to be constant values, and the tweak is initialized in such a way that $tk[0] = \#17[0] \oplus \texttt{constant}$ and the left 7 bytes are constant values. This attack follows the same framework as previous attacks, the degree for the forward chunk is $d_1 = 8$, for the backward chunk is $d_2 = 32$, and the number of bits for matching is $m = 32$, the freedom that lies in the target in multi-target setting can be utilized. Applying Eq. 3, we have that the time complexity is $2^{120-\min(t,24)}$ and the memory complexity is $2^{8+\min(t,24)}$. Note the tweak values are used here, but not the key values, so both Type-II and Type-III TBC-to-BC conversions fit the attack setting here. In summary, this attack is enabled by introducing neutral bytes from the tweak and make them play the role in the second way, *i.e.*, $\mathbf{A} \oplus \bar{\mathbf{A}} = \mathbf{C}$, so that extending to one more round is quite straightforward.

7 Discussion

Applying to Other Secure PGV Modes. As mentioned in Sect. 3.3, there are twelve secure PGV mode-of-operations to build a compression function from a block cipher. We note that for those modes in which the chaining state is fed into the key-schedule (*i.e.*, PGV modes in Class 1 and Class 3, refer to Table 2), because of the incompatibility between key size (192 or 256 bits) and state size (128 bits), AES-192/AES-256 are not applicable.

[4] Again, to obtain the values of neutral bytes fulfilling the constraints, one can efficiently solve the linear equations using linear algebra. Please refer to the example codes via https://www.dropbox.com/sh/c5dm28821f2jmfl/AAC3iGXUtA6crWaLwdS9LC2Ca?dl=0.

Apart from these incompatible instantiations of PGV modes by AES-192/AES-256, we conclude that the presented pseudo-preimage attacks are applicable to all the twelve PGV modes instantiated by AES.

Specifically, when H_{i-1} and M_i are of the same size, for our attacks, those PGV modes are essentially equivalent up to the exchange between H_{i-1} and M_i, and/or up to the XORing of whitening key (the material fed into the key-schedule of the block cipher), and/or up to replacing either H_{i-1} or M_i with $H_{i-1} \oplus M_i$. When applying the presented attacks, the output of the encryption, denoted by Z, and the input to the key-schedule, denoted by K, are firstly determined following the same MITM procedure for all PGV modes. The freedom required to find the correct Z and K are provided by H_{i-1} and M_i. The difference between the attacks on difference PGV modes lies in where the required freedom comes from. Thus, the equivalence up to the exchange between H_{i-1} and M_i is relatively easy to be understood, considering that for pseudo-preimage attacks, both H_{i-1} and M_i are unknown variables that are determined at the end of the attack procedure such that $\mathcal{H}(H_{i-1}, M_i) = T$ for the given target T. Examples for this case of equivalence are PGV No.1 $\equiv$ No.5, No.2 $\equiv$ No.6, No.3 $\equiv$ No.7, No.4 $\equiv$ No.8, No.9 $\equiv$ No.10, No.11 $\equiv$ No.12 (refer to Figure 13).

For the equivalence up to the XORing of the whitening key, the explanation is as follows. Recall that, in AES, the whitening key is the master key and is XORed at the very beginning of the encryption, and recall that our attacks use the splice-and-cut method. For PGV modes in which both of the material fed into the key-schedule and the material fed into the encryption are XORed at the end of the encryption, splicing in our attacks will result in a cancellation effect on the whitening key at the beginning of the encryption. However, as can be seen in Figure 7, whether the master key is XORed or is not XORed at the beginning of the encryption has no essential influence on our attacks. Thus, there is no essential difference for our attacks applying to those PGV modes. Examples for this case of equivalence are PGV No.2 $\equiv$ No.3, No.6 $\equiv$ No.7, No.1 $\equiv$ No.4, No.5 $\equiv$ No.8, No.9 $\equiv$ No.12, No.10 $\equiv$ No.11 (refer to Figure 13).

At last, it is easy to see that PGV No.1 and No.2 are essentially equivalent up to taking $H_{i-1} \oplus M_i$ as the variable providing the freedom required for finding correct Z, PGV No.1 and No.9 are essentially equivalent up to taking $H_{i-1} \oplus M_i$ as the variable providing the freedom required for finding correct K (refer to Figure 13).

In conclusion, for our attacks, the PGV modes are essentially equivalent and our attacks are applicable to them all (apart from the above mentioned incompatible instances).

Converting to (Second-) Preimage Attacks Whether or not the presented pseudo-preimage attacks can be converted to full preimage attacks depends on the padding rule of the hashing mode. That is because, all our pseudo-preimage attacks require that the message block can be freely chosen (for providing the freedom required by the input to key-schedule or by the input to the encryption). When the padding rule imposes special format to the last message block such that a valid padded message block cannot take arbitrary value (*i.e.*, the padding rule makes the domain of valid padded block smaller than that of normal blocks), converting our attacks to full preimage attacks requires additional computations (repeating the procedure of the pseudo-preimage attack for the last block so that the recovered block happens to be a valid one, *e.g.*, when using the padding rule in SHA-3 that reduces the domain by a factor of 2^{-1}, we expected to repeat two times; when using the Merkel-Damgård strengthening, we may need to repeat many times such that the overall complexity is higher than that of a brute-force attack).

Nevertheless, all our pseudo-preimage attacks can be converted into second-preimage attacks, in which the crafted messages reuse the last block of the given message, and the target is the second to the last chaining value in the processing of the given message. The complexity analysis on converting pseudo-preimage attack to preimage attack provided in

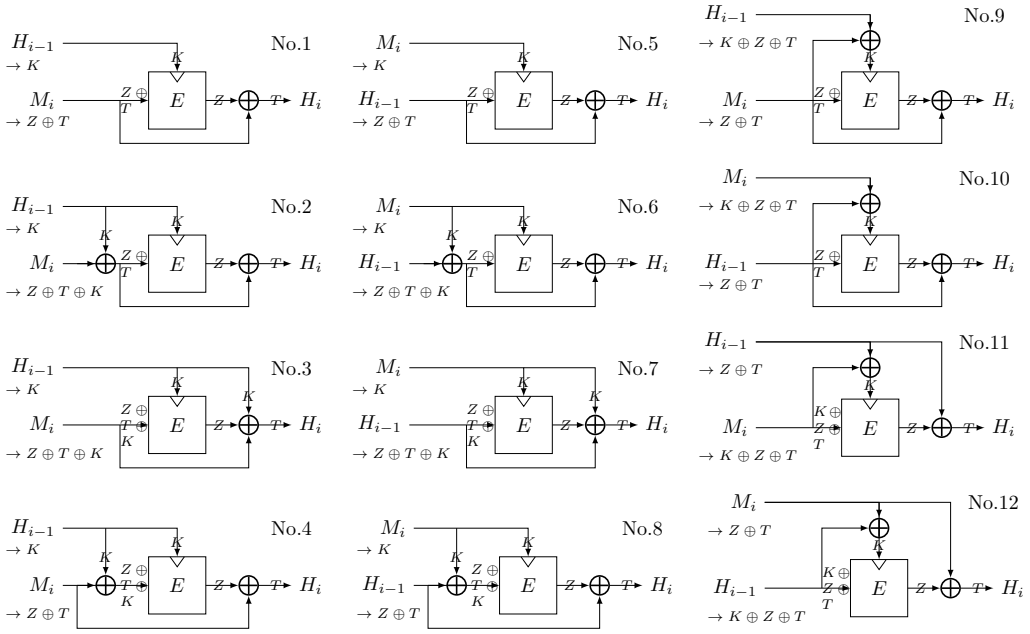


Figure 13: Equivalence among difference PGV modes for our attacks

Sect. 3.2 applies directly without any modification to the case of converting pseudo-preimage attack into a second-preimage attack.

8 Conclusion

Under the general framework of meet-in-the-middle preimage attack against AES hashing modes introduced by Sasaki in 2011 and improved by Wu *et al.* in 2012, we made two observations: the key bits are not used, and the neutral bits in the two chunks are not balanced in Wu *et al.*'s improvement. In this paper, we introduced neutral bits from the key, and to avoid dealing with the fast diffusion of the AES key-schedule, neutral bits from the key are introduced for one chunk only. By carefully choosing the key neutral bits, we found it was indeed possible while keeping the computation of the other chunk unaffected. Then the additional degrees of freedom are used in three ways to play two roles, *i.e.*, to reduce time complexities and to extend the attack to more rounds. As a result, we improved the MITM preimage attack complexities for 7-round AES hashing modes under all 3 versions of AES, and extended the attack to 8 rounds under AES-192 and AES-256. The same was applied to Kiasu-BC.

Acknowledgments

We thank Lei Wang for helpful discussions during the early phase of this work. We would like to thank all the reviewers of ToSC 2019 for their valuable comments and suggestions, and would like to specially thank Yu Sasaki for willing to be our shepherd. This research is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Strategic Capability Research Centres Funding Initiative (Grant No. M4062510.J30), Nanyang Technological University under grant M4082123, and Singapore's Ministry of Education under grants M4012049, M4012153, and M4020466. Wenying Zhang is supported by the National Natural Science Foundation of China (Grant No. 61672330).

References

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AGM+09] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for step-reduced SHA-2. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[AMM09] Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 120–135, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Heidelberg, Germany.

[AS09a] Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[AS09b] Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Heidelberg, Germany.

[AY14a] Riham AlTawy and Amr M. Youssef. Preimage attacks on reduced-round Stribog. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT 14: 7th International Conference on Cryptology in Africa*, volume 8469 of *Lecture Notes in Computer Science*, pages 109–125, Marrakesh, Morocco, May 28–30, 2014. Springer, Heidelberg, Germany.

[AY14b] Riham AlTawy and Amr M. Youssef. Second Preimage Analysis of Whirlwind. In Dongdai Lin, Moti Yung, and Jianying Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2014.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1465–1482, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[Bos16] Raphael Bost. $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1143–1154, Vienna, Austria, October 24–28, 2016. ACM Press.

[CL01] Bram Cohen and Ben Laurie. AES-hash. *NIST: Modes of Operation for Symmetric Key Block Ciphers*, 2001.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer*, 10(6):74–84, 1977.

[EFK15] Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 683–701, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[GLRW10a] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[GLRW10b] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full tiger, and improved results on MD4 and SHA-2. Cryptology ePrint Archive, Report 2010/016, 2010. http://eprint.iacr.org/2010/016.

[GSY15] Jian Guo, Chunhua Su, and Wun-She Yap. An improved preimage attack against HAVAL-3. *Inf. Process. Lett.*, 115(2):386–393, 2015.

[HKS10] Deukjo Hong, Bonwook Koo, and Yu Sasaki. Improved preimage attack for 68-step HAS-160. In Donghoon Lee and Seokhie Hong, editors, *ICISC 09: 12th International Conference on Information Security and Cryptology*, volume 5984 of *Lecture Notes in Computer Science*, pages 332–348, Seoul, Korea, December 2–4, 2010. Springer, Heidelberg, Germany.

[IP07] Sebastiaan Indesteege and Bart Preneel. Preimages for reduced-round tiger. In Stefan Lucks, Ahmad-Reza Sadeghi, and Christopher Wolf, editors, *Research in Cryptology, Second Western European Workshop, WEWoRC 2007, Bochum, Germany, July 4-6, 2007, Revised Selected Papers*, volume 4945 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2007.

[ISO10] ISO/IEC. 10118-2:2010 Information technology — Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher. 3rd ed., International Organization for Standardization, Geneve, Switzerland, October, 2010.

[Jea16] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[JNP14a] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. KIASU v1. Additional first-round candidates of CAESAR compeition, https://competitions.cr.yp.to/caesar-submissions.html, 2014.

[JNP14b] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[KK12] Simon Knellwolf and Dmitry Khovratovich. New preimage attacks against reduced SHA-1. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 367–383, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 - Efficient short-input hashing for post-quantum applications. *IACR Transactions on Symmetric Cryptology*, 2016(2):1–29, 2016. http://tosc.iacr.org/index.php/ToSC/article/view/563.

[Leu08] Gaëtan Leurent. MD4 is not one-way. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428, Lausanne, Switzerland, February 10–13, 2008. Springer, Heidelberg, Germany.

[LIS12] Ji Li, Takanori Isobe, and Kyoji Shibutani. Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to SHA-2. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 264–286, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[MvV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[SA08] Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271, Melbourne, Australia, December 7–11, 2008. Springer, Heidelberg, Germany.

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany.

[Sas11] Yu Sasaki. Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 378–396, Lyngby, Denmark, February 13–16, 2011. Springer, Heidelberg, Germany.

[WFW+12] Shuang Wu, Dengguo Feng, Wenling Wu, Jian Guo, Le Dong, and Jian Zou. (Pseudo) preimage attack on round-reduced Grøstl hash function and others. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 127–145, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[WS10] Lei Wang and Yu Sasaki. Finding preimages of Tiger up to 23 steps. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 116–133, Seoul, Korea, February 7–10, 2010. Springer, Heidelberg, Germany.

[WSK+11] Lei Wang, Yu Sasaki, Wataru Komatsubara, Kazuo Ohta, and Kazuo Sakiyama. (Second) preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 197–212, San Francisco, CA, USA, February 14–18, 2011. Springer, Heidelberg, Germany.