# INTERNATIONAL STANDARD

# ISO/IEC 29192-6

First edition
2019-09

# Information technology — Lightweight cryptography —

## Part 6:
## Message authentication codes (MACs)

*Technologies de l'information — Cryptographie pour environnements contraints —*

*Partie 6: Codes d'authentification de message (MACs)*

Reference number
ISO/IEC 29192-6:2019(E)

# Contents

Page



 

# Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

A list of all parts in the ISO/IEC 29192 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

 

# Introduction

In an IT environment, it is often required that one can verify that electronic data has not been altered in an unauthorized manner and that one can provide assurance that a message has been originated by an entity in possession of the secret key. A MAC (Message Authentication Code) algorithm is a commonly used data integrity mechanism that can satisfy these requirements.

It is possible to take the first approach to realize a lightweight MAC by using the specified MAC algorithm in conjunction with a block cipher that can be chosen from ISO/IEC 29192-2 or ISO/IEC 18033-3, and in conjunction with a hash-function that can be chosen from ISO/IEC 29192-5. It is also possible to take the second approach to realize a lightweight MAC using a dedicated function. Examples of both approaches are specified in this document.

# Information technology — Lightweight cryptography —

# Part 6:
# Message authentication codes (MACs)

## 1 Scope

This document specifies MAC algorithms suitable for applications requiring lightweight cryptographic mechanisms. These mechanisms can be used as data integrity mechanisms to verify that data has not been altered in an unauthorized manner. They can also be used as message authentication mechanisms to provide assurance that a message has been originated by an entity in possession of the secret key.

The following MAC algorithms are specified in this document:

a) LightMAC;

b) Tsudik's keymode;

c) Chaskey-12.

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 18033-3, *Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*

ISO/IEC 29192-2, *Information technology — Security techniques — Lightweight cryptography — Part 2: Block ciphers*

ISO/IEC 29192-5, *Information technology — Security techniques — Lightweight cryptography — Part 5: Hash-functions*

## 3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 18033-3 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

**3.1**
**block cipher key**
key that controls the operation of a block cipher

[SOURCE: ISO/IEC 9797-1:2011, 3.2]

**3.2**
**encryption**
reversible operation by a cryptographic algorithm converting data into ciphertext so as to hide the information content of the data

[SOURCE: ISO/IEC 9797-1:2011, 3.6]

**3.3**
**hash-function**
function which maps strings of bits of variable (but usually upper bounded) length to fixed-length strings of bits, satisfying the following two properties:

— for a given output, it is computationally infeasible to find an input which maps to this output;

— for a given input, it is computationally infeasible to find a second input which maps to the same output

Note 1 to entry: Computational feasibility depends on the specific security requirements and environment.

[SOURCE: ISO/IEC 10118-1:2016, 3.4]

**3.4**
**key**
sequence of symbols that controls the operation of a cryptographic transformation

Note 1 to entry: Examples are encryption, decryption, cryptographic check function computation, signature, generation, or signature verification.

[SOURCE: ISO/IEC 9797-1:2011, 3.7]

**3.5**
**Message Authentication Code**
**MAC**
string of bits which is the output of a MAC algorithm

[SOURCE: ISO/IEC 9797-1:2011, 3.9]

**3.6**
**MAC algorithm**
algorithm for computing a function which maps strings of bits and a secret key to fixed-length strings of bits, satisfying the following two properties:

— for any key and any input string, the function can be computed efficiently;

— for any fixed key, and given no prior knowledge of the key, it is computationally infeasible to compute the function value on any new input string, even given knowledge of a set of input strings and corresponding function values, where the value of the $i$th input string might have been chosen after observing the value of the first $i$ - 1 function values (for integers $i$ > 1)

[SOURCE: ISO/IEC 9797-1:2011, 3.10]

**3.7**
**word**
string of 32 bits used in Chaskey-12 MAC algorithm

## 4 Symbols and abbreviated terms

| | |
|---|---|
| $a \leftarrow b$ | set variable $a$ to the value of $b$ |
| $E(K, P)$ | encryption of the plaintext $P$ with the block cipher $E$ using the key $K$ |
| $h$ | hash-function |

 

| | |
|---|---|
| $IV$ | $t$-bit initializing value |
| I2BS($x$, $g$) | function that takes as input a non-negative integer $x$ and outputs a bit string of length $g$ corresponding to its binary representation |
| $K_i$ | block cipher key taken by the underlying block cipher used in LightMAC ($i$ = 1, 2) |
| $m$ | message string to be input to the MAC algorithm |
| $m'$ | message string after the padding has been applied |
| $m_i'$ | $i$ th $n$-bit block of the padded-message string $m'$ |
| $m'_i{}^{(n-s)}$ | $i$ th $(n-s)$-bit block of the padded-message string $m'$ for $i < l$ where $m'_1{}^{(n-s)} \| m'_2{}^{(n-s)} \|...\| m'_\ell = m'$ |
| $s$ | counter size |
| $t$ | length of the MAC in bits |
| $v_i$ | 32-bit words used to store the results of intermediate computations |
| $X\|_j$ | $j$-bit unsigned integer obtained from the $u$-bit unsigned integer $X$ by taking the $j$ least significant bits of $X$ ($1 \leq j \leq u$) |
| $X$<<1 | operation of left shift by one bit, i.e. if $X$ is a word then $X$ << 1 denotes the word obtained by left-shifting the contents of $X$ by one position |
| $X$ <<< $n$ | operation of ‘circular left shift’ by $n$ bit positions, i.e. if $X$ is a word and $n$ is a non-negative integer then $X$<<<$n$ denotes the word obtained by left-shifting the contents of $X$ by $n$ positions in a cyclic fashion |
| $0^s$ | string consisting of $s$ zero-bits |
| $\oplus$ | bitwise exclusive-OR operation |
| $\boxplus$ | addition modulo $2^{32}$ |
| $\|X\|$ | the length of bit string $X$ in bits |
| $\|\|$ | concatenation of bit strings |
| $+_w$ | addition modulo $2^w$ operation, where $w$ is the number of bits in a word; i.e. if $A$ and $B$ are $w$–bit words, then $A +_w B$ is the word obtained by treating $A$ and $B$ as the binary representations of integers and computing their sum modulo $2^w$, where the result is constrained to lie between 0 and $2^w - 1$ inclusive |

# 5 Lightweight MACs based on block ciphers

## 5.1 General

This clause specifies a lightweight MAC algorithm that uses a secret key and an $n$-bit block cipher to calculate a $t$-bit MAC.

Annex A defines the object identifier which shall be used to identify the algorithm specified in Clause 5. Annex B provides numerical examples for the MAC algorithms in hexadecimal notation. Annex C gives the lightweight properties of the MAC algorithms described in this document.

## 5.2 LightMAC

### 5.2.1 General

LightMAC is a MAC algorithm that shall be used with any block cipher from ISO/IEC 29192-2 or ISO/IEC 18033-3. Users who wish to employ LightMAC[6] shall select:

— an $n$-bit block cipher $E$ from ISO/IEC 29192-2 or ISO/IEC 18033-3;

— a length $t$ in bits of the MAC;

— a counter size $s$, i.e. the number of bits allocated to represent the counter value, where $0 \le s < n$ .

The above parameters shall remain constant while using LightMAC under a given key. Different parameter sets should not be used under the same key.

NOTE 1 If any of the parameters above are modified while using a key, then no security can be guaranteed.

NOTE 2 Numerical examples, including for the cases $s = 8$ or 32 and $t = 64$, are listed in B.2.

LightMAC takes as input two independently generated block cipher keys $K_1$ and $K_2$, and a message $M$ of length at most $2^s(n-s)$ bits. LightMAC produces an output of length $t$ bits. LightMAC requires the following steps: padding, application of the block cipher, and truncation.

### 5.2.2 Step 1 (padding)

Let $m$ be the message input to LightMAC, and $d = |m| \bmod (n-s)$. Right-pad $m$ with a single ‘1’ bit, followed by $n$-$d$-1 ‘0’ bits. The result is denoted by $m'$.

### 5.2.3 Step 2 (application of the block cipher)

$m'$ shall be split into strings $m'_1, m'_2, \ldots, m'_\ell$, where ${m'_1}^{(n-s)}, {m'_2}^{(n-s)}, \ldots, {m'_{\ell-1}}^{(n-s)}$ are $n$-$s$ bit strings, $m'_\ell$ is an $n$ bit string, and ${m'_1}^{(n-s)}||{m'_2}^{(n-s)}||\ldots||m'_\ell = m'$. The string $S$ is computed using the following procedure.

$V \leftarrow 0^n$

For $i = 1$ to $\ell$-1:

$V \leftarrow E(K_1, \mathrm{I2BS}(i, s)||m'_i) \oplus V$

$V \leftarrow m_\ell' \oplus V$

$S \leftarrow E(K_2, V)$

Refer to Annex D for the specification of I2BS.

### 5.2.4 Step 3 (truncation)

The MAC of $t$ bits is derived by taking the least significant $t$ bits of the string $S$, i.e.:

$\mathrm{MAC} \leftarrow S|_t$

# 6 Lightweight MACs based on hash-functions

## 6.1 General

This clause specifies a lightweight MAC algorithm that uses a lightweight hash-function to compute a MAC.

 

Annex A defines the object identifier which shall be used to identify the algorithm specified in Clause 6. Annex B provides numerical examples for the MAC algorithms in hexadecimal notation. Annex C gives the lightweight properties of the MAC algorithms described in this document.

## 6.2 Tsudik's keymode

### 6.2.1 Requirements

Tsudik's keymode is a MAC algorithm that uses a hash-function. In order to use Tsudik's keymode[1], a lightweight hash-function $h$ shall be selected and agreed. The hash-function shall be chosen from amongst the lightweight hash-functions specified in ISO/IEC 29192-5:2016. An entity generating a MAC shall be equipped with a secret key $K$, which shall also be made available to all parties needing to verify the MAC.

NOTE 1 Tsudik's keymode is classified as lightweight because the number of calls to the underlying hash-function is typically smaller than generic-purpose hash-function-based MACs such as HMAC, as specified in ISO/IEC 9797-2.

NOTE 2 The reason why the underlying hash-function must be chosen from amongst those specified in ISO/IEC 29192-5 is described in Annex C.

NOTE 3 In the selection of the underlying hash-function used in Tsudik's keymode, it is up to the user to check its security against length extension attacks.

### 6.2.2 MAC calculation

To compute a MAC over the message $m$ using the Tsudik's keymode, the following operation is performed:

$$S \leftarrow h\ (K||m).$$

The MAC of $t$ bits is derived by taking the least significant $t$ bits of the string $S$, i.e.:

$$\text{MAC} \leftarrow S|_t.$$

# 7 Lightweight dedicated MACs

## 7.1 General

This clause specifies a lightweight dedicated MAC algorithm.

Annex A defines the object identifier which shall be used to identify the algorithm specified in Clause 7. Annex B provides numerical examples for the MAC algorithms in hexadecimal notation. Annex C gives the lightweight properties of the MAC algorithms described in this document.

## 7.2 Chaskey-12

### 7.2.1 General

Chaskey-12[8] is a lightweight MAC algorithm that processes an arbitrary-length message $m$ using a key $K$ of length 128 bits. It outputs a MAC of 128 bits or less.

NOTE 1 In the original proposal for the Chaskey algorithm[7], the number of rounds was set to 8, and the algorithm is referred to as Chaskey-8. Because of concerns that 8 rounds are insufficient to guarantee the required level of security, the scheme specified here has 12 rounds, and is thus referred to as Chaskey-12[8].

 

Chaskey-12 interchangeably considers an element $a$ of GF $(2^{128})$ as a 128-bit string $a[127]a[126]...a[0]$ and as the polynomial $a(x) = a[127]x^{127} + a[126]x^{126} + ... + a[0]$ with binary coefficients.

Let $f(x)$ be the irreducible polynomial $x^{128}+x^7+x^2+x+1$. To multiply two elements $a$ and $b$, they are represented as two polynomials $a(x)$ and $b(x)$, and $a(x)b(x)$ mod $f(x)$ is calculated. For example, how to multiply an element by $x$ is shown in Function 1. Note that a monomial $x$ corresponds to bitstring $0^{126}||10$. Note that bit string $0^{126}||11$ can be regarded as the decimal number 3.

**Function 1**: TimesTwo

function TimesTwo($a$)

if $a[127] = 0$

return $(a << 1) \oplus 0^{128}$

else

return $(a << 1) \oplus 0^{120}10000111$

When converting between 128-bit strings, 128-bit unsigned integers, and arrays of 32-bit unsigned integers, Chaskey-12 always uses little endian byte ordering. Inside every byte, bit numbering starts with the least significant bit.

Chaskey-12 requires the following steps: subkey derivation, padding, application of the permutation, and truncation.

### 7.2.2 Step 1 (subkey derivation)

Chaskey-12 takes a 128-bit key $K$, which should be chosen independently and uniformly at random from the entire key space. From $K$, two 128-bit subkeys $K_1$ and $K_2$ are derived, each by means of a 128-bit shift and a 128-bit bitwise exclusive-OR operation. The generation of the subkeys is defined in Function 2.

**Function 2:** Subkey derivation

function SubKeys($K$)

$K_1 \leftarrow$ TimesTwo($K$)

$K_2 \leftarrow$ TimesTwo($K_1$)

return $(K_1, K_2)$

### 7.2.3 Step 2 (padding)

Let $m$ be the message input to Chaskey-12, and let $d$ be the remainder when dividing the length of $m$ by 128. In case $d \neq 0$ or $|m| = 0$, then $m$ shall be padded by concatenating it with a single '1' bit, and then by 128-$d$-1 '0' bits. The padded version of the message $m$ is denoted by $m'$, and the Boolean variable ***padded*** is set to ***true***. In case $d = 0$ and $|m| \neq 0$, the message is not padded, so that $m' \leftarrow m$, and ***padded*** is set to ***false***.

### 7.2.4 Step 3 (application of the permutation)

The padded message $m'$ shall be split into strings $m_1', m_2', ..., m_\ell'$ of 128 bits each, so that $m_1'||m_2'||...||m_\ell' = m'$. To generate a MAC of at most 128 bits, Chaskey-12 iterates a 128-bit permutation π, as specified in Function 3 (See also Figure 1). In Figure 1, the application of the permutation (Step 2) when padded is false (top), and when padded is true. The round function of permutation π is specified in Function 4, the subkeys $K_1$ and $K_2$ are generated according to Step 1, and the message $m$ is padded to $m'$ according to Step 2. Optionally, the output S is truncated to t bits in Step 4.

 

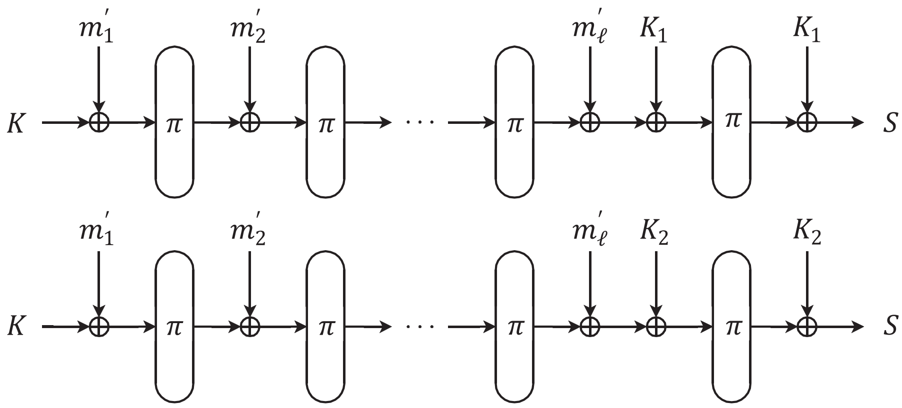


**Figure 1 — Application of the permutation**

In Chaskey-12, the permutation π consists of 12 applications of a round function, as described in Function 4. It operates on a 128-bit string, seen as an array of four 32-bit unsigned integers. The round function is also illustrated in Figure 2. Figure 2 shows a round of the Chaskey permutation π, defined as: $v_0||v_1||v_2||v_3 \leftarrow \pi(v_0||v_1||v_2||v_3)$.

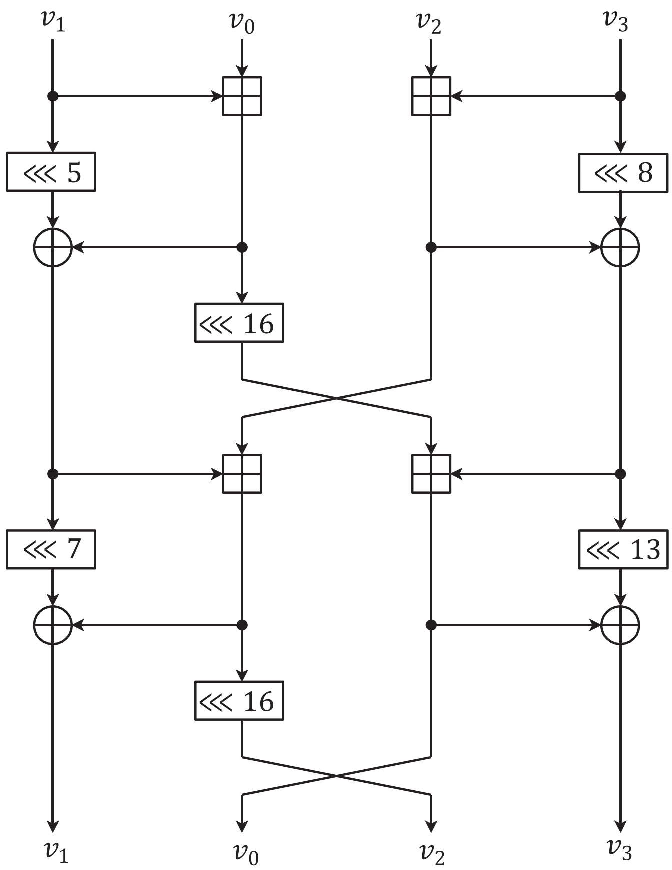


**Figure 2 — Round of the Chaskey permutation π**

**Function 3**: Application of the permutation

function Chaskey($K$, $K_1$, $K_2$, $m'$)

$V \leftarrow K$

For $j$=1 to $\ell$-1

 

$V \leftarrow \pi(V \oplus m_i')$

If *padded* = *false*

$L \leftarrow K_1$

Else

$L \leftarrow K_2$

EndIf

$S \leftarrow \pi(V \oplus m_\ell' \oplus L) \oplus L$

return $S$

**Function 4**: Permutation π

function $\pi(v)$

$(v_0, v_1, v_2, v_3) \leftarrow v$

For $i = 1$ to $r$

$v_0 \leftarrow v_0 +_{32} v_1$

$v_1 \leftarrow v_1 <<< 5$

$v_1 \leftarrow v_1 \oplus v_0$

$v_0 \leftarrow v_0 <<< 16$

$v_2 \leftarrow v_2 +_{32} v_3$

$v_3 \leftarrow v_3 <<< 8$

$v_3 \leftarrow v_3 \oplus v_2$

$v_0 \leftarrow v_0 +_{32} v_3$

$v_3 \leftarrow v_3 <<< 13$

$v_3 \leftarrow v_3 \oplus v_0$

$v_2 \leftarrow v_2 +_{32} v_1$

$v_1 \leftarrow v_1 <<< 7$

$v_1 \leftarrow v_1 \oplus v_2$

$v_2 \leftarrow v_2 <<< 16$

return $v_0 || v_1 || v_2 || v_3$

#### 7.2.5 Step 4 (truncation)

The MAC of $t$ bits is derived by taking the least significant $t$ bits of the string $S$, i.e.:

$\mathrm{MAC} \leftarrow S|_t$.

# Annex A
(normative)

# Object identifiers

This annex lists the object identifiers assigned to the lightweight MAC algorithms specified in this document.

```
--
-- Draft object identifiers of ISO/IEC 29192-6
--
LightweightMessageAuthenticationCodes {
 iso(1) standard(0) lightweight-cryptography(29192) part6(6)
 asn1-module(0) algorithm-object-identifiers(0)}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
EXPORTS ALL;
IMPORTS
 LightweightCryptographyIdentifier
 FROM LightweightCryptography-2 {iso(1) standard(0)
  lightweight-cryptography (29192) part2(2)
  asn1-module(0) algorithm-object-identifiers(0)}
 LightweightCryptographyIdentifier
 FROM LightweightCryptography-5 {iso(1) standard(0)
  lightweight-cryptography (29192) part5(5)
  asn1-module(0) algorithm-object-identifiers(0)};
OID ::= OBJECT IDENTIFIER -- Alias
-- Synonyms --
is29192-6 OID ::= {iso standard lightweight-cryptography(29192) part6(6)}
-- OID assignments --
light-bc-based-macs OID ::= {is29192-6 bc-based-macs(1)}
lightMAC OID ::= {light-bc-based-macs 1}

light-hf-based-macs OID ::= {is29192-6 hf-based-macs(2)}
tsudik OID ::= {light-hf-based-macs 1}

light-dedicated-macs OID ::= {is29192-6 dedicated-macs(3)}
chaskey-12 OID ::= {light-dedicated-macs 1}
-- note: assign any new OIDs above 3

LightweightMacAlgorithm ::= SEQUENCE {
   algorithm ALGORITHM.&id({LightweightMac}),
   parameters ALGORITHM.&Type({LightweightMac}{@algorithm}) OPTIONAL
}

LightweightMAC ALGORITHM ::= {
 { OID lightMAC PARMS MacParameters-1 } |
 { OID tsudik PARMS MacParameters-2 } |
 { OID chaskey-12 PARMS NullParms } ,
 ... -- expect additional algorithms --
}

-- MAC parameter types definitions
-- ================================
MacParameters-1 ::= SEQUENCE {
bcAlgo BlockCipher OPTIONAL,
t INTEGER (1..MAX)
}

MacParameters-2 ::= SEQUENCE {
hfAlgo HashFunction OPTIONAL,
t INTEGER (1..MAX)
}
BlockCipher ::=
LightweightCryptography-2.LightweightCryptographyIdentifier
```

```
HashFunction ::=
LightweightCryptography-5.LightweightCryptographyIdentifier

NullParms ::= NULL

-- Cryptographic algorithm identification --
ALGORITHM ::= CLASS {
    &id OBJECT IDENTIFIER UNIQUE,
    &Type OPTIONAL
}
   WITH SYNTAX { OID &id [PARMS &Type] }
END -- LightweightMessageAuthenticationCodes --
```

# Annex B
(informative)

# Numerical examples

## B.1 General

This annex provides numerical examples for each mechanism in hexadecimal notation.

## B.2 Numerical examples of LightMAC

The underlying block cipher: PRESENT-128 as specified in ISO/IEC 29192-2: 2012

**Case *s* = 8**

K1 = `00112233 44556677 8899aabb ccddeeff`

K2 = `833d3433 009f389f 2398e64f 417acf39`

- m = (0 octets) MAC = `c3c863d3 e954788b`
- m = `00` (1 octet) MAC = `021dff18 0cad82f9`
- m = `0001` (2 octets) MAC = `3f89441a a4492cb9`
- m = `000102` (3 octets) MAC = `858fffa6 d1d238cc`
- m = `00010203` (4 octets) MAC = `7aa56f92 0da21d54`
- m = `000102…04` (5 octets) MAC = `cead42e8 f4022b71`
- m = `000102…05` (6 octets) MAC = `939bdb08 9f993715`
- m = `000102…06` (7 octets) MAC = `0870a1d9 8336d9fe`
- m = `000102…07` (8 octets) MAC = `cf7fdf18 0e2494a2`
- m = `000102…08` (9 octets) MAC = `957872ff d8474fcc`
- m = `000102…09` (10 octets) MAC = `ef35a618 df40a363`
- m = `000102…0a` (11 octets) MAC = `ea690e1e c15c8817`
- m = `000102…0b` (12 octets) MAC = `fd0bc212 a16867a4`
- m = `000102…0c` (13 octets) MAC = `09bb6867 7241b04d`
- m = `000102…0d` (14 octets) MAC = `28eb9983 1d08b850`
- m = `000102…0e` (15 octets) MAC = `44cef839 dcd7c1cb`
- m = `000102…0f` (16 octets) MAC = `7f30f1f8 8d151ff5`
- m = `000102…10` (17 octets) MAC = `6af5747f ed140aa8`

- m = 000102…11 (18 octets) MAC = 11274bd5 394d388d
- m = 000102…12 (19 octets) MAC = 99c3a296 a8c8e548
- m = 000102…13 (20 octets) MAC = dc05c29e e8b9ab4e
- m = 000102…14 (21 octets) MAC = 95e0f4fa 3774a8cf
- m = 000102…15 (22 octets) MAC = 102ab033 7011a2f3
- m = 000102…16 (23 octets) MAC = 88d68a1c 48bbc0b7
- m = 000102…17 (24 octets) MAC = 5f028d39 6326567a
- m = 000102…18 (25 octets) MAC = 6f30b07a 025fa7ef
- m = 000102…19 (26 octets) MAC = e5ecbaba 994a36bc
- m = 000102…1a (27 octets) MAC = 409b49af 04f3c184
- m = 000102…1b (28 octets) MAC = dc88786e 74c298ad
- m = 000102…1c (29 octets) MAC = 76e6526a f8c125ce
- m = 000102…1d (30 octets) MAC = 942cfea1 68710b4c
- m = 000102…1e (31 octets) MAC = 6f5703a4 048145cb
- m = 000102…1f (32 octets) MAC = 4734b059 b872d41f
- m = 000102…20 (33 octets) MAC = 6f8f3297 4cb44284
- m = 000102…21 (34 octets) MAC = 12d537fe 00bb6046
- m = 000102…22 (35 octets) MAC = cd00fa52 a649e50a
- m = 000102…23 (36 octets) MAC = 5a0cdf1f 6e11f546
- m = 000102…24 (37 octets) MAC = 36c79cbe 956ed91a
- m = 000102…25 (38 octets) MAC = 9c854532 7b31c585
- m = 000102…26 (39 octets) MAC = a9a2eeed 33cee786
- m = 000102…27 (40 octets) MAC = b3343218 da6aa666
- m = 000102…28 (41 octets) MAC = 4f34e65d ba08b06a
- m = 000102…29 (42 octets) MAC = eebe3228 f7f21ed4
- m = 000102…2a (43 octets) MAC = 326d89ce bad3f651
- m = 000102…2b (44 octets) MAC = ff99dbfc 72b919d4
- m = 000102…2c (45 octets) MAC = 0b7224ab c148de6c
- m = 000102…2d (46 octets) MAC = 97928b8d f5c5f048
- m = 000102…2e (47 octets) MAC = 9e6e039a a6209f07
- m = 000102…2f (48 octets) MAC = a10deb90 41205b21

 

- m = 000102…30 (49 octets) MAC = 0e8208be 90e1e10b
- m = 000102…31 (50 octets) MAC = de0e2416 4f616f79
- m = 000102…32 (51 octets) MAC = 1761603e a4fbbd80
- m = 000102…33 (52 octets) MAC = eda5d05f b67b528d
- m = 000102…34 (53 octets) MAC = 48f4d47e 485b47d2
- m = 000102…35 (54 octets) MAC = df07346b bcff6eac
- m = 000102…36 (55 octets) MAC = 52d6b140 588dd5ff
- m = 000102…37 (56 octets) MAC = cc680cb4 b8b8a9b0
- m = 000102…38 (57 octets) MAC = 100160dc fb743c20
- m = 000102…39 (58 octets) MAC = d2ee268e c27309d6
- m = 000102…3a (59 octets) MAC = dcdcba02 e6b8a4b3
- m = 000102…3b (60 octets) MAC = 100e1ae6 ced7fede
- m = 000102…3c (61 octets) MAC = d394ff09 dea2010c
- m = 000102…3d (62 octets) MAC = 6e4b1fa6 30d3acc9
- m = 000102…3e (63 octets) MAC = 431f7d96 7c0bc59e

**Case $s$ = 32**

$K1$ = 01234567 89abcdef fedcba98 76543210

$K2$ = 9cf35e82 f26719c4 f91cf900 cc2cbcc1

- m = (0 octets)

  MAC = 8634d35b bcdbbeb9
- m = 00000001 00020003 … 007e007f 00 (257 octets)

  MAC = d60a9a88 90c26393
- m = 00000001 00020003 … 00fe00ff 0100 (514 octets)

  MAC = f93128da b71ecf5c
- m = 00000001 00020003 … 017e017f 018001 (771 octets)

  MAC = e9067e6c 774de5e2
- m = 00000001 00020003 … 01fe01ff 02000201 (1 028 octets)

  MAC = 774d44bb 7ce920a4
- m = 00000001 00020003 … 02800281 02 (1 285 octets)

  MAC = 38a9c371 00b908ff
- m = 00000001 00020003 … 03000301 0302 (1 542 octets)

  MAC = c88d0fb7 32647299

- m = `00000001 00020003 … 03800381 038203` (1 799 octets)

  MAC = `d2a1a618 3daacd78`
- m = `00000001 00020003 … 04000401 04020403` (2 056 octets)

  MAC = `481dd001 e71a3585`

## B.3 Numerical example of Tsudik's keymode

The underlying hash-function: Lesamnta-LW as specified in ISO/IEC 29192-5:

Case $t$ = 128

```
K  4c4c4c4c 4c4c4c4c 4c4c4c4c 4c4c4c4c
Message  =4c4c4c4c 4c4c4c4c 4c4c4c4c 4c4c4c4c
MAC  =  e01f6580 0eea165f 8c85b688 c36afca3
```

## B.4 Numerical examples of Chaskey-12

Case $t$ = 64

```
K  00112233  44556677  8899aabb  ccddeeff
K1 87224466  88aaccee  10335577  99bbddff
K2 894588cc  105599dd  2166aaee  3277bbff
m = (0 octets)                    MAC = dd3e1849 d6824555
m = 00 (1 octet)                  MAC = ed1da89e c93179ca
m = 0001 (2 octets)               MAC = 98fe20a3 43cd666f
m = 000102 (3 octets)             MAC = f6f418ac dd7d9fa1
m = 00010203 (4 octets)           MAC = 4cf04960 099949f3
m = 000102…04 (5 octets)          MAC = 75c83252 653d3b57
m = 000102…05 (6 octets)          MAC = 964b0461 fbe92273
m = 000102…06 (7 octets)          MAC = 141fa08b bf399636
m = 000102…07 (8 octets)          MAC = 412d98ed 936d4ab2
m = 000102…08 (9 octets)          MAC = fb0d98bc 70e305f9
m = 000102…09 (10 octets)         MAC = 36f88e1f da86c8ab
m = 000102…0a (11 octets)         MAC = 4d1a1815 868a5aa8
m = 000102…0b (12 octets)         MAC = 7a7912c1 999eae81
m = 000102…0c (13 octets)         MAC = 9ca11137 b4a34601
m = 000102…0d (14 octets)         MAC = 7905142f 3be77e67
m = 000102…0e (15 octets)         MAC = 6a3ee3d3 5c043397
```

```
m = 000102…0f (16 octets)          MAC = d13970d7 be9b2350
m = 000102…10 (17 octets)          MAC = 32acd914 bfda3bc8
m = 000102…11 (18 octets)          MAC = 8a58d816 cb7a1483
m = 000102…12 (19 octets)          MAC = 03f4d666 38efad8d
m = 000102…13 (20 octets)          MAC = f9932237 ff05e831
m = 000102…14 (21 octets)          MAC = f5fedb13 4862b471
m = 000102…15 (22 octets)          MAC = 8bb55486 f38d57ea
m = 000102…16 (23 octets)          MAC = 8a3acb94 b5ad591c
m = 000102…17 (24 octets)          MAC = 7ce37087 23f7495f
m = 000102…18 (25 octets)          MAC = f42f3d2f 405710c2
m = 000102…19 (26 octets)          MAC = b3933a16 7e5636ac
m = 000102…1a (27 octets)          MAC = 899a7945 423a5e1b
m = 000102…1b (28 octets)          MAC = 65e12df5 a695fac8
m = 000102…1c (29 octets)          MAC = b82449d8 c8a06ae9
m = 000102…1d (30 octets)          MAC = a850dfba defa4229
m = 000102…1e (31 octets)          MAC = fd42c39d 08ab71a0
m = 000102…1f (32 octets)          MAC = b465c241 2610bf84
m = 000102…20 (33 octets)          MAC = 89c4a9dd b53e6991
m = 000102…21 (34 octets)          MAC = 5a9af91e b095d331
m = 000102…22 (35 octets)          MAC = 8e54914c 151e46b0
m = 000102…23 (36 octets)          MAC = fab8ab0b 5beaaec6
m = 000102…24 (37 octets)          MAC = 60ad906a cd06c823
m = 000102…25 (38 octets)          MAC = 6b1e6bc2 426dad17
m = 000102…26 (39 octets)          MAC = 90328fd2 59889a8f
m = 000102…27 (40 octets)          MAC = f0f7815e e6f3d516
m = 000102…28 (41 octets)          MAC = 97e7e2ce bea826b8
m = 000102…29 (42 octets)          MAC = b0fa1845 f72a76d6
m = 000102…2a (43 octets)          MAC = a468bdfc df0aa9c7
m = 000102…2b (44 octets)          MAC = da84e113 38387da7
m = 000102…2c (45 octets)          MAC = b30d5ead 8e39f2bc
m = 000102…2d (46 octets)          MAC = 178a43d2 a008503e
```

```
m = 000102…2e (47 octets)          MAC = 6dfaa705 a8a06c70

m = 000102…2f (48 octets)          MAC = aa047f07 c5ae8db4

m = 000102…30 (49 octets)          MAC = 305bbb42 0c5d5ecc

m = 000102…31 (50 octets)          MAC = 08328031 59750f49

m = 000102…32 (51 octets)          MAC = 9080254f b79bab1a

m = 000102…33 (52 octets)          MAC = 61c285ca 245774a4

m = 000102…34 (53 octets)          MAC = 2aae035c fb61f97a

m = 000102…35 (54 octets)          MAC = f5289075 c9ab39e5

m = 000102…36 (55 octets)          MAC = e65c4237 32dae795

m = 000102…37 (56 octets)          MAC = 4b22cf0d 9da8de3d

m = 000102…38 (57 octets)          MAC = 2626ea2f a1f9abcf

m = 000102…39 (58 octets)          MAC = d1e17e6e c4a88da6

m = 000102…3a (59 octets)          MAC = 16574428 27ff640a

m = 00102…3b (60 octets)           MAC = fd155a40 df15f630

m = 000102…3c (61 octets)          MAC = ffeb596f 299f58b2

m = 000102…3d (62 octets)          MAC = be4ee4ed 3975df87

m = 000102…3e (63 octets)          MAC = fc7f9df7 991b87bc
```

# Annex C
(informative)

# Security information and feature tables

## C.1 General

This annex gives the lightweight properties of the MACs described in this document.

## C.2 Information on LightMAC

### C.2.1 Lightweight properties of LightMAC

A lightweight property of LightMAC is that the message length has no effect on the security bound, allowing an order of magnitude more data to be processed per key. Furthermore, LightMAC has almost no overhead over the block cipher, and is parallelizable. As a result, LightMAC not only offers compact authentication for resource-constrained platforms, but also allows high-performance parallel implementations. The use of non-lightweight block ciphers (from ISO/IEC 18033-3 rather than from ISO/IEC 29192-2) does not affect the lightweight properties of LightMAC.

According to ISO/IEC 29192-1, a lightweight property of LightMAC is that it requires small resources with respect to cycle/byte shown in Table C.1.

**Table C.1 — Baseline performance of ciphers PRESENT and AES on Skylake (AVX2, AES-NI)[6]**

| The underlying block cipher | encryption | key schedule |
|---|---|---|
| | [cycles/byte] | [cycles] |
| PRESENT (table-based) | 57,83 | 353 |
| PRESENT (8 blocks bitsliced) | 11,23 | 790 |
| AES (AES-NI, serial) | 2,57 | 116 |
| AES (AES-NI, pipelined) | 0,63 | 116 |

The lightweight properties are independent of the underlying block cipher used in LightMAC.

For LightMAC, $t = 64$ is allowed (see B.2).

### C.2.2 Parameter *s*

The parameter $s$ determines the maximum message length that can be processed with LightMAC and fixes the efficiency of the algorithm as measured in block cipher calls per message block. Specifically, LightMAC is faster for smaller $s$, but the maximum message length it can process decreases as a result.

## C.3 Information on the underlying hash-functions to be used in Tsudik's mode

The Tsudik keymode described in Clause 6 requires a hash-function which is collision-resistant and resistant to length-extension attacks. To ensure security against the length extension attack, appropriate property on the underlying hash-function is required.

In comparison with the most widely used mode for converting a hash-function in to a MAC, HMAC, Tsudik's mode is much simpler. In general, any mode of operation proved to be indifferentiable from a random oracle should be suitable for Tsudik's mode. This includes PHOTON and SPONGENT from ISO/IEC 29192. Bibliographic reference [1] presents a proof that Lesamnta-LW from ISO/IEC 29192 can be used in Tsudik's mode. Therefore, the hash-functions of ISO/IEC 29192-5 can be used with this mode of Tsudik. However, ISO/IEC 10118-3 hash-functions cannot be used with this mode of Tsudik.

According to ISO/IEC 29192-1, the lightweight property of Tsudik mode is that it requires small resources with respect to execution time. More specifically, for short message, the number of calls to the underlying hash-function can be one. This property is independent of the underlying hash-function.

## C.4 Information on the usage of Chaskey-12

The security of MAC algorithms is affected by the number of message blocks that are processed under the same key. For Chaskey-12, as in most block-cipher-based algorithms, the number of blocks processed under the same key should stay well below the security bound, which is $2^{64}$ blocks for a 128-bit block cipher. For example, the key may be refreshed before $2^{48}$ message blocks of 128 bits are processed, which corresponds to four pebibytes[2] of data[7]. IEC 80000-13[2] defines standard binary prefixes used to denote powers of 1 024 as $1\,024^5$ (pebi-).

According to ISO/IEC 29192-1, the lightweight property of Chaskey-12 is that Chaskey-12 is only 15 % to 30 % slower than Chaskey-8, depending on the target platform[8] and it requires small resources with respect to data, ROM size, and cycle/byte shown in Table C.2.

**Table C.2 — Benchmark results for Chaskey-8 on Cortex-M0/M4[7]**

| Microcontroller | Data [byte] | ROM size [byte] | Speed [cycles/byte] |
|---|---|---|---|
| Speed optimized | | | |
| Cortex-M0 | 16 | 1 308 | 21,3 |
| Cortex-M4 | 16 | 908 | 10,6 |
| | 128 | 908 | 7,0 |
| Size optimized | | | |
| Cortex-M0 | 16 | 414 | 21,8 |
| Cortex-M4 | 16 | 402 | 16,1 |
| | 128 | 402 | 11,2 |

# Annex D
(informative)

# Specification of I2BS

## D.1 General

This annex gives the specification of the function I2BS (Integer to Bit String conversion).

## D.2 I2BS

To represent a non-negative integer $x$ as a bit string of length $g$ ($g$ has to be such that $2^g > x$), the integer shall be written in its unique binary representation [see Formula (D.1)]:

$$x = 2^{g-1} x_{g-1} + 2^{g-2} x_{g-2} + \ldots + 2x_1 + x_0 \tag{D.1}$$

where $0 \leq x_i < 2$ (note that one or more leading digits will be zero if $x < 2^{g-1}$). The bit string I2BS($x$, $g$) shall be as Formula (D.2):

$$x_{g-1} x_{g-2} \ldots x_0 \tag{D.2}$$

# Bibliography


[1] Hirose S., Ideguchi K., Kuwakado H., Owada T., Preneel B., Yoshida H., An AES Based 256-bit Hash Function for Lightweight Applications: Lesamnta-LW", IEICE Transactions **95-A** (1): 89-99 (2012)

[2] IEC 80000-13, *Quantities and units — Part 13: Information science and technology*

[3] ISO/IEC 9797-1, *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*

[4] ISO/IEC 10118-1, *Information technology — Security techniques — Hash-functions — Part 1: General*

[5] ISO/IEC 10118-3, *IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions*

[6] Luykx A., Preneel B., Tischhauser E., Yasuda K. *A MAC Mode for Lightweight Block Ciphers",* In Proceedings of Fast Software Encryption - FSE 2016, Lecture Notes in Computer Science **volume 9783**, pp. 43-59, Springer-Verlag, 2016

[7] Mouha N., Mennink B., Van Herrewege A., Watanabe D., Preneel B., Verbauwhede I. *Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers",* In Proceedings of Selected Areas in Cryptography - SAC 2014, Lecture Notes in Computer Science **volume 8781**, pp. 306-323, Springer-Verlag, 2014

[8] Mouha N., Chaskey: a MAC Algorithm for Microcontrollers – Status Update and Proposal of Chaskey-12 –", Cryptology ePrint Archive, Report 2015/1182, 2015

[9] Tsudik G., Message Authentication with One-Way Hash Functions", ACM, Computer Communications Review, vol. **22**, no.5, pp.29-38, 1992


 

**ICS 35.030**

Price based on 20 pages