# Specification of Cryptographic Technique PC-MAC-AES

NEC Corporation

# Contents

# 1 Design Criteria

This documents specifies a message authentication code (MAC) called PC-MAC-AES.

PC-MAC-AES is a deterministic MAC function, i.e. it does not need an initial vector such as a counter. It is based on AES [3], however, unlike usual block cipher modes of operation it uses a 4-round version of AES in addition to the full AES. With this it enables about 1.4 to 2.5 times (1.4 to 2.0 within the recommended parameters) faster operation than usual MAC mode, e.g., CBC-MAC, using AES. Similar ideas can be seen in previous proposals, for instance alpha-MAC[7] and Pelican[8]. However, unlike alpha-MAC and Pelican PC-MAC-AES uses a sub key scheduling for 4-round AES to retain the provable security. I.e., it is proved that if AES is secure then PC-MAC-AES is also secure. The overall procedure is simple. It is similar to CBC-MAC, where AES encryption is periodically substituted with a 4-round AES. Its implementation is almost as easy as CBC-MAC-AES, since it does not need any special algebraic operation such as multiplication over GF($2^{128}$). In addition, the finalization is done with the second 128-bit key in a similar manner to TMAC[13] and CMAC. Thanks to this finalization we remove redundant AES invocations possibly caused with the use of unoptimized message padding. This can be beneficial in particular for short messages.

# 2 Specification

## 2.1 Notations

We will use the following notations. A binary string is expressed by big endian.

- $|x|$ : the bit length of a binary string, $x$
- $\|$ : concatenation of binary strings
- $\oplus$ : exclusive-or operation
- $x \ll 1$ : one bit logical left shift of $x$ (i.e. if $x = x_1\|x_2\|\dots\|x_n$、 $|x_i| = 1$, $x \ll 1 = x_2\|x_3\|\dots x_n\|0$)
- $\mathtt{msb}(x)$ : the most significant bit (MSB) of $x$
- $0^s$ : a sequence of $s$ 0s. When $s = 0$ it means an empty string, i.e., $x\|0^0 = x$
- $[i]$ : 128-bit representation of a nonnegative integer $i$, e.g., $[2] = 00\dots10$.

Throughout this document we use the word "AES" to mean the 128-bit key version of AES.

## 2.2 Basic Functions

We define the basic functions.

- $M$ : a message of any bit length
- $T$ : tag used for authentication
- $K$ : 128-bit key of AES
- $L$ : secondary 128-bit key
- $d$ : order (a parameter of PC-MAC-AES, positive integer)
- $\pi$ : bit length of tag $T$
- $E_K$ : AES encryption function using 128-bit key, $K$
- $G_U$ : 4-round AES function with 384-bit key, $U$. Here the first round sub key is all-zero, the second is $U^{(1)}$, the third is $U^{(2)}$, and the fourth is $U^{(3)}$. See Fig. 1 for reference.
- $\mathtt{cut}_\pi(x)$ : the first $\pi$ bits of $x$, where $1 \le \pi \le |x|$
- mul2($x$) : multiply-by-two operation defined over the finite field GF($2^{128}$), defined as

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

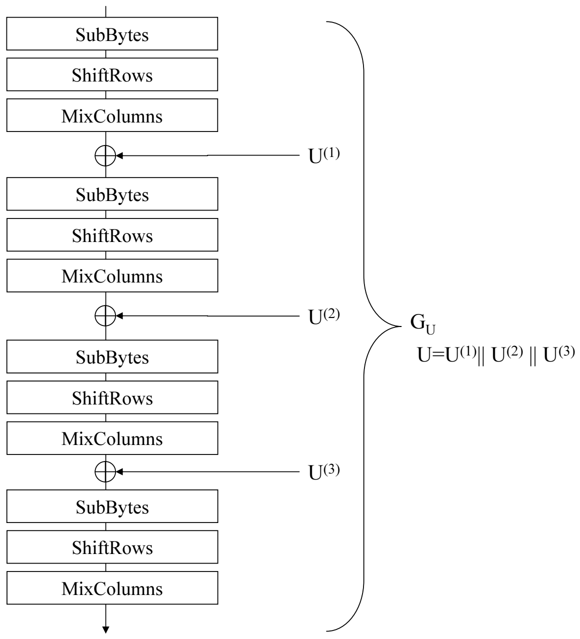

Fig. 1 4-round AES function $G_U$

- $\texttt{pad}(x)$ : a padding function for $x$ with $|x| \leq 128$, defined as

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

## 2.3 Overall Structure

This section describes the overall structure of the MAC function PC-MAC-AES. It consists of AES and its 4-round version. PC-MAC-AES is a deterministic MAC, which means that it does not need any initial vector such as a counter. The key of PC-MAC-AES consists of the 128-bit key of AES, $K$, and an independent 128-bit key, $L$. Thus PC-MAC-AES's key length is 256 bits. PC-MAC-AES has two parameters, the tag length, $\pi$, and $d$, a positive integer called order. In fact $\pi$ can be any integer satisfying $1 \leq \pi \leq 128$, however we recommend $\pi \geq 64$ for security. Also we recommend $d$ to be an integer from 1 to 5 for implementation.

The 4-round AES function is denoted by $G_U$, where 384-bit $U = (U^{(1)}\|U^{(2)}\|U^{(3)})$ is the key (see Section 2.2). When $x$ is given to $G_U$, it omits the first round sub key addition, and performs SubBytes, ShiftRows, MixColumns, and AddRoundKey for three times, and finally performs SubBytes, ShiftRows, MixColumns (see reference [3] for the details of these functions). As described, $U^{(1)}$ is the second round sub key, $U^{(2)}$ is the third, and $U^{(3)}$ is the fourth. We use PC-MAC-AES$_{(\pi,d)}$ to denote PC-MAC-AES having parameters $\pi$ and $d$. If we omit the notation of $\pi$ (e.g. PC-MAC-AES$_{(d)}$), we implicitly assume $\pi = 128$. Moreover, if $d$ and $\pi$ are not needed to be specified or clear from contexts, we simply write PC-MAC-AES.

PC-MAC-AES$_{(\pi,d)}$ needs a key schedule (a preprocessing) called KeySch using AES. After KeySch, PC-MAC-AES$_{(\pi,d)}$ can process a message of any length[*1] to generate the authentication tag (we will simply call it tag hereafter). The procedure of tag generation is called TagGen, which accepts a message, $M$, and generates a $\pi$-bit tag, $T$. Then the pair $(M, T)$ will be sent to the receiver. The receiver confirms

[*1] The current specification does not support the empty string, an input of length 0.

if the sent message is authentic or not, using the verification procedure, TagVer. It accepts $(M, T)$, $\pi$, and $d$ to produce a binary output, where 1 indicates the acceptance (i.e. the sent message is authentic) and 0 indicates the rejection (i.e. the sent message is not authentic). In the following, we describe the detail of each procedure.

## 2.4 Key Schedule

Key schedule KeySch produces a string of $384 \cdot d + 128 \cdot (d-1)$ bits using AES. KeySch works as follows. First we produce $d$ keys of 4-round AES function $G_U$. Here, we compute

$$E_K(L \oplus [0]) \| E_K(L \oplus [1]) \| \ldots \| E_K(L \oplus [3d-1]) \tag{1}$$

and partition it into 384-bit sequences (i.e. three blocks) from the first. That is, for $i = 1, \ldots, d$ we use $U_i = (E_K(L \oplus [3(i-1)]), E_K(L \oplus [3(i-1)+1]), E_K(L \oplus [3(i-1)+2]))$ as $U_i$ which denotes the $i$-th key of 4-round AES function $G_U$. Then we compute $d-1$ 128-bit supplementary keys, denoted by $K_1^{\mathrm{xor}}, \ldots, K_{d-1}^{\mathrm{xor}}$, as

$$K_1^{\mathrm{xor}} = E_K(L \oplus [3d]), K_2^{\mathrm{xor}} = E_K(L \oplus [3d+1]), \ldots, K_{d-1}^{\mathrm{xor}} = E_K(L \oplus [4d-2]). \tag{2}$$

Here, if $d = 1$ there is no need to compute a supplementary key and KeySch only produces a 384-bit key, $U_1$.

A concrete description of KeySch is written in Algorithm 2.1.

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

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

## 2.5 Tag Generation

After KeySch, we invoke TagGen for a message of any bit length, $M$, to produce $\pi$-bit tag $T$. Formally, a TagGen's input consists of $d$, $\pi$, $K$, $L$, $M$, and an output of KeySch, i.e., $U_1, \ldots, U_d$ and $K_1^{\mathrm{xor}}, \ldots, K_{d-1}^{\mathrm{xor}}$. The core components of TagGen are the AES encryption function $E_K$, $G_{U_1}$, and $G_{U_i}^{\oplus}$ for $i = 2, \ldots, d$, defined as

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

We first partition $M$ into 128-bit strings from the first, denoted by $M_1, M_2, \ldots, M_m$. Note that $M = M_1 \| M_2 \| \ldots M_{m-1} \| M_m$ and for $i = 1, \ldots, m-1$ we have $|M_i| = 128$, $1 \le |M_m| \le 128$.

Next, we define Ch function (which is slightly different from a function of the same name defined in [14]) as

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

where $F_i$ is a keyed function of 128-bit inputs and outputs. It is defined as

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

for $i = 1, \dots, d$.

Tag generation TagGen for $M$ is done by first computing a 128-bit string, $h$, as

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

and then computing the $\pi$-bit tag $T$ as

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

Concrete procedure of TagGen is described in Algorithm 2.2. Here, we omit the notation of $U_1, \dots, U_d, K^{\mathrm{xor}}_1, \dots, K^{\mathrm{xor}}_{d-1}$ from inputs of TagGen.

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

```
Partition $M$ into $M_1, M_2, \dots, M_m$
if $m = 1$
  then $h \leftarrow \mathtt{pad}(M_1)$
  else {
    $s \leftarrow 0^{128}$
    for $i \leftarrow 1$ to $m - 1$
      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 $s \leftarrow G_{U_1}(s \oplus M_i)$
          else $s \leftarrow G_{U_w}(s \oplus K^{\mathrm{xor}}_{w-1} \oplus M_i)$
      }
    $h \leftarrow s \oplus \mathtt{pad}(M_m)$
  }
if $|M_m| \bmod 128 = 0$
  then $h \leftarrow \mathrm{mul2}(L) \oplus h$
  else $h \leftarrow \mathrm{mul2}(\mathrm{mul2}(L)) \oplus h$
$T \leftarrow \mathtt{cut}_\pi(E_K(h))$
return $(T)$
```

For reference the operation of TagGen is depicted in Figs.2 and 3.

## 2.6 Tag Verification

Tag verification TagVer is as the same as the standard case of deterministic MAC function. That is, TagVer invokes KeySch as a preprocessing and on receiving a message-tag pair, $(M', T')$, gives it to TagGen (with $\pi$ and $d$) and compares the output of TagGen with $T'$. If they are the same, TagVer decides $M'$ to be authentic and outputs 1. Otherwise, TagVer decides $M'$ to not be authentic and outputs 0. As well as TagGen, it KeySch as a preprocessing and its input consists of $d, \pi, K, L, M$, and $U_1, \dots, U_d, K^{\mathrm{xor}}_1, \dots, K^{\mathrm{xor}}_{d-1}$, and the received tag, $T$. The procedure is in Algorithm 2.3. Here we omit the notation of $U_1, \dots, U_d, K^{\mathrm{xor}}_1, \dots, K^{\mathrm{xor}}_{d-1}$ from inputs of TagVer and TagGen.

## 2.7 Version Information

The difference between PC-MAC-AES defined in this document and PC-MAC defined in the reference [14], which we call "original" PC-MAC. The primal difference is that, original PC-MAC was designed to

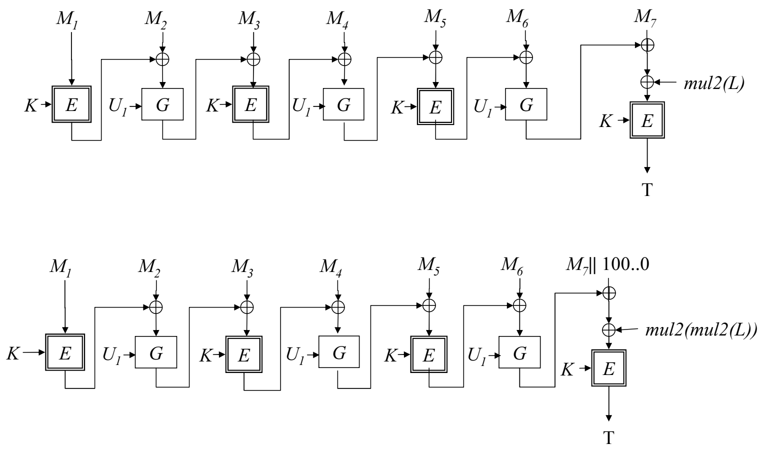

Fig. 2 Tag generation of PC-MAC-AES$_{(128,1)}$ (Top : the case $|M_7| = 128$, bottom : the case $|M_7| < 128$)

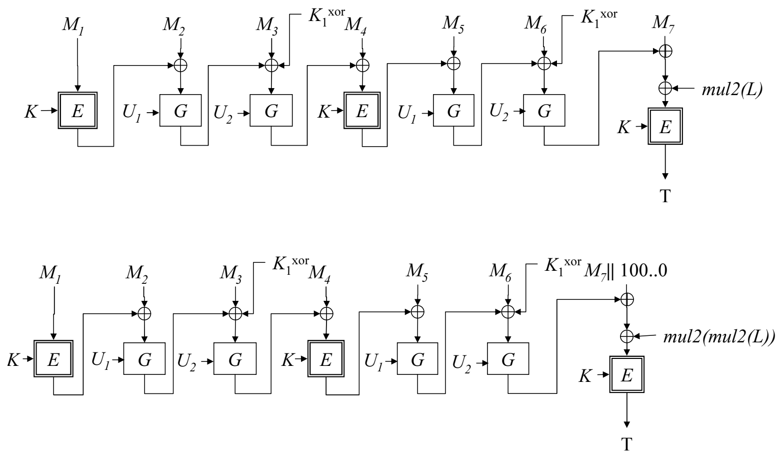

Fig. 3 Tag generation of PC-MAC-AES$_{(128,2)}$ (Top : the case $|M_7| = 128$, bottom : the case $|M_7| < 128$)

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

```
$T' \leftarrow$ TagGen$(d, \pi, K, L, M)$
if $T = T'$
  then return (1)
  else return (0)
```

be used with not only AES but also any (iterated) block cipher with a specification of the reduced-round version. In reference [14] it was proposed that original PC-MAC could be instantiated with AES and a "simplified" 4-round AES function obtained by omitting the last ShiftRows and MixColumns functions. However, we specify PC-MAC-AES without this omission. The reason is that such a "simplification" can complicate the implementation when ShiftRows and MixColumns are jointly computed with SubBytes by a table look up. Such implementation is typical in software. Note that ShiftRows and MixColumns are linear, thus the cost of these operations are quite small for both hardware and software.

In general a change in the underlying component of a MAC function can change the security. However, the security proof of original PC-MAC with simplified 4-round AES [14] can be applied to PC-MAC-AES too. This is because the security proof of PC-MAC-AES only depends on the security of AES and the differential probability of 4-round AES, and that post-processing of linear functions does not change the differential probability. For details, see the self-evaluation report [1].

Finally, we allow the final tag truncation to $\pi$ bits, as shorter tag is sometimes required in practice. The effect of this change is also described in the self-evaluation report.

## 3 Recommended Usage

PC-MAC-AES is considered to be useful for relatively small hardwares or for software with small memory. Moreover, it works fine for software under multiple platforms (such as Java) and platforms with a dedicated AES functionality, for instance AES-NI[2]. In addition, the ease of implementation and performance prediction is an advantage of PC-MAC-AES over MAC functions based on a special function independent of AES.

## References

[1] Self-evaluation report of PC-MAC-AES. NEC Corporation, 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`

[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.