Security Evaluation of PC-MAC-AES

February, 2011

Lei Wang (University of Electro-Communications)
Kazuo Sakiyama (University of Electro-Communications)
Iwamasa Nishikado (Cyber Creative Institute)
Kazuo Ohta (University of Electro-Communications)

目 次

PC-MAC-AES の効率と安全性のトレードオフについての要旨

Summary On Efficiency/Security Tradeoff of PC-MAC-AES

PC-MAC-AES の効率と安全性のトレードオフについての要旨

本報告書では PC-MAC-AES 技術の妥当性を検証するために次の観点で検討した．

PC-MAC 構成法の安全性について

MAC 構成技術の歴史的な流れにそって，MAC 構成技術を図 4 のグループに分けて，PC-MAC に適用可能な攻撃法を網羅した（表 1). PC-MAC 固有の攻撃方法 Subkey-Recovery Attack と DIS-PC-MAC$^{E_K(\cdot),\mathrm{RP}}$ attack (Section 7 参照) を発見した。

提案された PC-MAC-AES では，高速化を実現するために 4 ラウンドの AES（本来の AES は 10 ラウンド）を構成要素として使用するので，PA-MAC の構成で 4 round AES を使った場合の安全性についても検討した．

その結果，第 8 章に示す新しい攻撃法を発見した．この攻撃法は図 4 で示す CBC MAC 配下のすべての構成法に対して有効であり，今回発見した攻撃方法は，PC-MAC 構成に対する固有の脆弱性ではない．

なお，今後も，PC-MAC-AES 固有の攻撃方法が発見される可能性は残されている．

補足: 本報告書で指摘したすべての攻撃法の計算量は，設計者によって与えられた安全性証明のバウンド，すなわち如何なる攻撃に要する攻撃の下限値よりも大きな値であった．よって，これらの攻撃法は PC-MAC-AES の安全証明と矛盾する，あるいはそれを否定するものではない．

PC-MAC-AES の効率について

ブロック長 l であるメッセージに対し、PC-MAC-AES は AES を $0.4 * l \sim 0.7 * l$ 回使う。CBC MAC は AES を l 回使う。PC-MAC-AES は CBC MAC より 1.4 $\sim$ 2.5 倍速い。

CRYPTREC 候補とするかの判断について

CMAC [16] と比較して，その優位性が確認できれば，CRYPTREC に採用してよいと考える．

安全性の観点からは，Subkey-Recovery Attack と DIS-PC-MAC$^{E_K(\cdot),\mathrm{RP}}$ attack (Section 7 参照) が可能なので，PC-MAC 構成は劣っている．速度の観点からは，PC-MAC-AES は CMAC に比べて 1.4 $\sim$ 2.5 倍高速である[1]．

[1]CMAC は一部で 4 ラウンドの AES を使うと、CMAC の方がコンパクトな実装が可能と思われるが、安全性証明ができなくなると思われる。両方式の比較は継続検討が必要と思われる．

Summary On Efficiency/Security Tradeoff of PC-MAC-AES

In order to contribute to a fair judgment on PC-MAC-AES, we will make a comparison between PC-MAC-AES [15] and three other MACs based on AES: EMAC [1], MacDES [11][2] and TMAC [12], which are described in Figures 1, 2 and 3, respectively. We refer the specification of PC-MAC-AES to main body of this report (Section 2). The reasons why we select these three MACs are:

I. each of these MACs uses two independent keys, which is the same with PC-MAC-AES;

II. the differences between these MACs will be used by us to separate CBC MAC variants into several groups (see Figure 4). We will discuss the difference of the security margin (by proposing concrete attacks) among these groups, whose result is summarized in Table 1.

1. High Efficiency of PC-MAC-AES

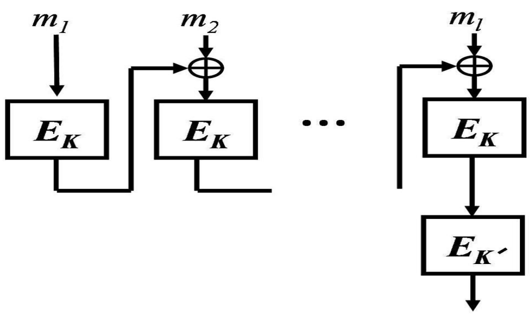
Among these MACs, the most efficient MAC is PC-MAC-AES, and the slowest one is MacDES. The detailed argument is as follows.

Suppose that an input message (after padding) is l blocks long. We will count the number of executions of AES (full version: 10 rounds). Moreover, we will consider the complexity on average, and thus ignore the complexity of pre-process.

- MacDES: $l + 2$ executions.
- EMAC-AES: $l + 1$ executions.
- TMAC-AES: l executions.
- PC-MAC-AES: suppose $l = (d+1) \times t + s$, where d is a positive integer as a parameter of PC-MAC-AES, t and s are positive integers, $t \leq \frac{l}{d+1}$ and $s \leq d$. The number of executions of AES is $(1+0.4*d)*t+0.4*(s-1)+1 \leq \frac{(1+0.4*d)*l}{d+1} + 0.4*d+1$. Loosely speaking, PC-MAC-AES needs $0.4*l \sim 0.7*l$ executions, which depends the value of d.

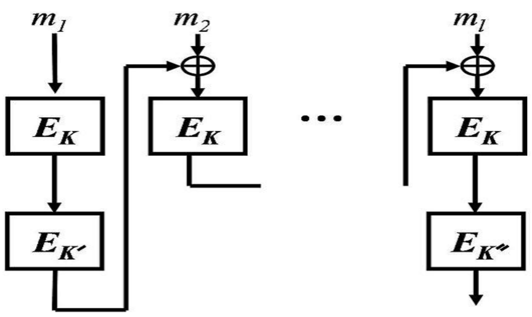
Therefore PC-MAC-AES is about 1.4 to 2.5 times faster than the other three MACs.

[2]MacDES is designed based on the block cipher DES. Here for a comparison, we will consider a MacDES variant which is based on AES.



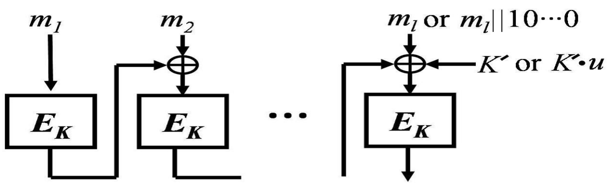
K and K' are two independent secret keys.

Figure 1: EMAC



K and K' are two independent secret keys, and K'' is derived from K'.

Figure 2: MacDES



K and K' are two independent secret keys.

Figure 3: TMAC

2. Security Margin Loss of PC-MAC-AES

Improving efficiency is usually with a loss of security margin. First of all, we will discuss the security margin loss of PC-MAC [13], compared with EMAC [1], MacDES [11] and TMAC [12]. Then we will discuss the security margin loss of CBC MAC variants based on 4-round AES, compared with those based on full-round AES. Finally we make a comparison between PC-MAC-AES and CMAC [16], which was recommended for block-cipher-based authentication mode by NIST in 2005, on the efficiency/security tradeoff.

2.1 Security Margin Loss of PC-MAC

PC-MAC seems not designed completely from scratch. It seems to have adopted several design features of priorly proposed MACs. The historical development from original CBC MAC [2] to PC-MAC is shown in Figure 4.

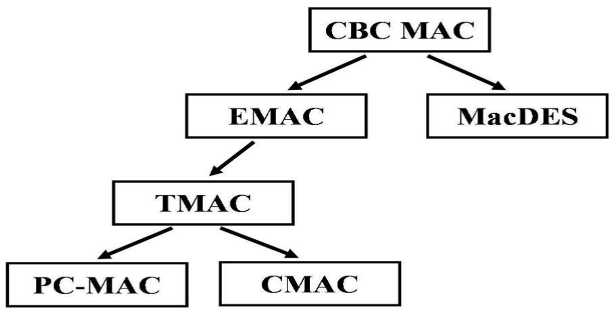


Figure 4: Historical Development of CBC MAC Variants

We will separate the security margin loss of PC-MAC into two cases.

- Loss because of adopting prior design features.

  Namely the security loss is not restricted to just PC-MAC, but shared with other priorly proposed MACs. We have summarized several features of PC-MAC.

  - *On-the-fly.* This feature is from original CBC MAC, and shared with all its variants.

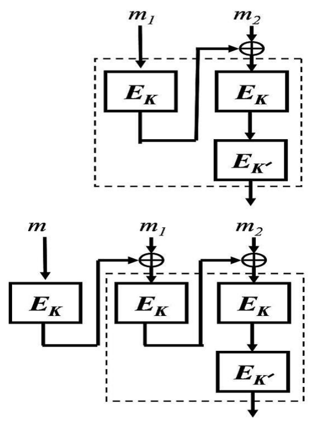
    Loosely speaking, *on-the-fly* structure means that

    1) message is divided to blocks;
    2) blocks is hashed sequentially; and
    3) each block is hashed only once.

– *Suffix.* This feature is from EMAC, and shared with its variants such as XCBC [4], TMAC and OMAC [8]. EMAC and its variants differ from original CBC MAC only in the last block cipher operation.

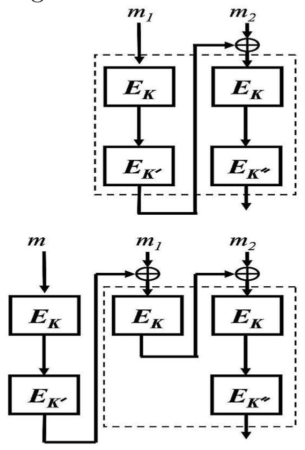
  Loosely speaking, *suffix* structure means that the iteration sequences of block ciphers for processing message M_2 and for part M_2 of $M_1||M_2$, where M_1 is a message of multiple-block length, are the same.

  EMAC is a suffix MAC, while MacDES is a suffix-free MAC. And the reason is explained in Figures 5 and 6.



The iteration sequences of block ciphers to process $m_1||m_2$ are the same: $E_K||E_K||E_{K'}$.

Figure 5: EMAC is suffix



The iteration sequences of block ciphers to process $m_1||m_2$ are different: $E_K||E_{K'}||E_K||E_{K''}$ and $E_K||E_K||E_{K''}$.

Figure 6: MacDES is suffix-free.

  - *Tweaking key.* This feature is from XCBC, and shared with its variants such as TMAC and OMAC [8].

    Loosely speaking, *tweaking key* means that a secret key is used to XOR the input of the last block cipher operation.

    XCBC uses two independent tweaking keys: one is for messages with multiple blocks length and the other one is for messages with non-multiple block length. Later TMAC made a modification, which uses only one secret tweaking key, and the other tweaking key is derived from it. PC-MAC follows the same way with TMAC.

- Loss because of new design features of PC-MAC-AES.

  Namely the security loss is restricted to PC-MAC. We have summarize one new feature of PC-MAC.

  - *Subkey generation algorithm.* All the keys of auxiliary permutations are generated by using the tweaking key and the block cipher.

In the following, we will discuss the security margin loss of PC-MAC in detail. Again in order to contribute to a fair judgment on PC-MAC, we will discuss the security margin loss (by pointing out concrete attacks) from original CBC MAC to PC-MAC, which is also summarized in Table 1.

2.1.1 Security Margin Loss of CBC MAC and MacDES

With a complexity of **the birthday bound**, an attacker is able to carry out

- Distinguishing MAC from a random function. (Section 4.1)
  Attacker will identify an oracle of being either MAC or a random function.
- Existential Forgery Attack. (Section 4.2)
  Attacker will provide a message/tag pair, where the message was not priorly queried to MAC.
- Selective Forgery Attack. (Section 4.3)
  Attacker will select a message she needs, and then forge a valid tag without directly querying it to MAC.

2.1.2 Security Margin Loss of EMAC

With a complexity of **the birthday bound**, besides the above attacks, an attacker is further able to carry out

- Internal-State-Recovery Attack. (Section 5.1)
  Attacker will recover the value of an internal state of some message.

- Universal Forgery Attack. (Section 5.2)
  Attacker is able to forge valid tags for any given message without querying it to MAC.
- Low Complexity-on-Average Universal Forgery Attack. (Section 5.2)
  Here by *complexity-on-average*, we mean the attacker will forge more than one message, and we count the complexity of forging one message on average.

Moreover, the attacker is also able to carry out

- Full-Key Recovery Attack. (Section 5.3)
  Suppose the master key has k bits, and the tweaking key has n bits. It takes $2^{n/2+1}$ queries, $2^{n/2+1}$ memory and $2^{k+1} + 2^k$ computations to recover both of them, which is faster than the brute-force attack.

2.1.3 Security Margin Loss of TMAC

With a complexity of **the birthday bound**, besides all the above attacks, an attacker will further be able to carry out

- Partial-key Recovery Attack. (Section 6.1)
  The secret key, which is used to tweak the last block, will be recovered.
- Low Complexity-on-Average Internal-State Recovery Attack. (Section 6.3)

Moreover, the attacker will be able to carry out

- Full-Key Recovery Attack Faster than the Attacks on EMAC. (Section 6.4)
  There are two improved key-recovery attacks. The first approach is to save memory, which needs 2 queries, 2 memory and 2^{k+1} computations. The second approach is to save time complexity, which needs $2^{n/2+1}$ queries, $2^{n/2+1}$ memory and $2^k + 2^{k/2+1}$ computations.

2.1.4 Security Margin Loss of PC-MAC

With a complexity of **the birthday bound**, besides the above attacks, an attacker will be able to carry out

- Subkey-Recovery Attacks. (Section 7.1)
  All the keys of auxiliary permutations will be recovered.
- Distinguishing a PC-MAC based on a dedicated block cipher, e.g. AES, from PC-MAC based on random permutations. (Section 7.2)

Table 1: On Security Margin Loss of CBC MAC Variants

Birthday Bound Attacks ($2^{n/2}$)		
Features	Applicable MACs	Attack Scenario

On-the-fly;	Most CBC MAC variants	$\mathrm{DIS}^{\mathrm{MAC,RF}}$;
		Existential forgery attack;
		Selective forgery attack;

On-the-fly;
Suffix;
	EMAC; XCBC;
	ANSI retail MAC; [3]
	TMAC, CMAC [16];
	PC-MAC
		$\mathrm{DIS}^{\mathrm{MAC,RF}}$;
		Existential forgery attack;
		Selective forgery attack;
		Internal-state recovery attack;
		Universal forgery attack;
		Low complexity-on-average universal forgery attack;

On-the-fly;
Suffix;
Tweaking key;
	XCBC; TMAC;
	CMAC;
	PC-MAC
		$\mathrm{DIS}^{\mathrm{MAC,RF}}$;
		Existential forgery attack;
		Selective forgery attack;
		Internal-state recovery attack;
		Universal forgery attack;
		Low complexity-on-average universal forgery attack;

	TMAC, CMAC; **PC-MAC**;	Low complexity-on-average Internal-State-Recovery Attack
	TMAC; **PC-MAC**;	Partial-key Recovery Attack;
Specific Attacks on **PC-MAC**: Subkey Generation Algorithm		Subkey-Recovery Attack; $\mathrm{DIS\text{-}PC\text{-}MAC}^{E_K(\cdot),\mathrm{RP}}$;
Attack beyond Birthday Bounds (2^k)		

On-the-fly;
Suffix;
	EMAC; XCBC;
	ANSI retail MAC;
	TMAC; **PC-MAC**;
		Full-key recovery attack;
		(complexity: 2^{k+1})

On-the-fly;
Suffix;
Tweaking key;
	TMAC; **PC-MAC**;	Full-key recovery attack;
		(complexity: 2^k)

We refer $\mathrm{DIS}^{\mathrm{MAC,RF}}$ to as distinguishing a MAC from a random function, and $\mathrm{DIS\text{-}PC\text{-}MAC}^{E_K(\cdot),\mathrm{RP}}$ to as distinguishing PC-MAC based on $E_K(\cdot)$ from PC-MAC based on random permutations. Denote the length of tag and key as n and k respectively.

Statement. All these attacks have complexities larger than the security bound proved by the designers. Thus, these attacks don't contradict with or disprove the provable security bound of PC-MAC-AES.

2.2 Security Margin Loss of Using 4-Round AES

For a CBC MAC variant using 4-round AES, besides the attacks listed in Table 1, with a complexity of **the birthday bound**, an attacker is able to carry out

- Internal-state-recovery attack. (Section 8.3)
- Subkey-recovery attack. (Section 8.2)
  The attacker is able to recover the round keys of 4-round AES.

Of course these attacks are applicable to PC-MAC-AES.

2.3 Comparison between CMAC and PC-MAC on Efficiency/Security Tradeoff

NIST has recommended CMAC [16] for block-cipher-based authentication in 2005. Here we briefly make a comparison between CMAC and PC-MAC.

- On efficiency. For l blocks long messages, CMAC based on AES needs l executions, while PC-MAC-AES needs $0.4 * l \sim 0.7 * l$ executions. Thus PC-MAC-AES is $1.4 \sim 2.5$ times faster than AES-based CMAC.
- On security. As shown in Figure 4, CMAC also have features: *on-the-fly*, *suffix* and *tweaking key*. Thus the security margin difference between CMAC and PC-MAC will be the attacks based on new features of PC-MAC, which are subkey-recovery and DIS-PC-MAC$^{E_K(\cdot),\mathrm{RP}}$ attacks. Note CMAC does not have subkeys. Thus the security margin loss from CMAC to PC-MAC will be DIS-PC-MAC$^{E_K(\cdot),\mathrm{RP}}$ attacks.

PC-MAC-AES の安全性評価

（本文）

Security Evaluation of PC-MAC-AES

(Main Body)

Contents

Abstract of Main Body

This report will evaluate the security of PC-MAC-AES and its underlying components. This report consists of 9 sections.

Section 1 will briefly introduce the backgrounds of Message Authentication Code (MAC). More precisely, we will introduce the security requirements on MAC, and common attack models on MAC.

Section 2 will briefly recall the specification of PC-MAC-AES and its design rationale, which has been included in the designers' report.

Section 3 will make a comparison between PC-MAC-AES and other block-cipher-based MACs, and derives several interesting features: *on-the-fly*, *suffix*, *Tweaking key*, which are shared by PC-MAC-AES and other MACs. On each feature, we will specify MACs which share it with PC-MAC-AES. Moreover, we derive new features of PC-MAC. This will contribute to a fair understanding of the meaning of our attacks on PC-MAC-AES in Section $4 \sim 7$.

Sections $4 \sim 7$ will evaluate the security of PC-MAC-AES, which is the main part of this report. We will describe distinguishing, forgery, internal-state-recovery, and key-recovery attacks on PC-MAC-AES.

Previous Works. Strictly speaking, so far only two papers were published explicitly on the evaluation of PC-MAC-AES. One was published by Yuan *et al.* in CRYPTO 2009 [20], which proposed internal-state-recovery and full-key-recovery attacks. The other one was published by Jia *et al.* in CANS 2009 [9], which proposed a forgery attack. However, PC-MAC-AES shares similar structures with other MACs such as TMAC and CMAC. Therefore previous published attacks on other MACs may also be applicable to PC-MAC-AES.

- **The designers have mentioned the possible applicability of previous attacks, which were published to attack other MACs instead of PC-MAC-AES, in their report. However, they did not state which previous attacks can be adapted to attack PC-MAC-AES and how much the complexity will be.**

Our Works. We divide our evaluations into two cases.

Case I. We will summarize and describe several attacks including distinguishing, forgery and key-recovery attacks on PC-MAC-AES, most of which were previously proposed to attack other prior MACs. Interestingly, our full-key-recovery attacks on PC-MAC-AES are more efficient than Yuan *et al.*'s attack [20]. [3]

Case II. We will propose several novel attacks on PC-MAC-AES, which are based on its new features.

[3] On the other hand, Yuan *et al.*'s attack seems to have wider applications.

1. *Subkey-recovery attacks.*
   Subkeys $\{U_1, U_2, \ldots, U_d, K_1^{xor}, \ldots, K_{d-1}^{xor}\}$ are generated by using L and $E_K(\cdot)$. We showed that after obtaining the knowledge of L, these subkeys can be easily recovered. The complexity is just one chosen query for each subkey.
2. *Distinguish PC-MAC-AES from PC-MAC-RP.*
   We will propose an attack procedure, which can distinguish PC-MAC-AES from another PC-MAC instantiation with a Random Permutation (RP).

Section 8 will focus on an interesting question:

what potential weakness will the usage of 4-round AES introduce to CBC-MACs including PC-MAC-AES and its variants?

As we can see, the most attractive design point of PC-MAC-AES is using 4-round AES permutations instead of the full-round AES to process partial message blocks, which will improve the efficiency, but still keep provable security. The theoretical foundation of such replacement is that 4-round AES with independent round keys have similar differential property with full-round AES. Besides PC-MAC-AES, several other MACs such as Pelican-MAC [6] are also using 4-round AES to improve the efficiency. On the other hand, a question will arise: will such replace weaken the security of CBC-MACs from any sense?

Our answer is *yes*. We will propose an attack on generic CBC-MAC variants based on 4-round AES with independent round keys. More precisely, we are able to recover the round keys and the internal state value with a complexity of 2^{67} online queries and 2^{40} offline computations.

Our attack implies an upper bound of subkey-recovery resistance for CBC-MAC variants based on 4-round AES.

Section 9 will conclude this report.

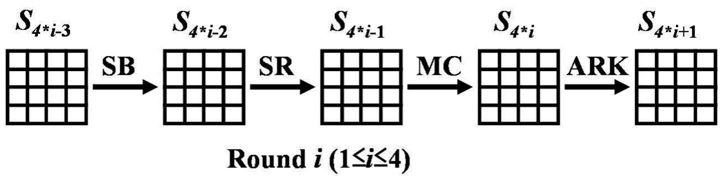
Abbreviations and Notations

AES

- AES ······ Advanced Encryption Standard
- P ······ Plaintext
- C ······ Ciphertext
- SB ······ SubByte
- SR ······ ShiftRow
- MC ······ MixColumn
- ARK ······ AddRoundKey
- AES-4R ······ 4-Round AES with independent round keys
- Byte order of 4×4 byte matrix:

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

- Internal state notation $\{S_1, S_2, \ldots, S_{16}\}$ in AES-4R:



- $S_{i,j}$ ······ Byte j of S_i
- $\Delta S_{i,j}$ ······ Difference at byte $S_{i,j}$

PC-MAC-AES

- MAC ······ Message Authentication Code
- UF-CMA ······ UnForgeability against Chosen Message Attack
- CBC ······ Cipher Block Chaining
- PCH ······ Periodic CBC Hash
- $\mathrm{TG}_K(\cdot)$ ······ Tag Generation oracle
- $\mathrm{VF}_K(\cdot,\cdot)$ ······ tag VeriFication oracle

- M $\cdots\cdots$ Message
- T $\cdots\cdots$ Tag
- K $\cdots\cdots$ Secret key
- $E_K(\cdot)$ $\cdots\cdots$ AES with the seret key as K
- L $\cdots\cdots$ Tweaking key
- $\{G_1, \ldots, G_d\}$ $\cdots\cdots$ AES-4R in PC-MAC-AES
- U_i $(= U_i^1 || U_i^2 || U_i^3)$ $\cdots\cdots$ Round keys in G_i
- $\{K_1^{xor}, \ldots, K_{d-1}^{xor}\}$ $\cdots\cdots$ Mask keys

Others

- $\oplus$ $\cdots\cdots$ Bitwise XOR
- $\ll 1$ $\cdots\cdots$ Left shift by 1 bit
- $a||b$ $\cdots\cdots$ a concatenated with b
- $|a|$ $\cdots\cdots$ bit length of a
- $[t]$ $\cdots\cdots$ 128-bit binary encode of integer t
- MDP $\cdots\cdots$ Maximum Differential Probability
- MEDP $\cdots\cdots$ Maximum Expected Differential Probability
- AUHF $\cdots\cdots$ Almost Universal Hash Function
- RF $\cdots\cdots$ Random Function
- PRF $\cdots\cdots$ Pseudo-Random Function
- RP $\cdots\cdots$ Random Permutation
- $\mathcal{A}$ $\cdots\cdots$ Attacker
- $\mathrm{DIS}^{\mathrm{MAC,RF}}$ $\ldots\ldots$ Distinguish MAC from RF
- $\mathrm{DIS\text{-}PC\text{-}MAC}^{\mathrm{AES,RP}}$ $\ldots\ldots$ Distinguish PC-MAC-AES from PC-MAC-RP

List of Figures

1 Backgrounds

This section consists of

- Security requirements on MAC
- Common attacks on MAC

Message Authentication Code (MAC) is a symmetric-key cryptosystem, which provides a short tag for a message to protect both its integrity and its authenticity. Suppose Alice and Bob share a common key K, and agree on a tag generation protocol $TG_K(\cdot)$. When Alice sends a message M to Bob, she will send both M and $TG_K(M)$. When Bob receives the message/tag pair, he will compute $TG_K(M)$ using his own copy of K, and check whether it matches with the tag from Alice. If it matches, Bob is assured that

1) the message is from Alice; and

2) the message is not modified by a third party.

1.1 Security Requirements on MAC

The most important security requirement on MAC is *unforgeability*: it should be hard for a third party without the knowledge of the secret key K to generate a valid tag for a message. The following is the formal definition of *Unforgeability against Chosen Message Attack (UF-CMA).*

Definition 1.1.0.1 *Let TG:* $\{0,1\}^k \times \{0,1\}^* \longrightarrow \{0,1\}^n$ *be a message authentication code. For a randomly chosen key* $K \longleftarrow \{0,1\}^k$*, denote by* $TG_K(\cdot)$ *the tag generation oracle. Denote by* $VF_K(\cdot,\cdot)$ *the tag verification oracle, which outputs 1 if the message/tag pair is valid, and outputs 0 otherwise. If within practical resource (time, memory etc) no attacker* $\mathcal{A}$ *with access to* $TG_K(\cdot)$ *and* $VF_K(\cdot,\cdot)$ *can produce a message/tag pair of* (M,T) *satisfying the following two conditions with a non-negligible probability:*

- $VF_K(M,T) = 1$*; and*
- $\mathcal{A}$ *did not priorly query* M *to* $TG_K(\cdot)$*,*

we will say this MAC satisfies Unforgeability against Chosen Message Attack (UF-CMA).

A random function (RF) surely can satisfy UF-CMA, because the success probability of any attacker forging RF is at most 2^{-n}. Due to the following theorem, if a MAC is proven indistinguishable from RF, it satisfies UF-CMA.

Theorem 1.1.0.1 *Denote by $Adv^{uf\text{-}cma}(q, \sigma)$ the upper bound of success probability of a forgery attacker, where q and σ are the number of queries and the total number of queried message blocks made by the attacker. Denote by $Adv^{prf}(q, \sigma)$ the upper bound of success probability of an attack distinguishing MAC from a random function. The following relation holds.*

$$Adv^{uf\text{-}cma}(q, \sigma) \leq Adv^{prf}(q, \sigma) + \tfrac{q}{2^n}$$

Thus, a stronger security property on MAC is *Pseudo-Random Function*. The formal definition is as follows.

Definition 1.1.0.2 *Let TG: $\{0,1\}^k \times \{0,1\}^* \longrightarrow \{0,1\}^n$ be a message authentication code. For a randomly chosen key $K \longleftarrow \{0,1\}^k$, denote by $TG_K(\cdot)$ the tag generation oracle. Let $RF(\cdot)$ be a random function having the same domain and range with $TG_K(\cdot)$. If within practical resource no distinguisher can distinguish $TG_K(\cdot)$ from $RF(\cdot)$ with a non-negligible probability, we will say this MAC satifies Pseudo-Random Function (PRF).*

1.2 Common Attacks on MAC

There are several types of attacks on MACs. Denote by $\mathcal{A}$ the attacker.

- **DIS$^{\mathbf{MAC,RF}}$:** $\mathcal{A}$ will identify an oracle of either $\mathrm{TG}_K(\cdot)$ or RF. Namely $\mathcal{A}$ aims to evaluate the PRF property of MAC.
- **DIS$^{\mathbf{MAC\text{-}BC,\ MAC\text{-}RP}}$**: suppose a MAC iterates a block cipher (BC).$\mathcal{A}$ will distinguish a block cipher (BC) from a random permutation (RP) under the MAC environment. Namely, $\mathcal{A}$ will identify an oracle of MAC iterating either BC or a RP.
- **Forgery attack:** $\mathcal{A}$ is able to access to $\mathrm{TG}_K(\cdot)$ and $\mathrm{VF}_K(\cdot, \cdot)$, and will produce a valid tag T for a message M, which was not priorly queried to $\mathrm{TG}_K(\cdot)$.

  There are three types of forgery attacks.

  - **Existential forgery attack:** no constraint is set on M. Thus M could be any message, e.x. without any particular meaning.
  - **Selective forgery attack:** M is selected by $\mathcal{A}$.
  - **Universal forgery attack:** for any message M, $\mathcal{A}$ can forge a valid tag.
- **Internal-state recovery attack:** $\mathcal{A}$ is able to access to $\mathrm{TG}_K(\cdot)$, and will recover an intermediate state of some message.
- **Key recovery attack:** $\mathcal{A}$ is able to access to $\mathrm{TG}_K(\cdot)$, and will recover the secret key K.

2 PC-MAC-AES

This section consists of

- Specification of PC-MAC-AES
- Design rationale of PC-MAC-AES
- Comparison with other MACs

2.1 Specification

2.1.1 AES

The Advanced Encryption Standard (AES) [7] is a block cipher adopting the Substitution-Permutation Network. AES has three variants differing in the key size and the number of rounds. We will mainly deal with one variant, which uses a 128-bit key and consists of 10 rounds. In the rest of this paper, we refer AES to as this variant.

The plaintext block size is 128 bits and represented by a byte matrix of size 4×4. The state matrix will go through a serial of rounds. Each round applies the following four operations to update the state matrix., which is also illustrated by Fig. 7.

SubBytes (SB) : apply a 8-bit to 8-bit S-box on each byte of the state matrix;

ShiftRows (SR) : cyclic left shift the i-th row of the state matrix by i bytes $(0 \leq i \leq 3)$;

MixColumns (MC) : multiply each column of the state matrix by a constant 4×4 matrix.

AddRoundKeys (ARK) : XOR the state with a 128-bit subkey;

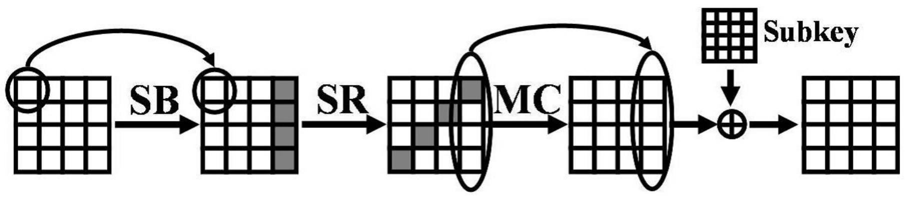


Figure 7: One AES Round Operation $\text{ARK} \circ \text{MC} \circ \text{SR} \circ \text{SB}(\cdot)$

Moreover, the first round applies an additional ARK operation for whitening the plaintext, which becomes ARK $\circ$ MC $\circ$ SR $\circ$ SB $\circ$ ARK$(\cdot)$. The last round omits the MC operation, which becomes ARK $\circ$ SR $\circ$ SB$(\cdot)$.

AES transforms the 128-bit key k to eleven 128-bit subkeys $\{k_0, k_1, \ldots, k_{10}\}$. The algorithm is as follows, which is also illustrated in Fig. 8. k is divided into four 32-bit words $W_0||W_1||W_2||W_3$, which is expanded to $W_0||\cdots||W_{43}$, and regrouped to construct k_i: $W_{4i}||W_{4i+1}||W_{4i+2}||W_{4i+3}$. The expansion algorithm is as follows.

```
**for** $i = 4$ to 43 **do**
  **if** $i = 0$ mod 4 **then**
    $W_i = W_{i-4} \oplus \text{Rotate}(\text{SB}(W_{i-1})) \oplus C$
  **else**
    $W_i = W_{i-1} \oplus W_{i-4}$
  **end if**
**end for**
```

where Rotate$(\cdot)$ rotates the word by 8-bits, and C is predetermined constant.

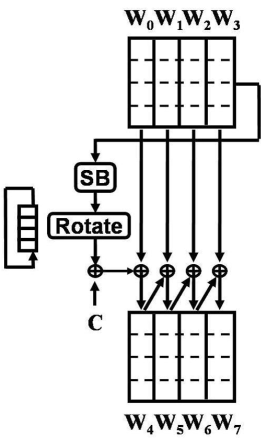


Figure 8: Key Schedule Algorithm

2.1.2 PC-MAC

PC-MAC [13] splits a message M into blocks $m_1||m_2||\cdots||m_l$. If m_l is short of a full block, it will be padded a single bit '1' and a number of of '0's.

PC-MAC iterates a structure named *Periodic CBC Hash (PCH)*, which is built by using a block cipher $E_K(\cdot)$, a set of keyed auxiliary permutations $\{G_1, G_2, \ldots, G_d\}$ (d: a positive integer) and a set of mask keys $\{K_1^{xor}, \ldots, K_{d-1}^{xor}\}$, to hash $m_1||\cdots||m_{l-1}$. Denote the output as Y. PCH is described in Figure 9.

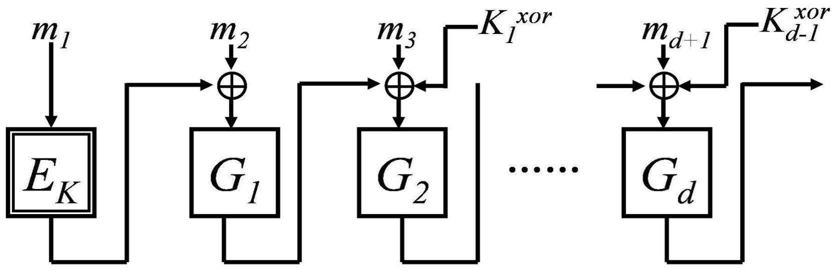


Figure 9: Periodic CBC Hash

PC-MAC uses a sub-key L to process the last block m_l. If m_l is a full block, a tag T is computed as $E_K(Y \oplus (L \bullet u) \oplus m_l)$. Otherwise, T is $E_K(Y \oplus (L \bullet u^2) \oplus (m_l||10\cdots0))$.

Note on secret keys. PC-MAC uses two independent keys K and L. $\{U_1, U_2,$

$\ldots, U_d, K_1^{xor}, \ldots, K_{d-1}^{xor}\}$ are all generated by using $E_K(\cdot)$ and L, where U_i is the secret key in G_i.

2.1.3 PC-MAC-AES

PC-MAC-AES [15] instantiates $E_K(\cdot)$ as AES with a 128-bit key and G_i as a 4-round AES permutation with independent round keys (AES-4R). G_i is described in Figure 10, which shows that U_i consists of 3 round keys, and thus is 384 bits long.

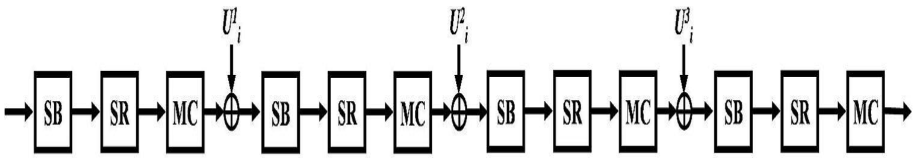


Figure 10: Auxiliary permutation G_i

U_i^j and K_i^{xor} are generated as follows:

- $U_i^j = E_K(L \oplus [3 * (i-1) + j - 1])$, and
- $K_i^{xor} = E_K(L \oplus [3d + j - 1])$,

where $[t]$ is 128-bit binary encode of integer t.

$L \bullet u$ in finite field GF(2^{128}) is detailed as below, and denoted as $mul2(L)$ in the rest of this report.

$$\text{mul2}(L) = \begin{cases} L \ll 1 & \text{if } msb(L) = 0 \\ (L \ll 1) \oplus (0^{120}||10000111) & \text{else} \end{cases}$$

A pseudo-code description of PC-MAC-AES is shown below in Algorithm 1.

Algorithm 1 PC-MAC-AES(K, L, M)

```
Pre-process:
  for $i = 1$ to $d$ do
    $U_i^1 \to E_K(L \oplus [3 * (i-1)])$
    $U_i^2 \to E_K(L \oplus [3 * (i-1) + 1])$
    $U_i^3 \to E_K(L \oplus [3 * (i-1) + 2])$
  end for
  for $i = 1$ to $d - 1$ do
    $K_i^{xor} = E_K(L \oplus [3d + i - 1])$
  end for

Generate tag:
  $m_1 || m_2 || \cdots || m_l \leftarrow M$
  if $l = 1$ then
    $h \leftarrow \mathrm{pad}(m_1)$
  else
    $s \leftarrow 0^{128}$
    for $i = 1$ to $l - 1$ do
      $w \leftarrow (i-1) \bmod (d+1)$
      if $w = 0$ then
        $s \leftarrow E_K(s \oplus m_i)$
      else if $w = 1$ then
        $s \leftarrow G_1(s \oplus m_i)$
      else
        $s \leftarrow G_i(s \oplus K_{w-1}^{xor} \oplus m_i)$
      end if
    end for
  end if
  if $m_l$ is 128 bit long then
    $h \leftarrow \mathrm{mul2}(L) \oplus h$
  else
    $h \leftarrow \mathrm{mul2}(\mathrm{mul2}(L)) \oplus h$
  end if
  $T \leftarrow E_K(h)$
  return $T$
```

2.2 Rationale

2.2.1 Hash-to-PRF to Build VIL-PRF

A well-known approach, usually named *Hash-to-PRF*, to build a variable-input-length pseudo-random function (VIL-PRF) is concatenating an Almost-Universal Hash Function (AUHF) defined below and a fixed-input-length pseudo-random function (FIL-PRF).

Definition 2.2.1.1 *Let H be a function: $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$. For two messages M and M', the colliding probability on H is*

$$Coll_H(M, M') = \frac{\Sigma_{K \in \{0,1\}^k} Pr[H(K, M) = H(K, M')]}{2^k}.$$

If $Coll_H(M, M') \leq \epsilon$ holds for any two messages M and M', we will say that H is an ϵ-almost universal hash function (ϵ-AUHF).

Loosely speaking, for a new messages inputted into AUHF, the output will be different from previous queried messages with an overwhelming probability. Then after the re-mapping of PRF, the final output will be random and uniformly distributed.

2.2.2 CBC Iteration to Build AUHF

An approach to build an AUHF is Cipher Block Chaining (CBC) iteration, which is shown in Figure 11. CBC iteration has been proven to be AUHF assuming F is a RF [4].

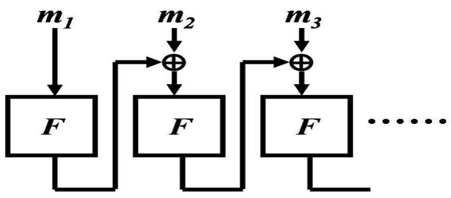


Figure 11: CBC iteration

2.2.3 Novelty Design of PC-MAC

Adopting Hash-to-PRF and CBC iteration, one can build a provable VIL-PRF, which is a secure MAC. However, the assumption on F is too strong. When such a MAC is practically implemented, one have to instantiate F to a complicated function to preserve the security confidence.

PC-MAC introduces extra mask keys to reduce the assumption on the underlying component to a weaker property *Uniformly Differential.* PC-MAC requires that auxiliary permutations G_i has a negligible Maximum Expected Differential Probability (MEDP).

Definition 2.2.3.1 *Let G be a keyed permutation: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. Maximum expected differential probability of G, denoted as $MEDP_G$ is defined as follows:*

$$MEDP_G = max_{\Delta \neq 0, \Delta'} \frac{\Sigma_{K \in \{0,1\}^k} Pr[G(K,M) \oplus G(K, M \oplus \Delta) = \Delta']}{2^k},$$

where M is uniformly sampled in $\{0,1\}^n$.

One can build a AUHF using a set of permutations $\{G_i\}$, which have small MEDP, as shown in Figure 12. Loosely speaking, with the usage of mask keys, an attacker can only control the input difference of G_i, but without the knowledge of the exact values. Since G_i has a small MEDP, the probability that an attacker will produce a collision becomes small.

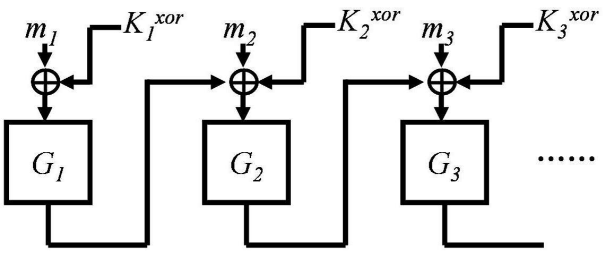


Figure 12: AUHF based on permutations with small MEDP

However, such an AUHF has a disadvantage. Numerous mask keys are necessary if the input message is long. Thus PC-MAC proposes PCH. By setting a parameter d, an execution of $E_K(\cdot)$, which is assumed to be PRF, will be involved every $d+1$ message blocks. Thanks to PRF $E_K(\cdot)$, PC-MAC can repeadly use d auxiliary permutations $\{G_i\}$, without influencing the collision resistance of AUHF.

As a summary, PC-MAC is proven to be PRF under two assumptions [13]:

- $E_K(\cdot)$ is a psuedo-random permutation; and
- $G_i(\cdot)$ has a small MEDP.

PC-MAC-AES. AES is U.S. government standard and the most widely implemented block cipher, which is a plausible instantiation of $E_K(\cdot)$. Moreover, AES-4R has been proven to be with a MEDP comparable to full AES [17, 10, 18], and thus is a proper instantiation of $G_i(\cdot)$.

3 Comparison Between PC-MAC-AES and Other Block-Cipher-Based MACs

This section will compare PC-MAC-AES with other CBC MAC variants. Particularly we derive four features.

- on-the-fly: shared with all CBC MAC variants.
- suffix: shared with EMAC and its variants.
- tweaking key: shared with XCBC and its variants.
- new features of PC-MAC-AES.

This section will make a comparison between PC-MAC-AES and other block-cipher-based MACs. More precisely, we will derive several feature of PC-MAC-AES, and pointed out which MACs share the same features. Such a comparison will contribute to a fair understanding of the weakness of PC-MAC-AES, which will be discussed in Sections $4 \sim 8$.

3.1 On-the-fly

On-the-fly structure is defined as follows:

- a message will be splited into small blocks, which will be hashed sequentially; and
- each block will be hashed only *once*.

Almost all the block-cipher-based MACs are variants of CBC MAC [2]. They all share this feature.

3.2 Suffix

Suffix is defined as follows:

- for any message $M_1||M_2$, where M_1 is a message of multiple-block length, the iteration sequence of block ciphers for processing M_2 will not change.

This feature is shared with EMAC [1] and its variants such as XCBC [4], TMAC [12] and OMAC [8].

3.3 Tweaking Key

Tweaking key is defined as follows:

- one secret key will be used to XOR the input of the last block cipher operations.

XCBC MAC [4] firstly introduced the usage of tweak keys. It uses two independent tweak keys for full-last-block and non-full last block cases separately. After that, TMAC [12] proposed a refinement to XCBC: only one tweak key L is used, and the other tweak key is computed as $L \bullet u$. PC-MAC follows the same way with TMAC to tweak the last block, except that it actually uses $L \bullet u$ and $L \bullet u^2$.

3.4 New Features

We will list two new features of PC-MAC-AES.

- G_i is AES-4R.
- U_i and K_j^{xor} are generated by $E_K(\cdot)$ and L.

Statement.

Most attacks in Sections $4 \sim 7$ have been published to attack other MACs, which were proposed earlier than PC-MAC (refer to papers [19], [5],[14],[9] etc.). We mainly try to list all the attacks which are applicable to PC-MAC-AES, and describe them in detail. Moreover, in order to contribute to a fair judgment on the meaning of these attacks, we describe the security margin loss following the development history from original CBC MAC to PC-MAC.

4 Type I of Attacks: Weakness of on-the-fly Structure

This section consists of

- distinguishing attacks
  - $\mathrm{DIS}^{\mathrm{PC-MAC-AES,RF}}$
- forgery attacks
  - existential forgery attack
  - selective forgery attack

Attacks of Type I are based on the weakness of on-the-fly structure, which are shared by CBC MAC and its variants, and thus applicable to these CBC MAC variants.

4.1 $\mathrm{DIS}^{\text{PC-MAC-AES, RF}}$ Attacks

We will describe two attack approaches, which distinguish PC-MAC-AES from RF.

4.1.1 Approach Using Internal Collisions

PC-MAC-AES has one property due to the on-the-fly structure.

- If two different messages M and M' collide on PC-MAC-AES, namely they have the same tag value. $M||m$ and $M'||m$ will also collide on it, where $||$ means concatenation and m can be any message.

However, RF does not have such a property. Thus PC-MAC can be distinguished from RF when a pair of colliding messages is found. Attack procedure is detailed below. Suppose $\mathcal{A}$ is given an oracle $\mathcal{O}(\cdot)$ of being either PC-MAC-AES or RF.

1. Initialize a table $\mathcal{T}$ as empty.
2. Select a random message M, query it to $\mathcal{O}(\cdot)$, and obtain a response T.
3. Check whether T has been included in $\mathcal{T}$.
    - Yes: derive the corresponding message/tag pair (M', T) in $\mathcal{T}$.
    - No: store (M, T) in $\mathcal{T}$ and repeat Steps 2 and 3.
4. Select a random message m.
5. Query $M||m$ to $\mathcal{O}(\cdot)$, and obtain a response T_1.
6. Query $M'||m$ to $\mathcal{O}(\cdot)$, and obtain a response T_2.
7. If $T_1 = T_2$, $\mathcal{O}(\cdot)$ is PC-MAC. Otherwise, $\mathcal{O}(\cdot)$ is RF.

Complexity. Due to the birthday attack, Steps 2 and 3 will be repeated 2^{64} times to generate a colliding pair with a success probability 0.33.

4.1.2 Approach Using Bijection of Permutation

If the domain of PC-MAC-AES is restricted to $\{0, 1\}^{128}$, it actually becomes a permutation, and thus has no colliding message pairs. However, RF does not have such a property. Thus PC-MAC-AES can be distinguished from RF when a pair of 128 bits long colliding messages is found. Attack procedure is detailed as below. Suppose $\mathcal{A}$ is given an oracle $\mathcal{O}(\cdot)$ of being either PC-MAC-AES or RF.

1. Initialize a table $\mathcal{T}$ as empty.
2. Initialize a counter C as 0.
3. Query C to $\mathcal{O}(\cdot)$, and obtain a response T.
4. Check whether T has been included in $\mathcal{T}$.
    - Yes: $\mathcal{O}(\cdot)$ is RF.
    - No: if $C \leq 2^{64}$, $C \leftarrow C + 1$, and repeat Steps 3 and 4. Otherwise, $\mathcal{O}(\cdot)$ is PC-MAC-AES.

Complexity. It needs 2^{64} queries with a success probability 0.33.

Remark. Interestingly, this attack is not highly related to *on-the-fly* structure, but can be regarded as weakness of using block ciphers.

4.2 Existential Forgery Attack

The distinguishing attack in Section 4.1.1 can be converted to an existential forgery attack. After a pair of colliding messages M and M' is generated, $\mathcal{A}$ is able to forge $M||m$ by querying $M'||m$ to PC-MAC-AES, where m can be any message. Attack procedure is detailed below.

1. Initialize a table $\mathcal{T}$ as empty.
2. Select a random message M, query it to PC-MAC-AES, and obtain a response T.
3. Check whether T is included in $\mathcal{T}$.
    - Yes: derive the corresponding message/tag pair (M', T) in $\mathcal{T}$.
    - No: store (M, T) in $\mathcal{T}$ and repeat Steps 2 and 3.
4. Select a random message m.
5. Query $M'||m$ to PC-MAC-AES, and obtain a response T_1.
6. Output a valid tag T_1 for $M||m$, which was not priorly queried.

Complexity. It needs 2^{64} queries with a success probability 0.33.

4.3 Selective Forgery Attack

Without loss of generality, suppose $\mathcal{A}$ selects a 2 blocks long message M $(= m_1||m_2)$. If a pair of colliding messages, as shown in Figure 13, are generated, $\mathcal{A}$ has $E_K(m_1) \oplus m = E_K(m') \oplus m_2$, which implies $E_K(m_1) \oplus m_2 = E_K(m) \oplus m'$. Thus $\mathcal{A}$ is able to forge $m_1||m_2$ by querying $m||m'$ to PC-MAC-AES. Attack procedure is detailed below.

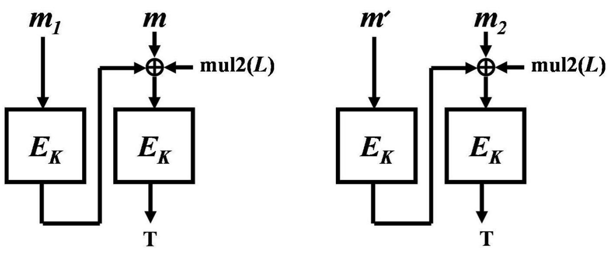


Figure 13: Selective forgery attack

1. Initialize a table $\mathcal{T}$ as empty.
2. Select 2^{64} random 128 bits long messages m $(m \neq m_2)$, query $m_1||m$ to PC-MAC-AES, and store the message/tag pairs in $\mathcal{T}$.

3. Select a random 128 bits long message m' ($m' \neq m_1$), query $m'||m_2$ to PC-MAC-AES, and obtain a response T.
4. Check whether T has been included in $\mathcal{T}$.
   - Yes: derive the message pair $(m_1||m, T)$ from $\mathcal{T}$;
   - No: repeat Steps 3 and 4.
5. Query $m||m'$ to PC-MAC-AES, and obtain a response T_1
6. Output a valid tag T_1 for $m_1||m_2$, which was not priorly queried.

Complexity. Steps 3 and 4 will be repeated 2^{64} times with a success probability 0.63. The overall complexity is 2^{65} queries.

Remark. This attack is a selective forgery attack, since $\mathcal{A}$ is able to select a message M, which is longer than 128 bits, and forge a valid tag without querying it to PC-MAC-AES. However, this attack cannot forge 128 bits long messages, and thus it is not a universal forgery attack.

5 Type II of Attacks: Weakness of Suffix Structure

This section consists of

- internal-state-recovery attack
- forgery attack
  - low complexity-on-average universal forgery attack
- full-key recovery attack

Attacks of Type II are based on the weakness of *suffix* structure, which are shared by EMAC and its variants, and thus applicable to them.

5.1 Internal-State-Recovery Attack

Without loss of generality, suppose $\mathcal{A}$ will recover the internal state right after m_1 of a message M $(= m_1||\cdots)$, which is more precisely $E_K(m_1)$. If a pair of colliding messages, as shown in Figure 14, is generated, $\mathcal{A}$ has $E_K(m_1)\oplus m = m'$, which implies $E_K(m_1) = m\oplus m'$. Thus $\mathcal{A}$ will know the value of $E_K(m_1)$, which is the internal state right after m_1. Attack procedure is detailed below.

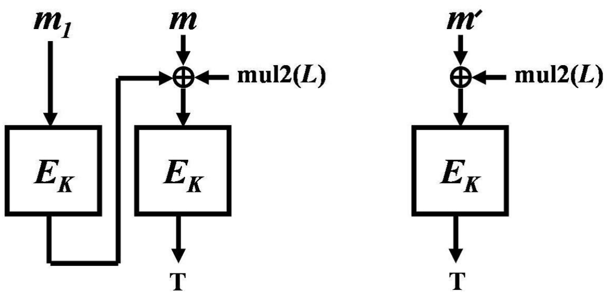


Figure 14: Internal state recovery attack

1. Initialize a table $\mathcal{T}$ as empty.
2. Select 2^{64} random 128 bits long messages m, query $m_1||m$ to PC-MAC-AES, and store the message/tag pairs in $\mathcal{T}$.
3. Select a random 128 bits long message m', query m' to PC-MAC-AES, and obtain a response T.

4. Check whether T is included in $\mathcal{T}$.
    - Yes: derive the message pair $(m_1||m, T)$ from $\mathcal{T}$;
    - No: repeat Steps 3 and 4.
5. Output $m \oplus m'$ as the value of $E_K(m_1)$.

Complexity. Steps 3 and 4 will be repeated 2^{64} times with a success probability 0.63. The overall complexity is 2^{65} queries.

5.2 Low Complexity-on-Average Universal Forgery Attack

Without loss of generality, suppose $\mathcal{A}$ is going to forge a one-block message m. Let M $(= m_1|| \cdots ||m_{d+1})$ be a random $(d+1)$-block message. As shown in Figure 15, $M||(\text{PCH}(M) \oplus m)$ and m are a collision on PC-MAC-AES. Thus $\mathcal{A}$ can forge a valid tag for m by querying $M||(\text{PCH}(M) \oplus m)$. Attack procedure is detailed below.

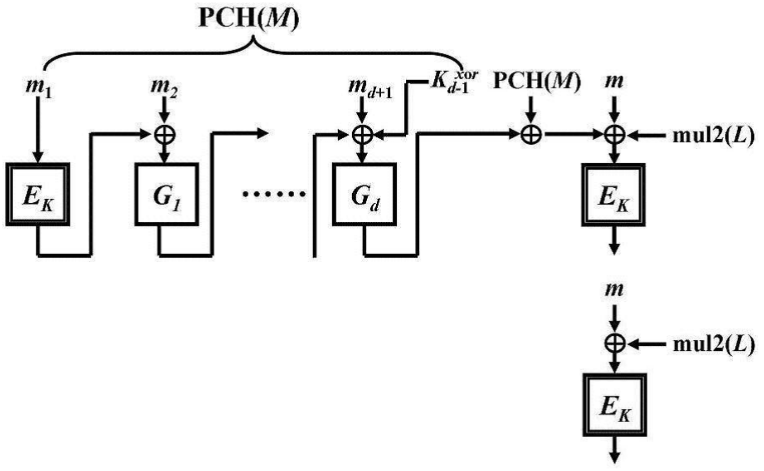


Figure 15: Universal Forgery Attack

1. Select a random $(d+1)$-block message M, and another random message M'.
2. Recover the internal state right after M of PC-MAC-AES$(M||M')$, which is PCH(M). Refer to Section 5.1 for the detailed procedure.
3. Query $M||(\text{PCH}(M) \oplus m)$ to PC-MAC-AES, and obtain a response T.
4. Output a valid tag T for m, which was not priorly queried.

Complexity. The dominant complexity is from Step 2. Thus $\mathcal{A}$ makes 2^{65} queries with a success probability 0.63.

Complexity on average. After PCH(M) is recovered, $\mathcal{A}$ only needs 1 query to forge a message. Thus $\mathcal{A}$ is able to forge a heavy number of message with only a negligible increasement of complexity, which will make the average complexity low.

5.3 Key Recovery Attack

PC-MAC-AES has two 128-bit keys K and L. Thus a trivial attack will need 2^{256} computations to recover both K and L. However, it only needs around 2^{128} computations to recover them. There are two attacks based on the weakness of the *suffix* iteration.

Remark. Besides *suffix*, these two attacks seem to also take advantage that EMAC and its variants only use the second secret key to remodel the last block operation of original CBC MAC.

5.3.1 Approach Using Internal Collisions

When a pair of colliding messages $m_1||m_2$ and $m'_1||m'_2$, which are 2 blocks long, is generated, $\mathcal{A}$ has $E_K(m_1) \oplus m_2 = E_K(m'_1) \oplus m'_2$. Thus $\mathcal{A}$ is able to exhaustively recover K without the knowledge of L. After K is recovered, $\mathcal{A}$ will further exhaustively recover L using 1 message/tag pair. Attack procedure is detailed below.

1. Generate a pair of 2-block long colliding messages $m_1||m_2$ and $m'_1||m'_2$ on PC-MAC-AES. Refer to Section 4.1.1 for the detailed procedure.
2. Set a counter C as 0^{128}.
3. Check whether $\text{AES}_C(m_1) \oplus m_2 = \text{AES}_C(m'_1) \oplus m'_2$ holds.
   - Yes: $K = C$.
   - No: $C \leftarrow C + 1$. Repeat Step 3.
4. Query a one-block message m to PC-MAC-AES, and obtain a response T.
5. Set C as 0^{128}.
6. Compute PC-MAC-AES(m) using recovered K and C as L, and check whether it will be equal to T.
   - Yes: $L = C$.
   - No: $C \leftarrow C + 1$. Repeat Step 6.

7. Output K and L.

Complexity. Step 1 needs 2^{65} queries with a success probability 0.63. Step 3 will be repeated 2^{128} times in the worst case. Similarly Step 6 will be repeated 2^{128} times in the worst case. Finally the overall complexity is 2^{65} online queries and 2^{129} offline computations.

5.3.2 Approach Using Internal-State-Recovery Attacks

When the internal state right after m_1 of 2 blocks long message $m_1||m_2$ is recovered, $\mathcal{A}$ is able to exhaustively recover K, and then exhaustively recover L. Attack procedure is detailed below.

1. Select a random 2 blocks long message $m_1||m_2$, and recover the value of $E_K(m_1)$. The procedure is detailed in Section 5.1.
2. Set C as 0^{128}.
3. Check whether $E_C(m_1) = E_K(m_1)$.
   - Yes: $K = c$
   - No: $C \to C + 1$ and repeat Step 3.
4. Exhaustively recover L. Refer to the attack procedure (Steps 5 and 6) in Section 5.3.1.
5. Output K and L.

Complexity. Step 1 needs 2^{65} online queries. Step 3 will be repeated 2^{128} times. Refer to Section 5.3.1, it needs 2^{128} computations to recover L. Overall, it needs 2^{65} queries and 2^{129} computations.

6 Type III of Attacks: Weakness of Tweaking Keys

This section consists of

- partial-key-recovery attack
- forgery attacks
  - (new) universal forgery attack
- low complexity-on-average internal-state-recovery attack
- (better) full-key recovery attack

Attacks of Type III are based on the weakness of using a secret key to tweak the last block, which are shared by TMAC and its variants, and thus applicable to them.

6.1 Partial-Key-Recovery Attack

When a pair of colliding messages, as shown in Figure 16, is generated, $\mathcal{A}$ has $m \oplus \text{mul2}(L) = (m'||10\cdots0) \oplus \text{mul2}(\text{mul2}(L))$, where m is 128 bits long, but m' is short of 128 bits long. Denote mul2(L) by X, and then $X \oplus \text{mul2}(X) = m \oplus (m'||10\cdots0)$ holds. $\mathcal{A}$ will get $X = (m \oplus (m'||10\cdots0) \bullet (u+1)^{-1}$, and further $L = (m \oplus (m'||1\cdots0)) \bullet (u+1)^{-2}$. The attack procedure is detailed below.

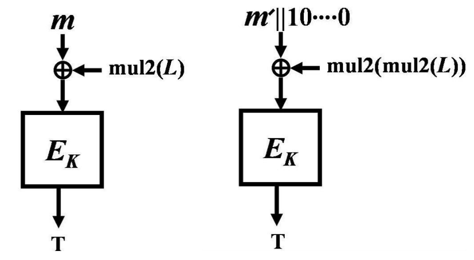


Figure 16: Partial-key Recovery Attack

1. Initialize a table $\mathcal{T}$ as empty.
2. Select 2^{64} random 128-bit messages, query them to PC-MAC-AES, and store the message/tag pairs in $\mathcal{T}$.

3. Select a random message m', which is short of 128 bits, query it to PC-MAC-AES, and obtain a response T.

4. Check whether T has been included in $\mathcal{T}$.

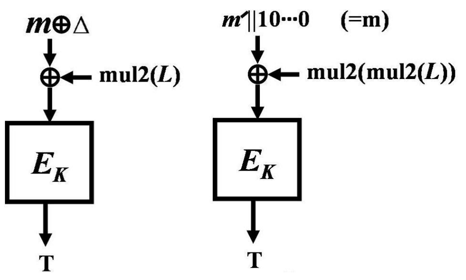
   - Yes: derive the message/tag (m, T) in $\mathcal{T}$.
   - No: repeat Steps 3 and 4.

5. Output $L = (m \oplus (m'||1 \cdots 0)) \bullet (u+1)^{-2}$.

Complexity. Steps 3 and 4 will be repeated 2^{64} times with a success probability 0.63. Th overall complexity is 2^{65} queries with a success probability 0.63.

6.2 Universal Forgery Attack

After $\mathcal{A}$ has the knowledge of L, universal forgery attack becomes trivial. Without loss of generality, suppose $\mathcal{A}$ will forge a non-full one block long message m', more precisely short of 128 bits. As shown in Figure 17, $m \oplus \Delta$ and m' is a pair of colliding messages on PC-MAC-AES. Thus $\mathcal{A}$ can forge m' by querying $m \oplus \Delta$ to PC-MAC-AES. Attack procedure is detailed below.



Δ means mul2(L) $\oplus$ mul2(mul2(L)), and $m = m'||10 \cdots 0$.

Figure 17: Universal Forgery Attack

1. Recover L. The procedure is detailed in Section 6.1.
2. Compute Δ as mul2(L) $\oplus$ mul2(mul2(L)).
3. Query $m \oplus \Delta$ to PC-MAC-AES, and obtain a response T.
4. Output T as a valid tag for m', which was not priorly queried.

Complexity. The dominant complexity is from Step 1. The complexity is 2^{65} queries with a success probability 0.63.

6.3 Low Complexity-on-Average Internal-State-Recovery Attack

After L is recovered, $\mathcal{A}$ can obtain internal state of a message trivially. Without loss of generality, suppose $\mathcal{A}$ will recover the internal state after m_1 for a 2 blocks long message $m_1||m_2$. As shown in Figure 18, $\mathcal{A}$ can query $m_1 \oplus \text{mul2}(L)$ to PC-MAC-AES, which is actually $E_K(m_1)$. Thus $\mathcal{A}$ has recovered the internal state after m_1 for $m_1||m_2$. Attack procedure is detailed below.

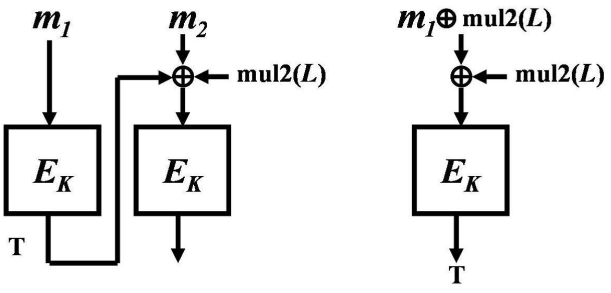


Figure 18: Internal State Recovery Attack

1. Recover L. The procedure is detailed in Section 6.1.
2. Query $m_1 \oplus \text{mul2}(L)$, and obtain a response T.
3. Output T as the internal state $E_K(m_1)$.

Complexity. The dominant complexity is from Step 1. The overall complexity is 2^{65} queries with a success probability 0.63.

Complexity on average. After L is recovered, $\mathcal{A}$ only needs 1 query to obtain an internal-state value of a message. Thus $\mathcal{A}$ is able to recover internal-state values of a heavy number of messages with only a negligible increasement of complexity, which makes the average complexity low.

6.4 Full Key Recovery Attack

L can be trivially computed after K is recovered. Thus there is better time/memory tradeoff compared with full-key-recovery attack of Type I.

6.4.1 Approach of Saving Memory

Guessing the value of K, $\mathcal{A}$ can compute the corresponding value of L by using one message/tag pair, and verify the correctness of guess by using another message/tag pair. By exhaustively guessing K, $\mathcal{A}$ will recover both K and L. Throughout the attack, $\mathcal{A}$ only needs to memorize two message/tag pairs. Attack procedure is detailed below.

1. Select two random 128 bits long messages m and m', query them to PC-MAC-AES, and obtain responses T and T' respectively.
2. Set a counter C as 0^{128}.
3. Compute $(E_C^{-1}(T) \oplus m) \bullet u^{-1}$.
4. Check whether $T' = E_C(m' \oplus (E_C^{-1}(T) \oplus m))$ holds.
    - Yes. $K = C$, and $L = (E_C^{-1}(T) \oplus m) \bullet u^{-1}$.
    - No. $C \leftarrow C + 1$, and repeat Steps 3 and 4.
5. Output K and L.

Complexity. Steps 3 and 4 will be repeated 2^{128} times in the worst case, and each times needs 2 computations. Overall, the complexity is 2^{129}. Interestingly, $\mathcal{A}$ only needs to memorize 2 message/tag pairs.

6.4.2 Approach of Saving Time

$\mathcal{A}$ will follow the procedures in Section 5.3 to recover K first, and then compute the value of L trivially. Attack procedure is detailed below.

1. Recover K using internal-state-recovery attacks. Refer to Section 5.3.2 for detailed procedure.
2. Compute L as $(E_K^{-1}(T) \oplus m) \bullet u^{-1}$, where (m, T) is a message/tag pair of PC-MAC-AES.
3. Output K and L.

Complexity. The complexity is reduced to 2^{128}. $\mathcal{A}$ has to memorize 2^{65} message/tag pairs.

7 Attack of Type IV: Weakness of Subkey Generation Algorithm

This section consists of

- Subkey-recovery attack
- DIS-PC-MAC$^{\text{AES, RP}}$ attack

Attacks of Type IV will discuss the weakness introduced by new feature of PC-MAC-AES.

7.1 Subkey-Recovery Attack

After L is recovered, $\mathcal{A}$ can recover $\{U_i\}$ and $\{K_i^{xor}\}$ trivially. Here pick U_1 as an example. U_1 $(= U_1^1||U_1^2||U_1^3)$ is generated as follows: $U_1^1 = E_K(L)$; $U_1^2 = E_K(L \oplus [1])$, and $U_1^3 = E_K(L \oplus [2])$ respectively. $\mathcal{A}$ can query a one-block message $L \oplus \text{mul2}(L)$ to PC-MAC-AES, which is shown in Figure 19, to obtain the value of U_1^1. Similarly $\mathcal{A}$ will recover U_1^2 and U_1^3. Thus U_1 is recovered. Attack procedure is detailed below.

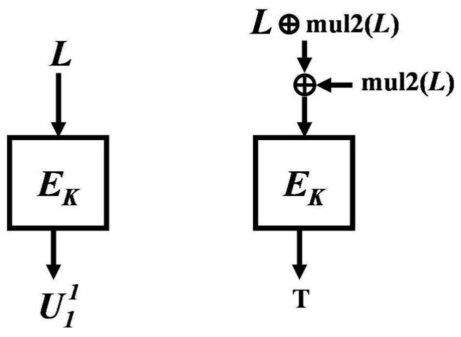


Figure 19: Recover U_1^1

1. Recover L. The procedure is detailed in Section 6.1.
2. Set i as 1.
3. Recover U_i as follows.
   (a) Query $L \oplus [3 * (i-1)] \oplus \text{mul2}(L)$ to PC-MAC-AES, and obtain a response as U_i^1.

   (b) Query $L \oplus [3 * (i-1) + 1] \oplus \mathrm{mul2}(L)$ to PC-MAC-AES, and obtain a response as U_i^2.

   (c) Query $L \oplus [3 * (i-1) + 2] \oplus \mathrm{mul2}(L)$ to PC-MAC-AES, and obtain a response as U_i^3.

4. Set $i \leftarrow i + 1$. If $i \leq d$, repeat Steps 3 and 4.

5. Set j as 1.

6. Query $L \oplus [3*(d-1)+j] \oplus \mathrm{mul2}(L)$ to PC-MAC-AES to obtain a response as K_j^{xor}.

7. Set $j \leftarrow j + 1$. If $j \leq d - 1$, repeat Steps 6 and 7.

8. Output $\{U_i\}$ and $\{K_i^{xor}\}$.

Complexity. The dominant complexity is from recovering L. The overall complexity is 2^{65} queries.

7.2 DIS-PC-MAC$^{\textbf{AES, RP}}$ Attacks

$\mathcal{A}$ is able to trivially distinguish PC-MAC-AES from PC-MAC-RP after recovering L, $\{U_i\}$ and $\{K_i^{xor}\}$. Since $\mathcal{A}$ knows the value of L, she will be able to recover internal state $E_K(m)$ for a 2 blocks long message $m||m'$ easily. With the knowledge of $\{U_1\}$ and $\{K_i^{xor}\}$, $\mathcal{A}$ is able to generate a colliding message pair on PC-MAC-AES. However, PC-MAC-RP does not have such a property. Thus $\mathcal{A}$ will be able to distinguish PC-MAC-AES from PC-MAC-RP. Attack procedure is detailed below. Denote the given oracle as $\mathcal{O}(\cdot)$, which is either being PC-MAC-AES or PC-MAC-RP.

1. Suppose $\mathcal{O}(\cdot)$ is PC-MAC-AES. Recover L, $\{U_i\}$ and $\{K_i^{xor}\}$. Procedure is detailed in Sections 6.1 and 7.1.

2. Select a random 128 bits message m_1, query $m_1 \oplus L$ to $\mathcal{O}(\cdot)$, and obtain a response t.

3. Select a random 128-bit message m_2, and compute $y = G_1(m_2 \oplus t)$.

4. Select another random 128-bit message m_2' ($m_2' \neq m_2$), and compute $y' = G_1(m_2' \oplus tT)$.

5. Select a random 128-bit message m_3, and compute $m_3' = m_3 \oplus y \oplus y'$.

6. Query $m_1||m_2||m_3$ and $m_1||m_2'||m_3'$ to $\mathcal{O}(\cdot)$, and obtain responses T and T'.

7. Check where $T = T'$ holds.

   - Yes: $\mathcal{O}(\cdot)$ is PC-MAC-AES.

- No: $\mathcal{O}(\cdot)$ is PC-MAC-RP.

Complexity. The dominant complexity is from recovering L. Thus the overall complexity is 2^{65} queries.

8 Attacks of Type V: Weakness of AES-4R

This section consists of

- a subkey-recovery attack on a CBC MAC variant using AES-4R
- an internal-state-recovery attack

Attacks of Type V is to deal with an interesting question:

Will using AES-4R weaken the security of a CBC MAC variant from any sense?

Our answer is *yes*. We will propose a subkey recovery attack on a CBC MAC variant using AES-4R. AES-4R is detailed in Figure 20. More precisely we will recover the subkeys K_2, K_3 and K_4.

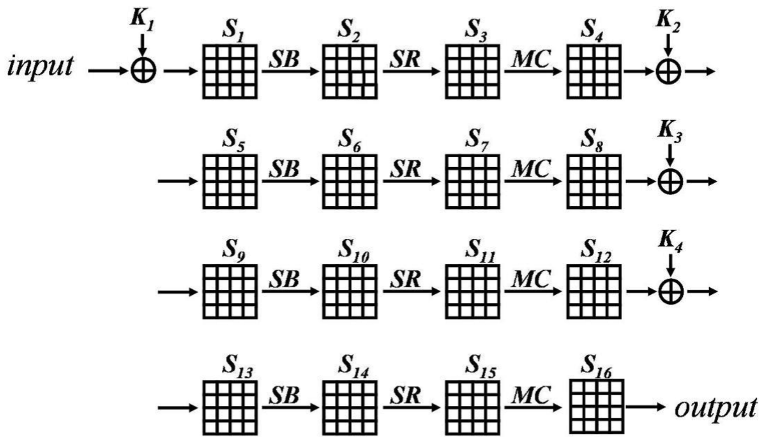


Figure 20: AES-4R

8.1 Minimum Available Information

All CBC MACs share a common character on-the-fly structure. Suppose a CBC MAC will use AES-4R to process i-th block of input messages, which is shown in Figure 21.

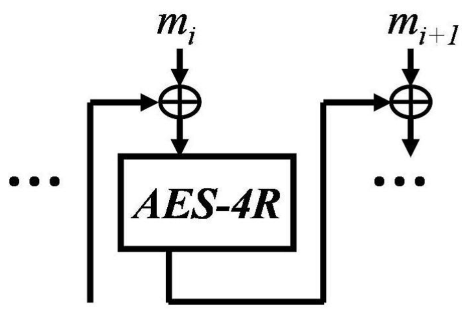


Figure 21: AES-4R to process i-th block

We will show how to obtain the output difference of AES-4R for a chosen input difference with a complexity of 2^{65} queries. The main idea is to generate a pair of colliding messages, which only differs at i-th and $(i+1)$-th blocks, and difference of i-th block is equal to the chosen input difference of AES-4R. We can get that the corresponding output difference of AES-4R should be equal to the difference of $(i+1)$-th blocks. Attack procedure is detailed belwow. Suppose the chosen input difference of AES-4R is Δ.

1. Initialize a table $\mathcal{T}$ as empty.
2. Select i random blocks m_1, …, m_i.
3. Compute $m'_i = m_i \oplus \Delta$.
4. Select 2^{64} random block m, and query $m_1||m_2||\cdots||m_{i-1}||m_i||m$ to MAC, and obtain corresponding tags. Store the message/tag pairs in $\mathcal{T}$.
5. Select a random block m', and query $m_1||m_2||\cdots||m_{i-1}||m'_i||m'$ to MAC, and obtain a corresponding tag T.
6. Check whether T has been included in $\mathcal{T}$.
    - Yes: derive the message/tag pair $(m_1||m_2||\cdots||m_{i-1}||m_i||m, T)$ from $\mathcal{T}$.
    - No: repeat Steps 5 and 6.
7. Output $m \oplus m'$ as the corresponding output difference of AES-4R.

Complexity. Steps 5 and 6 will be repeated 2^{64} times with a success probability 0.63. The overall complexity is 2^{65} queries with a success probability 0.63.

8.2 Subkey-Recovery Attacks

With the knowledge of output difference for a chosen input difference on AES-4R, we will show how to recover subkeys.

8.2.1 Useful Properties of AES

We will list several properties of AES which will be used in our attacks.

Property 1. The branch number of MC is 5.

Property 2. Any 4 byte differences of input and output of MC will uniquely determine the remaining 4 byte differences.

Property 3. The following 2 round differential path in Figure 22 is with a probability 1, where byte in white color has no difference and byte in grey color has a non-zero difference.

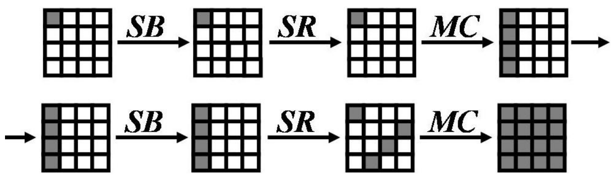


Figure 22: 2-Round Differential Path with a Probability 1

Hereafter, we will refer non-active byte to as byte with a zero difference, and active byte to as byte with a non-zero difference.

Property 4. For the 2-round differential path in Figure 22, the output byte differences after 2nd MC essentially only depend on bytes 0, 5, 10 and 15 of the pair inputs.

Proof. Let (x_1, x_1') and (x_2, x_2') be two pairs of inputs satisfying

- each pair differs at byte 0;
- the differences at byte 0 are equal for the two pairs;
- bytes 0, 5, 10 and 15 are equal for x_1 and x_2.

From the above conditions, we can trivially get that bytes 0, 7, 10 and 13 of internal state after 2nd SR are still equal for x_1 and x_2. The same situation happens for x_2 and x_2'. Thus we get that the byte differences after 2nd SR are equal for the two pairs. Due to linearity of MC transformation, the byte differences after 2nd MC will also be equal for the two pairs. Therefore Property 4 has been proven.

Property 5. for a pair of input difference Δ^i and output difference Δ^o, 2 corresponding values X satisfying $SB(X) \oplus SB(X \oplus \Delta^i) = \Delta^o$ can be derived with a probability of nearly 0.5 by looking up the differential distribution table of S-box.

8.2.2 How to Recover K_4

Our attack is based on a 4-round differential path in Figure 23, which has a probability 1 following Property 3. The main novelty of our attack is as follows.

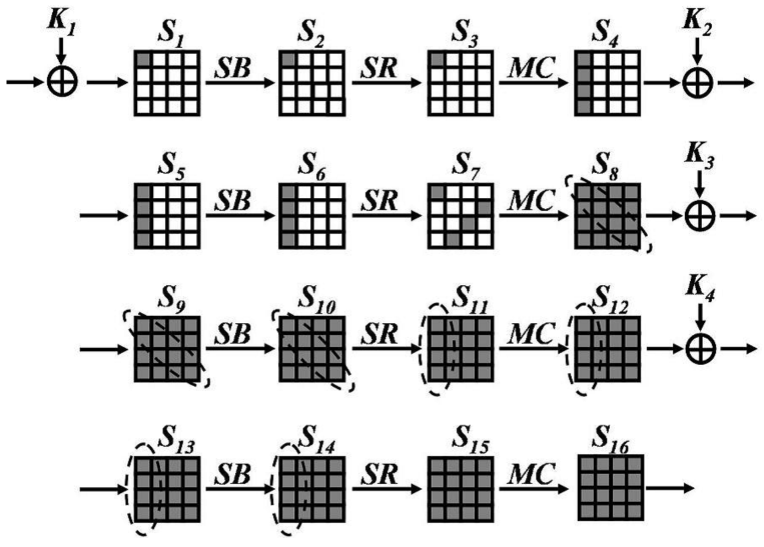


Figure 23: 4-Round Differential Path Used in Our Attacks

Let (m_1, m'_1), (m_2, m'_2) and (m_3, m'_3) be three pairs satisfying the following three conditions.

- each pair differs at byte 0; Thus the differential propagation for each pair following the path in Figure 23.
- the difference at byte 0 are equal for the three pairs;
- the values of bytes 0, 5, 10 and 15 are equal for m_1, m_2 and m_3.

From Property 4, these pairs share the same byte differences at State S_8.

We extend (m_1, m'_1), (m_2, m'_2) and (m_3, m'_3) to five pairs $\{(m_1, m'_1), (m_2, m'_2), (m_3, m'_3), (m_1, m_2), (m_1, m_3)\}$, and obtain the byte difference at State S_{16} for each pair.

Then we will compute the byte difference at State S_{14} for these five pairs due to the linearity of $SR^{-1} \circ MC^{-1}(\cdot)$.

Finally we will recover State S_{14} and k_4 column by column. Following the dotted computation in Figure 23, by guess one column of S_{14} and K_4, we will be able to compute differences of byte $0, 5, 10$ and 15 of State S_9 for three pairs (m_1, m'_1), (m_2, m'_2) and (m_3, m'_3), which should be equal. For a false guess, the probability of satisfying the property is 2^{-64}. In total, there are 2^{64} freedom.

Therefore only the correct value will be left. Similarly we can recover the other columns. Thus K_4 will be recovered.

Moreover, we use two ideas to reduce the complexity.

Idea I. We do not need to guess a column of K_4. First we will compute the byte difference of a diagonal at State S_{10} by guessing one column of S_{14}. Then we determine the value of each byte independently. Finally we can compute the value of the column of K_4. In such a manner, the complexity becomes 2^{40} from 2^{64}.

Idea II. After one column is recovered, actually the byte difference at State 9 has been uniquely determined. Thus it will be much easier to recover the following three columns.

8.2.3 Detailed Procedure of Recovering K_4

Notations. (m_1, m_1'), (m_2, m_2') and (m_3, m_3') are three pairs following the differential path in Figure 23. Moreover, m_1, m_2 and m_3 share the same values of bytes 0, 5, 10, and 15. Denote States of m_1, m_1', m_2, m_2', m_3 and m_3' as $S^1_{1\sim16}$, $S^{1'}_{1\sim16}$, $S^2_{1\sim16}$, $S^{2'}_{1\sim16}$, $S^3_{1\sim16}$ and $S^{3'}_{1\sim16}$ respectively. Denote Byte j of State S^t_i as $S^t_{i,j}$. Denote State difference of pairs (m_1, m_1'), (m_2, m_2'), (m_3, m_3'), (m_1, m_2), (m_1, m_3) as $\Delta S^{1,1}_{1\sim16}$, $\Delta S^{2,2}_{1\sim16}$, $\Delta S^{3,3}_{1\sim16}$, $\Delta S^{1,2}_{1\sim16}$ and $\Delta S^{1,3}_{1\sim16}$ respectively. Denote difference of byte j in $\Delta S^{t_1,t_2}_i$ as $\Delta S^{t_1,t_2}_{i,j}$.

Attack Procedure.

1. Select three message pairs (m_1, m_1'), (m_2, m_2') and (m_3, m_3') satisfying the conditions detailed in Section 8.2.2.
2. Obtain $\Delta S^{1,1}_{16}$, $\Delta S^{2,2}_{16}$, $\Delta S^{3,3}_{16}$, $\Delta S^{1,2}_{16}$ and $\Delta S^{1,3}_{16}$. The procedure is detailed in Section 8.1.
3. Inversely compute $\Delta S^{1,1}_{14}$, $\Delta S^{2,2}_{14}$, $\Delta S^{3,3}_{14}$, $\Delta S^{1,2}_{14}$ and $\Delta S^{1,3}_{14}$.
4. Guess the values of bytes $S^1_{14,0}$, $S^1_{14,4}$, $S^1_{14,8}$ and $S^1_{14,12}$.
5. Compute the values of bytes $S^1_{13,0}$, $S^1_{13,4}$, $S^1_{13,8}$ and $S^1_{13,12}$.
6. Compute the values of bytes $S^{1'}_{14,0}$, $S^{1'}_{14,4}$, $S^{1'}_{14,8}$ and $S^{1'}_{14,12}$ since $\Delta S^{1,1}_{14}$ is known.
7. Compute the values of bytes $S^{1'}_{13,0}$, $S^{1'}_{13,4}$, $S^{1'}_{13,8}$ and $S^{1'}_{13,12}$.
8. Compute $\Delta S^{1,1}_{13,0}$, $\Delta S^{1,1}_{13,4}$, $\Delta S^{1,1}_{13,8}$ and $\Delta S^{1,1}_{13,12}$.
9. Compute $\Delta S^{1,1}_{10,0}$, $\Delta S^{1,1}_{10,5}$, $\Delta S^{1,1}_{10,10}$ and $\Delta S^{1,1}_{10,15}$ due to the linearity of SR, MC and ARK.
10. Compute the values of bytes $S^2_{14,0}$, $S^2_{14,4}$, $S^2_{14,8}$ and $S^2_{14,12}$ since $\Delta S^{1,2}_{14}$ is known.

11. Compute the values of bytes $S^{2'}_{14,0}$, $S^{2'}_{14,4}$, $S^{2'}_{14,8}$ and $S^{2'}_{14,12}$ since $\Delta S^{2,2}_{14}$ is known.

12. Compute $\Delta S^{2,2}_{10,0}$, $\Delta S^{2,2}_{10,5}$, $\Delta S^{2,2}_{10,10}$ and $\Delta S^{2,2}_{10,15}$, similarly with Steps $5 \sim 9$.

13. Compute $\Delta S^{1,2}_{10,0}$, $\Delta S^{1,2}_{10,5}$, $\Delta S^{1,2}_{10,10}$ and $\Delta S^{1,2}_{10,15}$.

14. Compute the values of bytes $S^{3}_{14,0}$, $S^{3}_{14,4}$, $S^{3}_{14,8}$ and $S^{3}_{14,12}$ since $\Delta S^{1,3}_{14}$ is known.

15. Compute the values of bytes $S^{3'}_{14,0}$, $S^{3'}_{14,4}$, $S^{3'}_{14,8}$ and $S^{3'}_{14,12}$ since $\Delta S^{3,3}_{14}$ is known.

16. Compute $\Delta S^{3,3}_{10,0}$, $\Delta S^{3,3}_{10,5}$, $\Delta S^{3,3}_{10,10}$ and $\Delta S^{3,3}_{10,15}$.

17. Compute $\Delta S^{1,3}_{10,0}$, $\Delta S^{1,3}_{10,5}$, $\Delta S^{1,3}_{10,10}$ and $\Delta S^{1,3}_{10,15}$.

18. Guess the value of byte $S^{1}_{10,0}$.

19. Compute the values of bytes $S^{1'}_{10,0}$, $S^{2}_{10,0}$, $S^{2'}_{10,0}$, $S^{3}_{10,0}$ and $S^{3'}_{10,0}$.

20. Compute the values of bytes $S^{1}_{9,0}$, $S^{1'}_{9,0}$, $S^{2}_{9,0}$, $S^{2'}_{9,0}$, $S^{3}_{9,0}$ and $S^{3'}_{9,0}$.

21. Check whether $\Delta S^{1,1}_{9,0} = \Delta S^{2,2}_{9,0} = \Delta S^{3,3}_{9,0}$ holds.

    - Yes: the guess of $S^{1}_{10,0}$ at Step 18 is correct.
    - No: the guess of $S^{1}_{10,0}$ at Step 18 is wrong.
      If every candidate value of $S^{1}_{10,0}$ has been tried, the guess of the column of S^{1}_{14} is wrong. Guess next candidate value at Step 4 and repeat Steps $4 \sim 21$.
      Otherwise, guess next candidate value at Step 18, and repeat Steps $19 \sim 21$.

22. Recover the values of bytes $S^{1}_{10,5}$, $S^{1}_{10,10}$ and $S^{1}_{10,15}$ similarly with Step $18 \sim 21$, which also determines the values of bytes $S^{1}_{14,0}$, $S^{1}_{14,4}$, $S^{1}_{14,8}$ and $S^{1}_{14,12}$.

23. Recover bytes 0, 4, 8 and 12 of K_4.

24. Compute the values of bytes $S^{1}_{9,0}$, $S^{1}_{9,5}$, $S^{1}_{9,10}$ and $S^{1}_{9,15}$.

25. Compute the values of these bytes of $S^{1'}_{9}$.

26. Compute the values of $\Delta S^{1,1}_{9,0}$, $\Delta S^{1,1}_{9,5}$, $\Delta S^{1,1}_{9,10}$ and $\Delta S^{1,1}_{9,15}$. Then $\Delta S^{1,1}_{8,0}$, $\Delta S^{1,1}_{8,5}$, $\Delta S^{1,1}_{8,10}$ and $\Delta S^{1,1}_{8,15}$ are known.

27. Determine the values of $\Delta S_8^{1,1}$ ($=\Delta S_9^{1,1}$).

    This is because of Property 2. For the difference propagation $\Delta S_7^{1,1} \xrightarrow{\text{MC}} S_8^{1,1}$ in Figure 23, 4 byte differences of each column has been determined, and thus the remaining four byte difference can be computed. For example, for the first column, $\Delta S_{7,4}^{1,1}$, $\Delta S_{7,8}^{1,1}$, $\Delta S_{7,12}^{1,1}$ and $\Delta S_{8,0}^{1,1}$ are determined.

28. Recover the values of the remaining bytes of S_{14}^1 and K_4 similarly with Steps 4 $\sim$ 23.

    Note that the values of $\Delta S_9^{1,1}$ has been determined at Step 27. Thus when recover the other bytes of S_{10}^1, there is no need to carry out guess-the-verify approach (Steps 18 $\sim$ 21). Instead, it will be recovered by looking up the differential distribution table of S-box. This will save time complexity.

29. Output the values of K_4 and S_{14}^1.

Complexity. At Step 2, less than 2^{67} queries is necessary. The memory requirement is less than 2^{66}. At Steps 4 $\sim$ 23, the time complexity is around 2^{40}. At Steps 28, the time complexity is around 2^{32}. Overall, the complexity of recovering K_4 is 2^{67} online queries and 2^{40} offline computations. The memory requirement is 2^{66}.

Remark. This attack is also an internal-state-recovery attack.

8.2.4 How to Recover K_3

After K_4 is recovered, we will recover K_3 using two differential paths in Figures 24 and 25. The main novelty of our attacks is as follows.

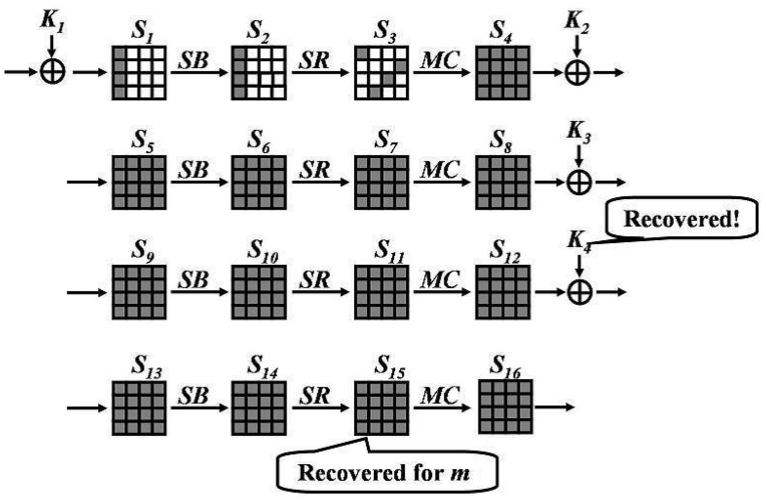


Figure 24: First Differential path for recovering K_3

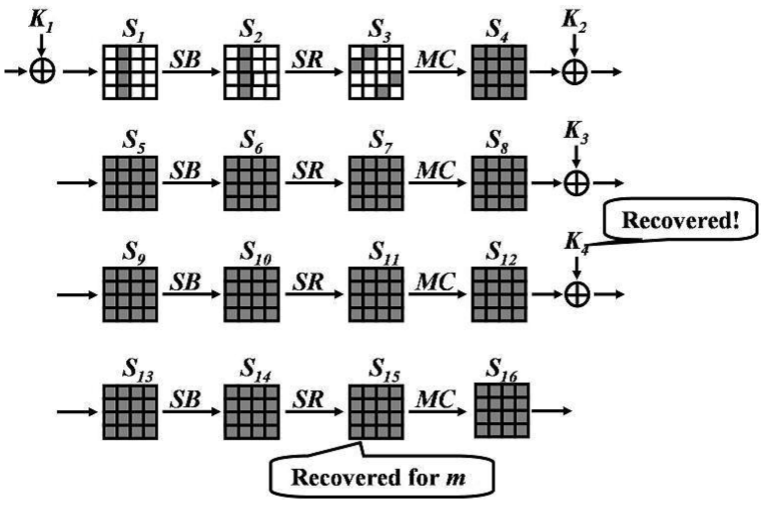


Figure 25: Second Differential path for recovering K_3

Let m be a message used for recovering K_4. We know that during recovering K_4, the internal states $S_{9\sim 16}$ of m are also recovered. Select two pairs (m, m') and (m, m'') satisfying the input differences of differential paths in Figures 24 and 25 respectively. For the pair (m, m'), first recover the output difference following the procedure in Section 8.1, which also determine the difference at State S_6. Then guess the value of bytes $S_{1,0}$, $S_{1,4}$, $S_{1,8}$ and $S_{1,12}$ of m, which will determine the difference at State S_5. By checking the input and output differences of S-box during $S_5 \xrightarrow{SB} S_6$, 2^{16} candidate values for S_5 and S_6 of m will be left with a probability 2^{-16}. In total, by exhaustively guessing bytes $S_{1,0}$, $S_{1,4}$, $S_{1,8}$ and $S_{1,12}$ of m, 2^{32} candidate values for S_5 and S_6 of m will be left. Each candidate value will contribute to a candidate value of K_3. Finally we carry out similar procedure for the other pair (m, m''), and obtain another 2^{32} candidate values of K_3. The colliding value will be the correct value of K_3. Moreover, we have also recovered the values of $S_{1,0}$, $S_{1,1}$, $S_{1,4}$, $S_{1,5}$, $S_{1,8}$, $S_{1,9}$, $S_{1,12}$ and $S_{1,13}$ of m.

Comlexity. It needs 2^{66} online queries, and 2^{32} computations. The memory requirement is 2^{64}.

8.2.5 How to Recover K_2

After K_3 and K_4 are recovered, we will recover K_2 using the differential path in Figure 26. The main novelty is as follows.

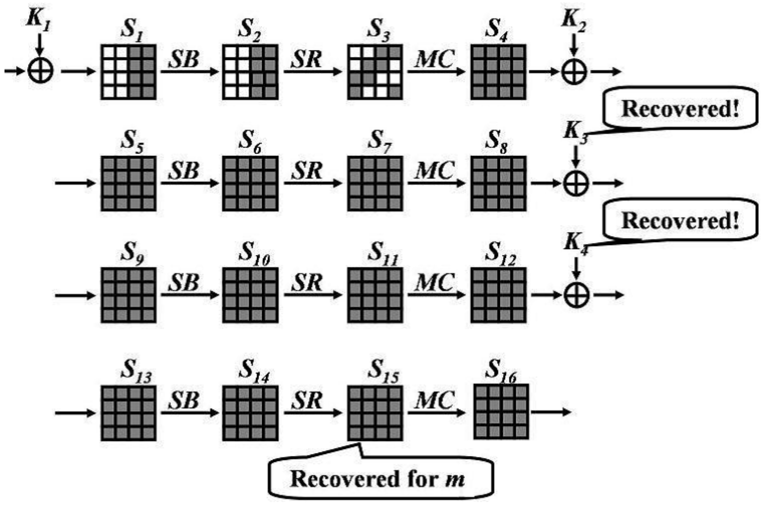

K_1
S_1
SB
S_2
SR
S_3
MC
S_4
K_2
Recovered!
K_3
S_5 SB S_6 SR S_7 MC S_8
Recovered!
K_4
S_9 SB S_{10} SR S_{11} MC S_{12}
S_{13} SB S_{14} SR S_{15} MC S_{16}
Recovered for m

Figure 26: Differential path for recovering K_2

Let m be a message used for recovering K_4 or K_3. The internal state $S_{5\sim16}$ for m has been recovered. Moreover, bytes $S_{1,0}$, $S_{1,1}$, $S_{1,4}$, $S_{1,5}$, $S_{1,8}$, $S_{1,9}$, $S_{1,12}$ and $S_{1,13}$ for m have been recovered. Select two message pairs (m, m') and (m, m'') satisfying the input difference of the differential path in Figure 26. For the pair (m, m'), first recover the output difference following the procedure in Section 8.1, which also determine the difference at State S_2. By checking the input and output differences of S-box during $S_1 \xrightarrow{SB} S_2$, 2^8 candidate values for S_1 and S_2 of m will be left, and each candidate value will contribute to a candidate value of K_2. Finally we carry out similar procedure for the other pair (m, m''), and obtain another 2^8 candidate values of K_2. The colliding value will be the correct value of K_2.

Complexity. It needs 2^{66} online queries and negligible offline computations. The memory requirement is 2^{64}.

8.3 Internal-State Recovery Attack

Refer to Remark in Section 8.2.3.

9 Conclusion

This section consists of

- summarization of our contributions.
- words for CRYPTOREC Committee.

This report has evaluated the security of PC-MAC-AES and its underlying component AES-4R.

9.1 Our Contributions

The high-level overview of our contributions is as follows.

First of all, we listed all the attacks applicable to PC-MAC-AES. Strictly speaking, so far only two papers were published explicitly on the evaluation of PC-MAC-AES to our best knowledge. One is published by Yuan *et al.* in CRYPTO 2009 [20], which proposed internal-state-recovery and full-key-recovery attacks. The other one is published by Jia *et al.* in CANS 2009 [9], which proposed a forgery attack. However, PC-MAC shares similar structure with other MACs such as EMAC and TMAC. Therefore previous published attacks on other MACs may be also applicable to PC-MAC-AES.

- **The designers have mentioned such event in their submission report [15]. However, they did not state which previous attacks can be adapted to attack PC-MAC-AES and how much the complexity will be.**

We summarized and formalized several attacks including distinguishing, forgery partial-key-recovery and full-key-recovery attacks on PC-MAC-AES. Interestingly, our full-key-recovery attacks on PC-MAC-AES are more efficient than Yuan *et al.*'s attack [20].

Moreover, we successfully find several attacks based on the weakness of new feature of PC-MAC-AES, which reveals the security margin loss of PC-MAC-AES compared with previous proposed MACs. The details are given below.

1. *Subkey-recovery attacks.*

   Subkeys $\{U_1, U_2, \ldots, U_d,\ K_1^{xor}, \ldots, K_{d-1}^{xor}\}$ are generated by using L and $E_K(\cdot)$. We showed that after obtaining the knowledge of L, these subkeys can be easily recovered. The complexity is just one chosen query for each subkey.

2. *Distinguish PC-MAC-AES from PC-MAC-RP.*

   We will firstly propose an attack procedure, which can distinguish PC-MAC-AES from another PC-MAC instantiation with a Random Permutation (RP).

Secondly, we discuss about an interesting question:

what potential weakness will the usage of 4-round AES introduce to CBC-MACs including PC-MAC-AES and its variants?

As we can see, the most attractive design point of PC-MAC-AES is using 4-round AES permutations instead of the full-round AES to process partial message blocks, which will improve the efficiency, but still keep provable security. The theoretical foundation of such replacement is that 4-round AES with independent round keys have similar differential property with full-round AES. Besides PC-MAC-AES, several other MACs such as Pelican-MAC are also using 4-round AES to improve the efficiency. On the other hand, a question will arise: will such replace weaken the security of CBC-MACs from any sense?

Our answer is *yes*. We proposed an attack on generic CBC-MAC variants based on 4-round AES with independent round keys. More precisely, we are able to recover the round keys and the internal state value with a complexity of 2^{67} online queries and 2^{40} offline computations.

Interestingly, our results imply an upper bound of subkey-recovery resistance for CBC-MACs based on 4-round AES.

9.2 For CRYPTOREC Committee

We made a comparison between CMAC [16], which was recommended by NIST for block-cipher-based authentication in 2005, and PC-MAC-AES.

- From the view of security.
  From our subkey-recovery and DIS-PC-MAC$^{E_K(\cdot),\mathrm{RP}}$. attacks, PC-MAC-AES has a weaker security margin than CMAC.

- From the view of efficiency.
  PC-MAC-AES is about 1.5 $\sim$ 2.5 times faster than CMAC.

References

[1] *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, volume 1007 of *Lecture Notes in Computer Science*. Springer, 1995.

[2] ISO/IEC 9797-1. Information technology – security techniques – message authentication codes (macs) – part 1: Mechanisms using a block cipher. Technical report, International Organization for Standards.

[3] American Bankers Association. Ansi x9.19, financial institution retail message authentication. 1986.

[4] John Black and Phillip Rogaway. Cbc macs for arbitrary-length messages: The three-key constructions. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2000.

[5] Karl Brincat and Chris J. Mitchell. New cbc-mac forgery attacks. In Vijay Varadharajan and Yi Mu, editors, *ACISP*, volume 2119 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2001.

[6] Joan Daemen and Vincent Rijmen. The pelican mac function. Cryptology ePrint Archive, Report 2005/088. `http://eprint.iacr.org/`.

[7] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[8] Tetsu Iwata and Kaoru Kurosawa. Omac: One-key cbc mac. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[9] Keting Jia, Xiaoyun Wang, Zheng Yuan, and Guangwu Xu. Distinguishing and second-preimage attacks on cbc-like macs. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS*, volume 5888 of *Lecture Notes in Computer Science*, pages 349–361. Springer, 2009.

[10] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear cryptanalysis for two-round advanced encryption standard. Cryptology ePrint Archive, Report 2005/321, 2005. `http://eprint.iacr.org/2005/321`.

[11] Lars R. Knudsen and Bart Preneel. Macdes: A mac based on des. In *Electronics Letter*, volume 34, pages 871–873, 1998.

[12] Kaoru Kurosawa and Tetsu Iwata. Tmac: Two-key cbc mac. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2003.

[13] Kazuhiko Minematsu and Yukiyasu Tsunoo. Provably secure macs from differentially-uniform permutations and aes-based implementations. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2006.

[14] Chris J. Mitchell. Partial key recovery attacks on xcbc, tmac and omac. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 155–167. Springer, 2005.

[15] NEC. メッセージ認証コード pc-mac-aes.

[16] NIST. Recommendation for block cipher modes of operation: The cmac mode for authentication. `http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf`.

[17] Sangwoo Park, Soo Hak Sung, Seongtaek Chee, E-Joong Yoon, and Jongin Lim. On the security of rijndael-like structures against differential and linear cryptanalysis. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 176–191. Springer, 2002.

[18] Sangwoo Park, Soo Hak Sung, Sangjin Lee, and Jongin Lim. Improving the upper bound on the maximum differential and the maximum linear hull probability for spn structures and aes. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2003.

[19] Bart Preneel and Paul C. van Oorschot. Mdx-mac and building fast macs from hash functions. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 1995.

[20] Zheng Yuan, Wei Wang, Keting Jia, Guangwu Xu, and Xiaoyun Wang. New birthday attacks on some macs based on block ciphers. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 209–230. Springer, 2009.