

暗号技術仕様書  
PC-MAC-AES

日本電気株式会社

## 目次

1	概要	2
2	暗号アルゴリズム仕様	2
2.1	記法 . . . . .	2
2.2	基本関数および変数 . . . . .	2
2.3	全体構成 . . . . .	3
2.4	鍵スケジュール . . . . .	4
2.5	認証タグ生成 . . . . .	4
2.6	タグ検証 . . . . .	6
2.7	バージョン情報 . . . . .	7
3	推奨用途	8

## 1 概要

この文書はメッセージ認証コード PC-MAC-AES の仕様を説明するものである。

PC-MAC-AES は、カウンターなどの初期値を必要としない、決定的 (deterministic) MAC 関数である。AES をベースとしており、メッセージ認証を目的としたブロック暗号モードの一種といえるが、AES の段関数を取り出して部品として用いる点が CBC-MAC など従来のモードと異なる。具体的には AES そのものとその 4 段関数を用いている。これにより、AES ベースの CBC-MAC などより 1.4 から 2.5 倍ほど (推奨パラメータにおいては 1.4 から 2 倍ほど) 高速に処理が可能となる。同様のアイデアに基づくメッセージ認証コードの提案は過去に alpha-MAC[7] や Pelican[8] などがあったが、これらと異なり、PC-MAC-AES は CMAC[4] などと同等の証明可能安全性を持つ、すなわちその安全性が純粋に AES そのものの安全性に帰着可能であるという利点がある。実装性に関しては、基本的に AES の段関数があれば実装可能であり、 $GF(2^{128})$  上の乗算など付加的な代数演算を必要としないため、実装規模を小さく抑えることが可能である。処理構造は CBC-MAC と類似しており、シンプルである。さらに、ブロック暗号の鍵と独立な鍵を用いて、終端処理において TMAC[13]、CMAC などと同様のマスク処理を用いており、これにより、メッセージパディングにより発生しうる冗長な暗号関数の呼び出しがないように設計されている。これは特にメッセージが短いときに効果的である。

## 2 暗号アルゴリズム仕様

### 2.1 記法

本稿で用いる記法を定義する。バイナリ系列はすべて big endian 表記とする。

- $|x|$ : バイナリ系列  $x$  のビット長
- $\parallel$ : データ連結
- $\oplus$ : 排他的論理和
- $x \ll 1$ :  $x$  を 1 ビット左論理シフト ( $x = x_1 \parallel x_2 \parallel \dots \parallel x_n$ ,  $|x_i| = 1$  なら  $x \ll 1 = x_2 \parallel x_3 \parallel \dots \parallel x_n \parallel 0$ )
- $\text{msb}(x)$ :  $x$  の最上位ビット
- $0^s$ : ビット 0 の  $s$  個の連結,  $s = 0$  のときは空 ( $x \parallel 0^0 = x$ )
- $[i]$ : 非負整数  $i$  について,  $i$  の 128 ビット表現, 例えば  $[2] = 00 \dots 10$ 。

また、本稿での AES はすべて 128 ビット鍵の AES を指すものとする。

### 2.2 基本関数および変数

本稿で用いる基本的関数と変数を定義する。

- $M$ : 任意長のメッセージ
- $T$ : 認証に用いるタグ
- $K$ : AES の 128 ビット鍵
- $L$ :  $K$  と独立な 128 ビット鍵

- $d$ : オーダ (PC-MAC-AES が用いるパラメータ (正整数))
- $\pi$ : タグのビット長
- $E_K$ : 128 ビットの  $K$  を鍵とした AES の暗号化関数
- $G_U$ : 4 段 AES 関数, ただし 1 段目の鍵は全ゼロ, 384 ビットの  $U = U^{(1)}\|U^{(2)}\|U^{(3)}$  が鍵,  $U^{(j)}$  は  $j + 1$  段目の 128 ビット段鍵 (図 1 参照)
- $K_i^{\text{xor}}$ : 128 ビット補助鍵 ( $i = 1, \dots, d - 1$ )
- $\text{cut}_\pi(x)$ :  $x$  の最上位ビットから  $\pi$  ( $1 \leq \pi \leq |x|$ ) ビットを取り出した系列
- $\text{mul2}(x)$ : 128 ビットの  $x$  を有限体  $\text{GF}(2^{128})$  上の元とした 2 倍算

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

- $\text{pad}(x)$ : 長さが 128 ビット以下の入力  $x$  について

$$\text{pad}(x) = \begin{cases} x & \text{if } |x| = 128 \\ x\|1\|0^{128-|x|-1} & \text{if } |x| < 128. \end{cases}$$

## 2.3 全体構成

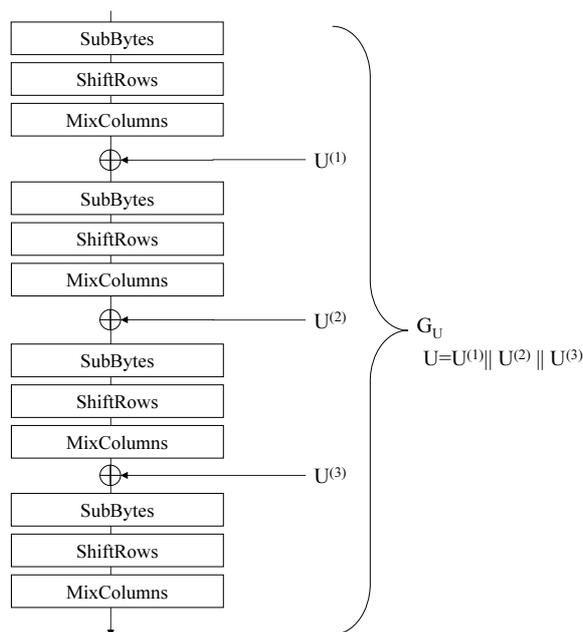
以下に PC-MAC-AES の全体構成を述べる。PC-MAC-AES は AES と 4 段 AES 関数をモジュールとする決定的 (deterministic) MAC であり, カウンターなどの初期値は必要としない。PC-MAC-AES の鍵は, AES の 128 ビット鍵  $K$  と, これと独立な 128 ビット鍵  $L$  の合計 256 ビットである。パラメータとして, タグの長さ  $\pi$  およびオーダと呼ぶ正整数  $d$  を用いる。  $\pi$  は  $1 \leq \pi \leq 128$  を満たすが, 推奨値として 64 以上を,  $d$  は実装性の観点から 1 から 5 までを推奨する。

4 段 AES 関数  $G_U$  は図 1 に示される 384 ビットの  $U = (U^{(1)}\|U^{(2)}\|U^{(3)})$  を鍵とした鍵付き置換である。AES の 1 段目の段鍵加算を省略し, 以後, SubBytes 関数, ShiftRows 関数, MixColumns 関数, 段鍵加算をこの順序で 3 回行い, 最後に SubBytes 関数, ShiftRows 関数, MixColumns 関数を適用する (各関数の詳細は文献 [3] 参照のこと)。それぞれ,  $U^{(1)}$  が 2 段目の鍵,  $U^{(2)}$  が 3 段目の鍵,  $U^{(3)}$  が 4 段目の鍵として使用される。

パラメータ  $(\pi, d)$  を持つ PC-MAC-AES を  $\text{PC-MAC-AES}_{(\pi, d)}$  と表記する。特に  $\pi$  の指定をせず  $\text{PC-MAC-AES}_{(d)}$  と書いたときは  $\pi = 128$  として扱う。さらに,  $d$  と  $\pi$  を特に指定する必要がないか, 文脈から明らかな場合には単に  $\text{PC-MAC-AES}$  と表記する。

$\text{PC-MAC-AES}_{(\pi, d)}$  は AES を用いた鍵スケジュール処理  $\text{KEYSCH}$  を事前処理として実行する。鍵スケジュール処理  $\text{KEYSCH}$  が終了したのち, 実際の任意長メッセージ  $M^*$ <sup>1</sup> に付与する認証タグ (以降は単にタグと呼ぶ) を生成できるようになる。タグ生成処理を  $\text{TAGGEN}$  とし, これを用いて  $\pi$  ビットタグ  $T$  を生成する。メッセージ  $M$  の送信者は  $(M, T)$  を送信する。受信者はこれを受け取った後, 検証手続き  $\text{TAGVER}$  を用いて, 受信したメッセージが正当なものであるかどうかを検証する。 $\text{TAGVER}$  の入力は  $(M, T)$  および  $\pi, d$  であり, 出力はバイナリ値であり, 1 ならば受理 (正当と判断), 0 ならば棄却を意味する。以下, それぞれの処理について説明する。

<sup>1</sup> 現在の仕様ではいわゆる empty string (長さ 0 のメッセージ) には未対応である。

図 1 4 段 AES 関数  $G_U$ 

## 2.4 鍵スケジュール

鍵スケジュール処理  $\text{KEYSCH}$  は AES の暗号化関数  $E_K$  と、 $K$  と独立な鍵  $L$  を用いて 4 段 AES 関数の  $d$  個の鍵  $U_1, \dots, U_d$  と、 $d-1$  個の 128 ビット補助鍵  $K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  (合計  $384 \cdot d + 128 \cdot (d-1)$  ビット) を生成するものであり、形式的には  $d, K, L$  を入力として以下のように行う。

まず 4 段 AES 関数の 384 ビット鍵を  $d$  個生成する。これは、

$$E_K(L \oplus [0]) \| E_K(L \oplus [1]) \| \dots \| E_K(L \oplus [3d-1]) \quad (1)$$

という  $384 \cdot d$  ビット系列を求め、先頭から 384 ビットづつを鍵とする。つまり  $i = 1, \dots, d$  について  $U_i = (U_i^{(1)} \| U_i^{(2)} \| U_i^{(3)}) = (E_K(L \oplus [3(i-1)]) \| E_K(L \oplus [3(i-1)+1]) \| E_K(L \oplus [3(i-1)+2]))$  とする。次に  $d-1$  個の 128 ビット補助鍵  $K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  を、

$$K_1^{\text{xor}} = E_K(L \oplus [3d]), K_2^{\text{xor}} = E_K(L \oplus [3d+1]), \dots, K_{d-1}^{\text{xor}} = E_K(L \oplus [4d-2]) \quad (2)$$

として生成する。

なお  $d=1$  の場合、補助鍵は存在せず、4 段 AES 関数の 384 ビット鍵  $U_1$  のみを生成することになる。

具体的な鍵スケジュール  $\text{KEYSCH}$  は Algorithm 2.1 に従って行われる。

## 2.5 認証タグ生成

鍵スケジュール  $\text{KEYSCH}$  の後、 $\text{TAGGEN}$  を用いて任意のバイナリ系列メッセージ  $M$  に対して  $\pi$  ビットの認証タグを生成する。 $\text{TAGGEN}$  は形式的には  $d, \pi, K, L, M$  および  $\text{KEYSCH}$  の出力  $U_1, \dots, U_d$  と  $K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  を入力とする。

**Algorithm 2.1:** KEYSCH( $d, K, L$ )

```

for  $i \leftarrow 1$  to  $d$ 
  do  $\begin{cases} U_i^{(1)} \leftarrow E_K(L \oplus [3(i-1)]) \\ U_i^{(2)} \leftarrow E_K(L \oplus [3(i-1) + 1]) \\ U_i^{(3)} \leftarrow E_K(L \oplus [3(i-1) + 2]) \\ U_i \leftarrow U_i^{(1)} \| U_i^{(2)} \| U_i^{(3)} \end{cases}$ 
for  $j \leftarrow 1$  to  $d-1$ 
  do  $K_j^{\text{xor}} \leftarrow E_K(L \oplus [3d + j - 1])$ 
return  $(U_1, U_2, \dots, U_d, K_1^{\text{xor}}, K_2^{\text{xor}}, \dots, K_{d-1}^{\text{xor}})$ 

```

関数として用いるのは AES 暗号化関数  $E_K$  および  $G_{U_i}$ ,  $i = 2, \dots, d$  について  $U_i$  と  $K_{i-1}^{\text{xor}}$  を鍵とした

$$G_{U_i}^{\oplus}(x) = G_{U_i}(K_{i-1}^{\text{xor}} \oplus x)$$

という鍵付き関数である。

まず  $M$  を先頭から 128 ビット単位に分割して,  $M_1, M_2, \dots, M_m$  を得る。ここで,  $M = M_1 \| M_2 \| \dots \| M_{m-1} \| M_m$  であり, また  $i = 1, \dots, m-1$  について  $|M_i| = 128$ ,  $1 \leq |M_m| \leq 128$  である。

次に Ch 関数を,

$$\text{Ch}[F_1, \dots, F_{m-1}](M) = \text{pad}(M_m) \oplus F_{m-1}(M_{m-1} \oplus F_{m-2}(\dots F_2(M_2 \oplus F_1(M_1)) \dots))$$

として定義する (文献 [14] における同名の関数と定義が細部で異なる点に注意)。ただし  $F_i$  は 128 ビット入出力を持つ鍵付き関数で,  $i = 1, \dots, d$  について

$$F_{(d+1)(i-1)+1} = E_K, F_{(d+1)(i-1)+2} = G_{U_1}, F_{(d+1)(i-1)+3} = G_{U_2}^{\oplus}, \dots, F_{(d+1)(i-1)+d+1} = G_{U_d}^{\oplus}$$

とおく。

タグ生成は,

$$h = \text{Ch}[F_1, \dots, F_{m-1}](M_1 \| M_2 \| \dots \| M_{m-1} \| \text{pad}(M_m))$$

という 128 ビット値  $h$  を計算し,  $\pi$  ビットタグ  $T$  を,

$$T = \begin{cases} \text{cut}_{\pi}(E_K(\text{mul2}(L) \oplus h)) & \text{if } |x_m| = 128 \\ \text{cut}_{\pi}(E_K(\text{mul2}(\text{mul2}(L)) \oplus h)) & \text{if } |x_m| < 128 \end{cases}$$

として出力する。

具体的なタグ生成は Algorithm 2.2 に従って行われる。ただし, TAGGEN の入力のうち KEYSCH 出力の部分  $U_1, \dots, U_d, K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  の記載は省略してある。

また,  $d = 1$  と  $d = 2$  でのタグ生成の例を図 2 と図 3 に示す。

**Algorithm 2.2:** TAGGEN( $d, \pi, K, L, M$ )

Partition  $M$  into  $M_1, M_2, \dots, M_m$

if  $m = 1$

  then  $h \leftarrow \text{pad}(M_1)$

  else  $\left\{ \begin{array}{l} s \leftarrow 0^{128} \\ \text{for } i \leftarrow 1 \text{ to } m-1 \\ \quad \text{do } \left\{ \begin{array}{l} w \leftarrow (i-1) \bmod (d+1) \\ \text{if } w = 0 \\ \quad \text{then } s \leftarrow E_K(s \oplus M_i) \\ \quad \text{else if } w = 1 \\ \quad \text{then } s \leftarrow G_{U_1}(s \oplus M_i) \\ \quad \text{else } s \leftarrow G_{U_w}(s \oplus K_{w-1}^{\text{xor}} \oplus M_i) \end{array} \right. \\ h \leftarrow s \oplus \text{pad}(M_m) \end{array} \right.$

if  $|M_m| \bmod 128 = 0$

  then  $h \leftarrow \text{mul2}(L) \oplus h$

  else  $h \leftarrow \text{mul2}(\text{mul2}(L)) \oplus h$

$T \leftarrow \text{cut}_\pi(E_K(h))$

return ( $T$ )

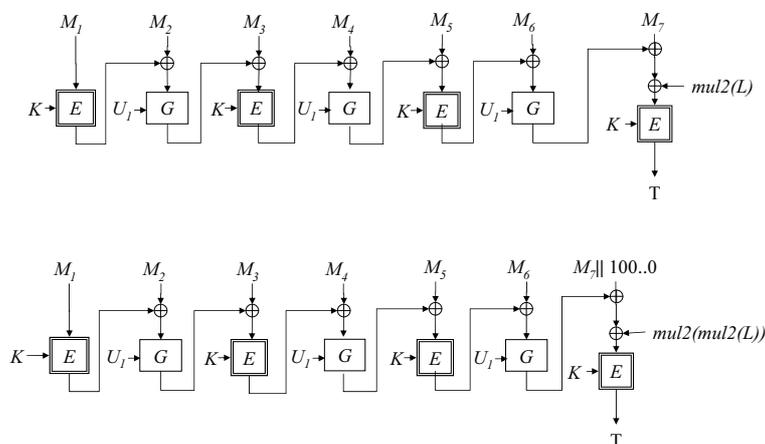


図2 PC-MAC-AES<sub>(128,1)</sub> のタグ生成の例 (上:  $|M_7| = 128$  のとき, 下:  $|M_7| < 128$  のとき.)

## 2.6 タグ検証

タグの検証手続き TAGVER は一般的な決定的 MAC 関数のケースと同様である。すなわち、受信したメッセージとタグの対  $(M, T)$  から  $M$  を取り出し、これに対して TAGGEN を適用して得た結果  $T'$  と受信した  $T$  が一致すれば受理として 1 を出力し、そうでなければ棄却として 0 を出力する。TAGGEN 同様に KEYSCH

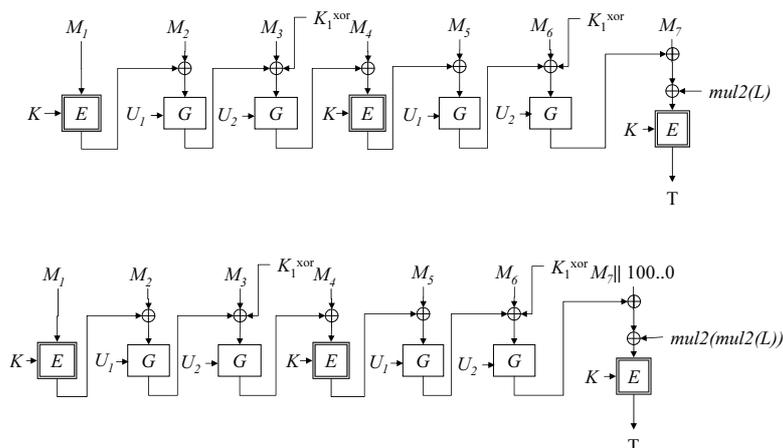


図3 PC-MAC-AES<sub>(128,2)</sub> のタグ生成の例 (上:  $|M_7| = 128$  のとき, 下:  $|M_7| < 128$  のとき)

を事前処理として必要とし, 入力は  $d, \pi, K, L, M$  と KEYSCH 出力である  $U_1, \dots, U_d, K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  と, 受信したタグ  $T$  となる. 手続きを Algorithm 2.3 に記す. ただし TAGVER と TAGGEN の入力のうち,  $U_1, \dots, U_d, K_1^{\text{xor}}, \dots, K_{d-1}^{\text{xor}}$  は記載を省略してある.

**Algorithm 2.3:** TAGVER( $d, \pi, K, L, M, T$ )

$T' \leftarrow \text{TAGGEN}(d, \pi, K, L, M)$

if  $T = T'$

  then return (1)

  else return (0)

## 2.7 バージョン情報

文献 [14] に記載の PC-MAC との相違について述べる. ここでは, 簡便のため文献 [14] に記載の PC-MAC をオリジナル PC-MAC と呼ぶことにする. オリジナル PC-MAC はそもそもブロック暗号の段間数を部分的に取り出して使用するものであり, AES に限ったアルゴリズムではなく, 原則的にはどのような (段構造を持つ) ブロック暗号でも使用できる. また文献 [14] ではオリジナル PC-MAC を AES で実装する際に 4 段 AES の最後の ShiftRows と MixColumns を省略して使用することが提案されているが, ここでは省略をしない. この理由は, 典型的なソフトウェア実装では SubBytes, ShiftRows, MixColumns をまとめた表を作成して効率化を図っており, SubBytes のみの取り出しが困難あるいは可能であっても速度の低下や実装コストの増加につながる可能性があるからである. ShiftRows と MixColumns は線形処理であり, 一般的にハードウェア・ソフトウェア問わず処理の負担はごくわずかである. さらに PC-MAC-AES の安全性証明は用いる部分関数の差分確率の上界にのみ依存している. また, 部分関数の最後の処理に ShiftRows と MixColumns があっても差分確率の上界にはまったく影響がないため, この変更は安全性証明にも全く影響を及ぼさない (自己評価書 [1] 参照のこと). 上記の, 4 段 AES 関数の最後の ShiftRows と MixColumns を省略したバー

ジョンと PC-MAC-AES とは互換性はない。

また、文献 [14] ではタグ長がブロックサイズで固定であるが、汎用性を鑑みてここではタグ長をパラメータとし 128 ビットよりも短い値を取りうるとした。この変更による証明可能安全性への影響については自己評価書 [1] で述べる。

### 3 推奨用途

推奨用途としては、比較的小規模のハードウェアや、ソフトウェア実装において省メモリ性と高速性の両立を求めるケース、あるいはコードが動作する環境が様々に変わる環境 (Java など)、ハードウェア・ソフトウェアが混在するネットワーク、または AES-NI[2] など専用 AES 命令があるプラットフォームにおける高速なメッセージ認証が考えられる。また、AES 以外の特別な関数を用いたユニバーサルハッシュ関数をベースとしたメッセージ認証コードと比べて、AES の実装環境のみで実装できることによる性能の予測のしやすさ、実装の容易性が利点としてあげられる。

### 参考文献

- [1] 自己評価書 PC-MAC-AES, 日本電気株式会社, 2010.
- [2] Intel Advanced Encryption Standard (AES) Instructions Set - Rev 3,  
<http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/>
- [3] NIST FIPS-197. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [4] NIST Special Publication 800-38B,  
Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.  
[http://csrc.nist.gov/publications/nistpubs/800-38B/SP\\_800-38B.pdf](http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf)
- [5] B. den Boer, J.P. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgård, M. Dichtl, W. Fumy, M. van der Ham, C.J.A. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, and J. Vandewalle, *RIPE Integrity Primitives*, final report of RACE Integrity Primitives Evaluation. 1995.
- [6] D. J. Bernstein. “The Poly1305-AES Message-Authentication Code.” *Fast Software Encryption, FSE’05*, LNCS 3557, pp. 32-49, 2005.
- [7] J Daemen and V. Rijmen. “A New MAC Construction ALRED and a Specific Instance ALPHA-MAC.” *Fast Software Encryption, FSE’05*, LNCS 3557, pp. 1-17, 2005.
- [8] J Daemen and V. Rijmen. “The Pelican MAC Function.” *IACR ePrint Archive*, 2005/088.
- [9] S. Halevi and H. Krawczyk. “MMH:Software Message Authentication in the Gbit/second rates.” *Fast Software Encryption, FSE’97*, LNCS 1267, pp. 172-189, 1997.
- [10] T. Iwata and K. Kurosawa. “OMAC: One-Key CBC MAC.” *Fast Software Encryption- FSE’03*, LNCS 2887, pp. 129-153, 2003.
- [11] L. Keliher and J. Sui. “Exact Maximum Expected Differential and Linear Probability for 2-Round Advanced Encryption Standard (AES).” *IACR ePrint Archive*, 2005/321.
- [12] L. Keliher and J. Sui. “Exact maximum expected differential and linear cryptanalysis for two-round Advanced Encryption Standard.” *IET Information Security*, Vol. 1, No. 2, pp. 53-57, June. 2007.

- 
- [13] K. Kurosawa and T. Iwata. “TMAC: Two-Key CBC MAC.” *Topics in Cryptology- CT-RSA 2003*, LNCS 2612, pp. 33-49, 2003.
- [14] K. Minematsu and Y. Tsunoo. “Provably Secure MACs From Differentially-uniform Permutations and AES-based Implementations.” *Fast Software Encryption, FSE '06*, LNCS 4047, 2006.