INTERNATIONAL STANDARD

ISO/IEC 9797-3

First edition
2011-11-15

Information technology — Security techniques — Message Authentication Codes (MACs) —

Part 3: Mechanisms using a universal hash-function

Technologies de l'information — Techniques de sécurité — Codes d'authentification de message (MAC) —

Partie 3: Mécanismes utilisant une fonction de hachage universelle

Reference number
ISO/IEC 9797-3:2011(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 9797-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

ISO/IEC 9797 consists of the following parts, under the general title *Information technology — Security techniques — Message Authentication Codes (MACs)*:

— *Part 1: Mechanisms using a block cipher*

— *Part 2: Mechanisms using a dedicated hash-function*

— *Part 3: Mechanisms using a universal hash-function*

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.

This part of ISO/IEC 9797 specifies four MAC algorithms using universal hash-functions: UMAC, Badger, Poly1305-AES and GMAC.

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 strength of the data integrity mechanism and message authentication mechanism is dependent on the length (in bits) and secrecy of the key, on the length (in bits) of a hash-code produced by the hash-function, on the strength of the hash-function, on the length (in bits) of the MAC, and on the specific mechanism.

NOTE A general framework for the provision of integrity services is specified in ISO/IEC 10181-6[7].

Information technology — Security techniques — Message Authentication Codes (MACs) —

Part 3:
Mechanisms using a universal hash-function

1 Scope

This part of ISO/IEC 9797 specifies the following MAC algorithms that use a secret key and a universal hash-function with an *n*-bit result to calculate an *m*-bit MAC based on the block ciphers specified in ISO/IEC 18033-3 and the stream ciphers specified in ISO/IEC 18033-4:

a) UMAC;

b) Badger;

c) Poly1305-AES;

d) GMAC.

2 Normative references

The following referenced documents are indispensable for the application 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 9797-1, *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*

ISO/IEC 18031, *Information technology — Security techniques — Random bit generation*

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

ISO/IEC 18033-4, *Information technology — Security techniques — Encryption algorithms — Part 4: Stream ciphers*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9797-1 and the following apply.

3.1
empty string
string of symbols of length zero

3.2
key
sequence of symbols that controls the operation of a cryptographic transformation

3.3
nonce
number used once

3.4
prime number
positive integer greater than 1 which has no integer divisors other than 1 and itself

3.5
tag
result of a MAC algorithm, adjoined to a possibly encrypted message to provide integrity protection

3.6
universal hash-function
function mapping strings of bits to fixed-length strings of bits, indexed by a parameter called the key, satisfying the property that for all distinct inputs, the probability over all keys that the outputs collide is small

NOTE Universal hash-functions were introduced by Carter and Wegman[4], and their application in MAC algorithms was first described by Wegman and Carter [10].

4 Symbols and abbreviated terms

The following notation is used in this part of ISO/IEC 9797:

bit(S,n) Returns the integer 1 if the n^{th} bit of the string S is 1, otherwise returns the integer 0 (indices begin at 1).

bitlength(S) Length of a string S in bits.

bitstr2uint(S) The non-negative integer whose binary representation is the string S. More formally, if S is t bits long then bitstr2uint(S) = 2^{t-1} * bit(S,1) + 2^{t-2} * bit(S,2) + ... + 2^{1} * bit(S,t-1) + bit(S,t).

NOTE Bit strings are treated big-endian, i.e. the first bit is the most significant.

blocklen Block length of the underlying block cipher in octets.

ceil Rounding-up operation, i.e. if x is a floating-point number, then ceil(x) is the smallest integer n with $n \geq x$.

Enc(K, X) Encryption of a plaintext block X under a key K using a block cipher Enc.

floor Rounding-down operation, i.e. if x is a floating-point number, then floor(x) is the largest integer n with $n \leq x$.

H Hash value.

K Master key.

K_E Encryption key.

K_H Hash key.

keylen Block cipher key length in octets.

$\log_2$ Binary logarithm function.

M Message.

MAC Message authentication code.

max Largest value amongst those given as argument.

N Nonce.

octetlength(*S*) Length of a string *S* in octets (where *S* is assumed to have bitlength a multiple of 8).

octetstr2uint(*S*) The non-negative integer defined as $S[0] + 2^8 * S[1] + 2^{16} * S[2] + ... + 2^{8n-8} * S[n-1]$, where n = octetlength(*S*).

NOTE Octet strings are treated little-endian, i.e. the first octet is the least significant.

prime(*n*) Largest prime number smaller than 2^n, for any positive integer *n*.

NOTE The prime numbers used in this part of ISO/IEC 9797 are listed in Table 1.

Table 1 — Prime numbers

n	prime(*n*)	prime(*n*) in hexadecimal format
32	$2^{32} - 5$	0x FFFFFFFB
36	$2^{36} - 5$	0x 0000000F FFFFFFFB
64	$2^{64} - 59$	0x FFFFFFFF FFFFFFC5
128	$2^{128} - 159$	0x FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF61
130	$2^{130} - 5$	0x 00000003 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFB

S[*i*] The *i*-th octet of the string *S* (indices begin at 0).

NOTE The specification for UMAC in 6.2 uses a starting index of 1 rather than 0.

S[*i*...*j*] The substring of *S* consisting of octets *i* through *j*.

taglen Length of the tag, in octets.

uint2bitstr(*x*,*n*) The *n*-octet string *S* such that bitstr2uint(*S*) = *x*.

uint2octetstr(*x*,*n*) The *n*-octet string *S* such that *x* = octetstr2uint(*S*).

$X|_s$ Left-truncation of the block of bits *X*: if *X* has bit-length greater than or equal to *s*, then $X|_s$ is the *s*-bit block consisting of the left-most *s* bits of *X*.

$X|^s$ Right-truncation of the block of bits *X*: if *X* has bit-length greater than or equal to *s*, then $X|^s$ is the *s*-bit block consisting of the right-most *s* bits of *X*.

X>>1 Right shift of a block of bits *X* by one position: the leftmost bit of *Y* = *X*>>1 will always be set to zero.

|*X*| The length of *X* in bits.

zeropad(*S*,*n*) For positive integer *n*, the string *S* is padded with zero-bits to the nearest positive multiple of *n* octets. Formally, zeropad(*S*,*n*) = *S* || *T*, where T is the shortest string of zero-bits (possibly empty) so that *S* || T is non-empty and *n* divides octetlength(*S* || *T*).

$\oplus$ Bit-wise exclusive-OR operation on bit-strings. If A, B are strings of the same length then $A \oplus B$ is the string equal to the bit-wise logical exclusive-OR of A and B.

$\wedge$ Bit-wise logical AND operation on bit-strings. If A, B are strings of the same length then $A \wedge B$ is the string equal to the bit-wise logical AND of A and B.

$+_{32}$ Addition of two 32-bit strings, resulting in a 32-bit string. More formally, $S +_{32} T$ = uint2bitstr(bitstr2uint(S) + bitstr2uint(T) mod 2^{32}, 4).

$+_{64}$ Addition of two 64-bit strings, resulting in a 64-bit string. More formally, $S +_{64} T$ = uint2bitstr(bitstr2uint(S) + bitstr2uint(T) mod 2^{64}, 8).

$*$ Multiplication operator on integers.

$*_{64}$ Multiplication of two 64-bit strings, resulting in a 64-bit string. More formally, $S *_{64} T$ = uint2bitstr(bitstr2uint(S) * bitstr2uint(T) mod 2^{64}, 8).

NOTE The operations $+_{32}$, $+_{64}$ and $*_{64}$ correspond well with the addition and multiplication operations that are performed efficiently by modern computers.

|| Concatenation of two bit strings. If A and B are bit strings of lengths a and b respectively, then $A \,||\, B$ is the bit string of length $a+b$ whose left most (first) a bits are the bits of A, and whose rightmost (last) b bits are the bits of B.

0^n String consisting of n zero-bits.

1^n String consisting of n one-bits.

{ } A bit-string with zero length.

$\bullet$ Multiplication in the field GF(2^{128}). The defining polynomial that determines the representation of GF(2^{128}) is $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$.

NOTE Let U and V be 128-bit blocks. Then the 128-bit block $W = U \bullet V$ can be computed as follows:

a) Let $W = 0^{128}$ and $Z = U$.

b) For i = 1, 2, …, 128, perform the following two steps:

1) If bit(V,i) = 1 then let $W = W \oplus Z$;

2) If bit(Z,128) = 0 then let Z = Z>>1; otherwise let Z = (Z>>1) $\oplus$ (11100001 || 0^{120}).

Variables in capital letters denote strings; variables in small letters are integers.

5 General model

Message authentication codes based on universal hashing makes use of an encryption algorithm (block cipher or stream cipher). This type of message authentication codes has the special property that their security can be proven under the assumption that the encryption algorithm is secure.

MAC algorithms based on universal hashing require a master key K, a message M and a nonce value N as input. A MAC is computed using the following sequence of steps:

1) Key preprocessing. The master key K is used to generate a hash key K_H and an encryption key K_E.

2) Message preprocessing. The input message M is encoded into the necessary input format for the hash-function.

3) Message hashing. The encoded message is hashed under the control of the hash key K_H, using a universal hash-function. The result is a hash value H of fixed, short length.

4) Finalization. The hash value H is encrypted under the control of the encryption key K_E. The result is the message authentication code MAC.

For all mechanisms presented in this part of ISO/IEC 9797, the length of the input message is expected to consist of an integer number of octets.

NOTE For all universal-hash based MAC algorithms, it is of utmost importance that a different nonce is used for each new message that is authenticated under the same key. If this security requirement is not met, the security of the algorithm is severely reduced.

6 Mechanisms

6.1 Introduction

In this clause, four mechanisms using a universal hash-function are specified.

6.2 UMAC

6.2.1 Description of UMAC

UMAC is a family of four MAC algorithms optimized for different output bit-lengths, denoted by UMAC-32, UMAC-64, UMAC-96, and UMAC-128. UMAC can be used with any block cipher from ISO/IEC 18033-3. If the block cipher used has key length $|K|$ bits and block length $|B|$ bits, then UMAC uses a $|K|$-bit key K, and the length of the nonce N is between 8 and $|B|$ bits. Depending on which member of the UMAC family is used, the length of the MAC produced is 32, 64, 96, or 128 bits. This is represented by the parameter *taglen*, which can be 4, 8, 12 or 16 octets, respectively. The length of the input message shall be less than 2^{67} octets. The message input to the UMAC function shall contain a whole number of octets, i.e. the bitlength shall be a multiple of 8. If the bitlength is not a multiple of 8, this mechanism shall not be used.

NOTE 1 The version of UMAC specified here must not be confused with earlier versions of the UMAC algorithm, e.g.[2].

NOTE 2 If the input to the MAC function contains a whole number of bytes, then the function specified here is identical to that described in RFC 4418[6].

6.2.2 Requirements

Before the use of UMAC, the following parameters shall be agreed upon:

— A block cipher standardized in ISO/IEC 18033-3. The choice of a block cipher determines the key length $|K|$ and the block length $|B|$.

— A tag length, *taglen*, which shall be either 4, 8, 12 or 16 octets.

— The length of the nonce, which shall be between 8 and $|B|$ bits.

6.2.3 Notation and auxiliary functions

6.2.3.1 Operations on strings

In contrast to the remainder of this part of ISO/IEC 9797, the specification of UMAC uses a starting index of 1 when numbering elements in a sequence. Thus, for UMAC, $S[i]$ denotes the ith octet of the string S, where $i \geq 1$.

6.2.3.2 Auxiliary function KDF

This key-derivation function generates pseudorandom bits. It returns *numoctets* output octets.

INPUT: Master key *K*, (*keylen*)-octet string
index, a non-negative integer less than 2^{64}
numoctets, a non-negative integer less than 2^{64}

OUTPUT: *Y*, (*numoctets*)-octet string

a) *n* = ceil(*numoctets* / *blocklen*)

b) Set *Y* to the empty string

c) for i = 1 to *n* do

1) *T* = uint2bitstr(*index*, *blocklen*-8) || uint2bitstr(*i*, 8)

2) *T* = Enc(*K*, *T*)

3) *Y* = *Y* || *T*

d) *Y* = *Y*[1...*numoctets*]

e) Output *Y*

NOTE The key-derivation function KDF uses a block cipher in counter mode as defined in ISO/IEC 10116[8].

6.2.3.3 Auxiliary function PDF

This pad-derivation function takes a key and a nonce and returns a pseudorandom padding sequence for use in tag generation. A pad of length 4, 8, 12, or 16 octets can be generated.

INPUT: Master key *K*, (*keylen*)-octet string
Nonce *N*, string of length 1 to *blocklen* octets
Tag length *taglen*, the integer 4, 8, 12 or 16

OUTPUT: *Y*, (*taglen*)-octets string

a) PDFnonce = *N*

b) if (*taglen* = 4 or *taglen* = 8)

1) *index* = bitstr2uint(*N*) mod (*blocklen*/*taglen*)

2) *PDFnonce* = *N* ⊕ uint2bitstr(*index*, octetlength(*N*))

c) padlen = blocklen - octetlength(PDFnonce)

d) PDFnonce = PDFnonce || $0^{padlen*8}$

e) *K'* = KDF(*K*, 0, *keylen*)

f) T = Enc(K', PDFnonce)

g) if (*taglen* = 4 or *taglen* = 8)

1) Y = T[(index * taglen) + 1 ... (index * taglen) + taglen]