BS ISO/IEC 9797-2:2021
Incorporating corrigendum November 2024

BSI Standards Publication

Information security — Message authentication codes (MACs)

Part 2: Mechanisms using a dedicated hash-function

National foreword

This British Standard is the UK implementation of ISO/IEC 9797-2:2021, incorporating corrigendum November 2024. It supersedes BS ISO/IEC 9797-2:2011, which is withdrawn.

The start and finish of text introduced or altered by corrigendum is indicated in the text by tags. Text altered by ISO/IEC corrigendum November 2024 is indicated in the text by ⟦AC1⟩ ⟨AC1⟧.

The UK participation in its preparation was entrusted to Technical Committee IST/33/2, Cryptography and Security Mechanisms.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2024

ISBN 978 0 539 29153 7

ICS 35.030

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2021.

Amendments/corrigenda issued since publication

Date	Text affected
31 December 2024	Implementation of ISO/IEC corrigendum November 2024

BS ISO/IEC 9797-2:2021

INTERNATIONAL STANDARD

ISO/IEC 9797-2

Third edition
2021-06

Technical corrigendum
2024-11

Information security — Message authentication codes (MACs) —

Part 2: Mechanisms using a dedicated hash-function

Sécurité de l'information — Codes d'authentication de message (MAC) —

Partie 2: Mécanismes utilisant une fonction de hachage dédiée

Reference number
ISO/IEC 9797-2:2021(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: 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.

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 or www.iec.ch/members_experts/refdocs).

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 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. In the IEC, see www.iec.ch/understanding-standards.

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

This third edition cancels and replaces the second edition (ISO/IEC 9797-2:2011), which has been technically revised.

The main changes compared to the previous edition are as follows:

— Using dedicated hash-function 17 for MAC Algorithms 1 and 3 has been added;

— Using dedicated hash-functions 11, 12, 13 to 16, and 17 for MAC Algorithm 2 has been added;

— MAC Algorithm 4 based on Keccak, a primitive in the definition of dedicated hash-functions 13 to 16 has been added.

A list of all parts in the ISO/IEC 9797 series can be found on the ISO and IEC websites.

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 and www.iec.ch/national-committees.

INTERNATIONAL STANDARD **ISO/IEC 9797-2:2021(E)**

Information security — Message authentication codes (MACs) —

Part 2: Mechanisms using a dedicated hash-function

1 Scope

This document specifies MAC algorithms that use a secret key and a hash-function (or its round-function or sponge function) to calculate an *m*-bit MAC. These mechanisms can be used as data integrity mechanisms to verify that data has not been altered in an unauthorized manner.

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

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 10118-3, *IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions*

3 Terms and definitions

For the purposes of this document, the following terms and definitions 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
bit-string of length L_1, i.e. the length of the first input to the round-function

[SOURCE: ISO/IEC 10118-3:2018, 3.1]

3.2
entropy
measure of the disorder, randomness or variability in a closed system

Note 1 to entry: The entropy of a random variable X is a mathematical measure of the amount of information provided by an observation of X.

[SOURCE: ISO/IEC 18031:2011, 3.11]

3.3
input data string
string of bits which is the input to a MAC algorithm

3.4
hash-code
string of bits which is the output of a hash-function

[SOURCE: ISO/IEC 10118-1:2016, 3.3]

3.5
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 infeasibility depends on the specific security requirements and environment. Refer to ISO/IEC 10118-1:2016, Annex C.

[SOURCE: ISO/IEC 10118-1:2016, 3.4, modified — "feasability" in Note 1 to entry changed to "infeasability".]

3.6
initializing value
value used in defining the starting point of a hash-function

[SOURCE: ISO/IEC 10118-1:2016, 3.5, modified — Note 1 to entry removed]

3.7
MAC algorithm key
key that controls the operation of a MAC algorithm

[SOURCE: ISO/IEC 9797-1:2011, 3.8]

3.8
message authentication code
MAC
string of bits which is the output of a MAC algorithm

Note 1 to entry: A MAC is sometimes called a cryptographic check value (see for example ISO 7498-2[1]).

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

3.9
message authentication code algorithm
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 the set of input strings and corresponding function values, where the value of the ith input string may have been chosen after observing the value of the first i-1 function values (for integer $i > 1$)

Note 1 to entry: A MAC algorithm is sometimes called a cryptographic check function (see for example ISO 7498-2).

Note 2 to entry: Computational infeasibility depends on the user's specific security requirements and environment.

[SOURCE: ISO/IEC 9797-1:2011, 3.10 modified — "feasability" in Note 2 to entry changed to "infeasability".]

3.10
output transformation
function that is applied at the end of the MAC algorithm, before the truncation operation

[SOURCE: ISO/IEC 9797-1:2011, 3.12]

3.11
padding
appending extra bits to a data string

[SOURCE: ISO/IEC 10118-1:2016, 3.7]

3.12
round-function
function ϕ (..,..) that transforms two binary strings of lengths L_1 and L_2 to a binary string of length L_2 that is used iteratively as part of a hash-function, where it combines a data string of length L_1 with the previous output of length L_2 or the initializing value

Note 1 to entry: The literature on this subject contains a variety of terms that have the same or similar meaning as round-function. Compression function and iterative function are some examples.

[SOURCE: ISO/IEC 10118-1:2016, 3.8]

3.13
security strength
number associated with the amount of work required to break a cryptographic algorithm or system and specified in bits such that security strength s bits implies the required number of operations is 2^s

Note 1 to entry: Computationally infeasible in 3.2, 3.6 and 3.10 implies the security strength is at least 112 bits. Refer to ISO/IEC 10118-1:2016, Annex C.

3.14
word
string of 32 bits used in dedicated hash-functions 1, 2, 3, 4, 8 and 17 or a string of 64 bits used in dedicated hash-functions 5, 6, 9 and 10 of ISO/IEC 10118-3:2018

[SOURCE: ISO/IEC 10118-3:2018, 3.2, modified — added specific bit lengths, 32 bits or 64 bits, for different dedicated hash-functions.]

4 Symbols and notation

C_i, C'_i	constant words used in the round-functions
D	input data string, i.e. the data string to be input to the MAC algorithm
$\bar{D}$	padded data string
$j \sim X$	string obtained from a string X at least j bits in length by taking the leftmost j bits of X
H	hash-code
H', H''	strings of L_2 bits which are used in the MAC algorithm computation to store an intermediate result
h	hash-function
h'	hash-function h with modified constants and modified IV

$\bar{h}$	simplified hash-function h without the padding and length appending, and without truncating the round-function output (L_2 bits) to its leftmost L_H bits NOTE 1 $\bar{h}$ is applied to input strings with a length that is a positive integer multiple of L_1. NOTE 2 The output of $\bar{h}$ is L_2 bits rather than L_H bits; in particular, in dedicated hash-functions 6 and 8 defined in ISO/IEC 10118-3:2018, L_H is always smaller than L_2.
IV	initializing value
IV', IV_1, IV_2	initializing values
k	length (in bits) of the MAC algorithm key
K	MAC algorithm key
K', K_0, K_1, K_2, $\bar{K}$, $\bar{K}_1$, $\bar{K}_2$	secret keys derived to be used for a MAC algorithm
$K_1[i]$	i^{th} word in the derived key K_1
KT	first input string of the function ϕ' used in the output transformation step of MAC Algorithm 1
$\tilde{L}$	bit string encoding the message length in MAC Algorithm 3
L_X	length (in bits) of a bit-string X
L_1	length (in bits) of the first of the two input strings to the round-function, ϕ
L_2	length (in bits) of the second of the two input strings to the round-function, ϕ, of the output string from the round-function, ϕ, and of IV
m	length (in bits) of the MAC
$OPAD$, $IPAD$	constant strings used in MAC Algorithm 2
q	number of blocks in the input data string D after the padding and splitting process
R, S_0, S_1, S_2	constant strings used in the computation of the constants for MAC Algorithm 1 and MAC Algorithm 3
T_0, T_1, T_2, U_0, U_1, U_2	constant strings used in the key derivation for MAC Algorithm 1 and MAC Algorithm 3
w	length (in bits) of a word; w is 32 when using dedicated hash-functions 1, 2, 3, 4, 8 and 17 of ISO/IEC 10118-3:2018, and w is 64 when using dedicated hash-functions 5, 6, 9 and 10 of ISO/IEC 10118-3:2018
$X \oplus Y$	bitwise exclusive-or of bit-strings X and Y
$X \mid\mid Y$	concatenation of bit-strings X and Y (in that order)
: =	symbol denoting the "set equal to" operation used in the procedural specifications of MAC algorithms, where it indicates that the value of the string on the left side of the symbol shall be made equal to the value of the expression on the right side of the symbol

ϕ round-function, i.e. if X and Y are bit-strings of lengths L_1 and L_2 respectively, then $\phi(X, Y)$ is the string obtained by applying ϕ to X and Y

ϕ' modified round-function with constants different from those used in the original round function

Ψ modulo 2^w addition operation, where w is the number of bits in a word, i.e. if A and B are words, then AΨB is the word obtained by treating A and B as the binary representations of integers and computing their sum modulo 2^w, and the result is constrained to lie between 0 and $2^w - 1$ inclusive

The value of w is 32 in dedicated hash-functions 1, 2, 3, 4, 8 and 17, and 64 in dedicated hash-functions 5, 6, 9 and 10.

[AC1] $\lceil x \rceil$ the smallest integer greater than or equal to the real number x [AC1]

5 Requirements

Users who wish to employ a MAC algorithm from this document shall select:

— a dedicated hash-function from the functions specified in ISO/IEC 10118-3:2018 so that the hash-function and its round-function or its sponge function is implemented or suitable to use; a MAC algorithm amongst those specified in Clauses 6, 7, 8 and 9 which can use the selected hash-function or its round-function or sponge function; and

— the length (in bits) m of the MAC, where m is at least 32.

The use of dedicated hash-functions 7 and 9 to 16 from ISO/IEC 10118-3 with MAC Algorithms 1 and 3 is not specified in this document. The use of dedicated hash-functions 9 and 10 from ISO/IEC 10118-3 with MAC Algorithm 2 is also not specified in this document. MAC Algorithm 4 makes use of the Keccak function, a primitive (known as a sponge function) used in defining dedicated hash-functions 13 to 16 from ISO/IEC 10118-3. The permitted combinations of MAC algorithms and hash-functions are summarized in Table 1.

Table 1 — Permitted combinations of MAC algorithms and dedicated hash-functions

Dedicated hash-function in ISO/IEC 10118-3		MAC Algorithm 1	MAC Algorithm 2	MAC Algorithm 3	MAC Algorithm 4
1	RIPEMD-160	√	√	√	
2	RIPEMD-128	√	√	√	
3	SHA-1	√	√	√	
4	SHA-256	√	√	√	
5	SHA-512	√	√	√	
6	SHA-384	√	√	√	
7	Whirlpool		√		
8	SHA-224	√	√	√	
9	SHA-512/224				
10	SHA-512/256				
11	STREEBOG 512		√		
12	STREEBOG 256		√		
13	SHA3–224		√		√
14	SHA3–256		√		√
15	SHA3–384		√		√
16	SHA3–512		√		√
17	SM3	√	√	√	

Agreement on these choices amongst the users is essential for use of the data integrity mechanism.

The key K used in a MAC algorithm shall have entropy that meets or exceeds the security strength to be provided by the MAC algorithm.

In every case, the MAC algorithm key K shall be chosen such that every possible key is approximately equally likely to be selected.

For MAC Algorithms 1 and 2, the length m of the MAC is a positive integer less than or equal to the length of the hash-code L_H. For MAC Algorithm 2, the length m of MAC value shall be at least 32 bits. For MAC Algorithm 3, the length m of the MAC is a positive integer less than or equal to half the length of the hash-code, i.e. $m \leq L_H/2$. The length in bits of the input data string may be limited by the dedicated hash-function and/or the MAC algorithm and is discussed for each MAC algorithm. For MAC Algorithm 4, the length in bits of the input data string D shall be at most $2^{2\,040} - 1$. The selection of a specific MAC Algorithm, dedicated hash-function as specified in Table 1, and value for m is beyond the scope of this document.

These choices affect the security level of the MAC algorithm. For a detailed discussion, see Annex C. The key used for calculating and verifying the MAC is the same. If the input data string is also being enciphered, the key used for the calculation of the MAC should be different from that used for encipherment, because it is considered as good cryptographic practice to have independent keys for confidentiality and for data integrity.

Annex A lists the object identifiers which shall be used to identify the mechanisms defined in this document.

Annex B provides numerical examples for the MAC algorithms specified in this document to be used for checking the correctness of implementations.

Annex C describes major attacks and proofs of security for the MAC algorithms specified in this document.

6 MAC Algorithm 1

6.1 General

This clause contains a description of MDx-MAC[10] with dedicated hash-functions 1 to 6, 8 and 17. Table 2 shows the commonly used names of MDx-MAC with individual dedicated hash-functions.

Table 2 — The MDx-MAC algorithm with different dedicated hash-functions

Dedicated hash-function:	The MDx-MAC algorithm is also known as
Dedicated hash-function 1	RIPEMD-160-MAC
Dedicated hash-function 2	RIPEMD-128-MAC
Dedicated hash-function 3	SHA-1-MAC
Dedicated hash-function 4	SHA-256-MAC
Dedicated hash-function 5	SHA-512-MAC
Dedicated hash-function 6	SHA-384-MAC
Dedicated hash-function 8	SHA-224-MAC
Dedicated hash-function 17	SM3-MAC

The use of MAC Algorithm 1 with dedicated hash-functions 7 and 9 to 16 of ISO/IEC 10118-3 is not specified in this document.

MAC Algorithm 1 requires one application of the hash-function to compute a MAC value but requires that the constants in the corresponding round-function be modified. The hash-function shall be selected from dedicated hash-functions 1 to 6, 8 and 17 from ISO/IEC 10118-3. MAC Algorithm 1 can accommodate the maximum of 128 bit key K and therefore provide at most 128 bits security strength. For MAC Algorithm 1, the length in bits of the input data string D shall be at most $2^{64} - 1$ when using

dedicated hash-functions 1, 2, 3, 4, 8 and 17, and at most $2^{128} - 1$ when using dedicated hash-functions 5 and 6.

6.2 Description of MAC Algorithm 1

6.2.1 General

MAC Algorithm 1 involves the following five steps: key expansion, modification of the constants and the *IV*, hashing operation, output transformation, and truncation.

6.2.2 Step 1 (key expansion)

〚AC1〉 If K is shorter than 128 bits, concatenate K to itself $\lceil 128/k \rceil$ times and select the leftmost 128 bits of the result to form the 128-bit key K': 〈AC1〛

K': = 128 ~ (K || K || ... || K). If the length (in bits) of K is greater than or equal to 128, K': = 128~K.

Compute the derived keys K_0, K_1, and K_2 as follows:

K_0:= $\overline{h}$ (K' || U_0 || K')

K_1:= 128 ~ $\overline{h}$ (K' || U_1 || K'), when using dedicated hash-functions 1, 2 and 3

K_1:= 256 ~ $\overline{h}$ (K' || U_1 || K'), when using dedicated hash-functions 4, 5, 6, 8 and 17

K_2:= 128 ~ $\overline{h}$ (K' || U_2 || K')

Here $\overline{h}$ is a simplified hash-function h selected from dedicated hash-functions listed in Table 2 and U_0, U_1, and U_2 are 768-bit constants that are defined in 6.4.1.

Padding and length appending can be omitted because in this case the length of the input string is either L_1 bits or $2L_1$ bits.

When deriving K_0, truncation is omitted and the length of K_0 is always L_2 bits.

When using dedicated hash-functions 1, 2, 3, 5 and 6, the derived key K_1 is split into four words denoted by $K_1[i]$ ($0 \le i \le 3$), i.e.

$K_1 = K_1[0] \,||\, K_1[1] \,||\, K_1[2] \,||\, K_1[3]$

When using dedicated hash-functions 4, 8 and 17, the derived key K_1 is split into eight words denoted by $K_1[i]$ ($0 \le i \le 7$), i.e:

〚AC1〉 $K_1 = K_1[0] \,\|\, K_1[1] \,\|\, K_1[2] \,\|\, K_1[3] \,\|\, K_1[4] \,\|\, K_1[5] \,\|\, K_1[6] \,\|\, K_1[7]$ 〈AC1〛

To convert a string into words, a byte ordering convention is required. The byte ordering convention for this conversion is that which is defined for the selected dedicated hash-functions in ISO/IEC 10118-3.

6.2.3 Step 2 (modification of the constants and the *IV*)

When using dedicated hash-functions 1, 2, 3, 4, 5, 6, 8 and 17, the additive constants used in the round-function are modified by the modulo 2^w addition of a word of K_1, for example:

C_0: = C_0 Ψ $K_1[0]$

Precisely which word of K_1 is added to each constant depends on the hash-function in use, and is specified in 6.4.

The initializing value, *IV*, of the hash-function is replaced by IV': = K_0.

6.2.4 Step 3 (hashing operation)

The string which is input to the modified hash-function h' is equal to the input data string D, i.e.

H': = h' (D).

6.2.5 Step 4 (output transformation)

The modified round-function ϕ' is applied one additional time, with as first input the string KT (defined below) and as second input the string H' (the result of Step 3), i.e.:

H'': = ϕ' (KT, H').

For dedicated hash-functions 1, 2, 3, 4, 8 and 17:

$KT = K_2 \,||\, (K_2 \oplus T_0) \,||\, (K_2 \oplus T_1) \,||\, (K_2 \oplus T_2)$

For dedicated hash-functions 5 and 6:

$KT = K_2 \,||\, (K_2 \oplus T_0) \,||\, (K_2 \oplus T_1) \,||\, (K_2 \oplus T_2) || \, K_2 \,||\, (K_2 \oplus T_0) \,||\, (K_2 \oplus T_1) \,||\, (K_2 \oplus T_2)$

Here T_0, T_1, and T_2 are 128-bit strings defined in 6.4 for each dedicated hash-function.

The output transformation corresponds to processing an additional data block derived from K_2 after padding and appending of the length field.

6.2.6 Step 5 (truncation)

The MAC of m bits is derived by taking the leftmost m bits of the string H'', i.e.

MAC: = $m \sim H''$

6.3 Efficiency

If the padded data string (where the padding algorithm depends on the selected hash-function) contains q blocks, then MAC Algorithm 1 requires q + 7 applications of the round-function when dedicated hash-functions 1, 2, 3, 4, 8 and 17 are selected, and q + 4 applications of the round function when dedicated hash-functions 5 and 6 are selected. This can be reduced to q + 1 applications of the round-function by pre-computing the values K_0, K_1 and K_2, and by replacing the initializing value IV by IV' in the application of the hash-function. It is recommended to make this modification to the code of the hash-function together with the mandatory modification required for Step 2. For long input strings, MAC Algorithm 1 has a performance which is comparable to that of the hash-function used.

6.4 Computation of the constants

6.4.1 General

The constants described in 6.4 are used in MAC Algorithms 1 and 3. MAC Algorithm 3 is specified in Clause 8. The strings T_i and U_i are fixed elements in the description of the MAC algorithm. They are computed (only once) using the hash-function; they are different for each of the eight hash-functions. The 128-bit constants T_i and 768-bit constants U_i are defined as follows. The definition of T_i involves the 496-bit constant R = "ab...yzAB...YZ01...89" and 16-bit constants S_0, S_1, S_2, where S_i is the 16-bit string formed by repeating twice the 8-bit representation of i (e.g. the hexadecimal representation of S_1 is 3131). In both cases, ASCII coding is used; this is equivalent to coding using ISO/IEC 646:

for i: = 0 to 2

T_i:= 128 $\sim \overline{h}$ (S_i || R) for dedicated hash-functions 1, 2, 3, 4, 8 and 17

T_i:= 128 $\sim \overline{h}$ (S_i || R || 0^{512}) for dedicated hash-functions 5 and 6, where 0^{512} is 512 zero bits

for i: = 0 to 2 U_i: = T_i || T_{i+1} || T_{i+2} || T_i || T_{i+1} || T_{i+2}

where the subscripts in T_i are taken modulo 3. In dedicated hash-functions 1, 2, 3, 4, 5, 6, 8 and 17, for all constants C_i, C'_i and all words $K_1[i]$ the most significant bit corresponds to the leftmost bit. The constants C_i and C'_i are presented using a hexadecimal representation.

6.4.2 Dedicated hash-function 1 (RIPEMD-160)

The 128-bit constant strings T_i for dedicated hash-function 1 are defined as follows (in hexadecimal representation):

T_0 = 1CC7086A046AFA22353AE88F3D3DACEB

T_1 = E3FA02710E491D851151CC34E4718D41

T_2 = 93987557C07B8102BA592949EB638F37

Two sequences of constant words C_0, C_1, ..., C_{79} and C'_0, C'_1, ..., C'_{79} are used in the round-function of dedicated hash-functions 1. They are defined as follows:

$C_i = K_1[0]$, $(0 \le i \le 15)$

$C_i = K_1[1] \Psi$ 5A827999, $(16 \le i \le 31)$

$C_i = K_1[2] \Psi$ 6ED9EBA1, $(32 \le i \le 47)$

$C_i = K_1[3] \Psi$ 8F1BBCDC, $(48 \le i \le 63)$

$C_i = K_1[0] \Psi$ A953FD4E, $(64 \le i \le 79)$

$C'_i = K_1[1] \Psi$ 50A28BE6, $(0 \le i \le 15)$

$C'_i = K_1[2] \Psi$ 5C4DD124, $(16 \le i \le 31)$

$C'_i = K_1[3] \Psi$ 6D703EF3, $(32 \le i \le 47)$

$C'_i = K_1[0] \Psi$ 7A6D76E9, $(48 \le i \le 63)$

〚AC1〛 $C'_i = K_1[1]$ $(64 \le i \le 79)$ 〚AC1〛

6.4.3 Dedicated hash-function 2 (RIPEMD-128)

The 128-bit constant strings T_i for dedicated hash-function 2 are defined as follows (in hexadecimal representation).

T_0 = FD7EC18964C36D53FC18C31B72112AAC

T_1 = 2538B78EC0E273949EE4C4457A77525C

T_2 = F5C93ED85BD65F609A7EB182A85BA181

Two sequences of constant words C_0, C_1, ..., C_{63} and C'_0, C'_1, ..., C'_{63} are used in the round-function of dedicated hash-function 2. They are defined as follows:

$C_i = K_1[0]$, $(0 \le i \le 15)$

$C_i = K_1[1] \Psi$ 5A827999, $(16 \le i \le 31)$

$C_i = K_1[2] \Psi$ 6ED9EBA1, $(32 \le i \le 47)$

$C_i = K_1[3] \Psi$ 8F1BBCDC, $(48 \le i \le 63)$

$C'_i = K_1[0] \Psi$ 50A28BE6, $(0 \le i \le 15)$

$C'_i = K_1[1] \Psi$ 5C4DD124, $(16 \leq i \leq 31)$

$C'_i = K_1[2] \Psi$ 6D703EF3, $(32 \leq i \leq 47)$

[AC1] $C'_i = K_1[3]$ $(48 \leq i \leq 63)$ [AC1]

6.4.4 Dedicated hash-function 3 (SHA-1)

The 128-bit constant strings T_i for dedicated hash-function 3 are defined as follows (in hexadecimal representation):

T_0 = 1D4CA39FA40417E2AE5A77B49067BBCC

T_1 = 9318AFEF5D5A5B46EFCA6BEC0E138940

T_2 = 4544209656E14F97005DAC76868E97A3

A sequence of constant words C_0, C_1, ..., C_{79} is used in the round-function of dedicated hash-function 3. It is defined as follows.

$C_i = K_1[0] \Psi$ 5A827999, $(0 \leq i \leq 19)$

$C_i = K_1[1] \Psi$ 6ED9EBA1, $(20 \leq i \leq 39)$

$C_i = K_1[2] \Psi$ 8F1BBCDC, $(40 \leq i \leq 59)$

$C_i = K_1[3] \Psi$ CA62C1D6, $(60 \leq i \leq 79)$

6.4.5 Dedicated hash-function 4 (SHA-256)

[AC1] The 128-bit constant strings T_i for dedicated hash-function 4 are defined as follows (in hexadecimal representation): [AC1]

T_0 = 13C10FB018D2C57E189060502F7DB523

T_1 = 3DD6B5AE05B11977F3BFDC25CB1F35A8

T_2 = E31F81250B926FEAD2A82A6F63DD66D5

A sequence of constant words C_0, C_1, ..., C_{63} is used in the round-function of dedicated hash-function 4. It is defined as follows.

$C_i = K_1[i \bmod 8] \Psi C''_i$ $(0 \leq i \leq 63)$

where the sequence C''_0, C''_1, ..., C''_{63} in a hexadecimal representation (where the most significant bit corresponds to the leftmost bit) is defined as follows, where the words are listed in the order C''_0, C''_1, ..., C''_{63}.

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

NOTE These values are the first 32 bits of the fractional parts of the cube roots of the first 64 primes. They are the constant sequence used in SHA-256.

6.4.6 Dedicated hash-function 5 (SHA-512)

[AC1] The 128-bit constant strings T_i for dedicated hash-function 5 are defined as follows (in hexadecimal representation): [AC1]

T_0 = 85f6e8b28ba014ed11d076ead90412a5

T_1 = 33a6da6c7aaaf2149104fe4183152828

T_2 = 7682094a7e45cf6bf27d19c2c7d6cf77

A sequence of constant words C_0, C_1, ..., C_{79} is used in the round-function of dedicated hash-function 5. It is defined as follows:

$C_i = K_1[i \bmod 4] \Psi C''_i \ (0 \le i \le 79)$,

where the sequence C''_0, C''_1, ..., C''_{79} in a hexadecimal representation (where the most significant bit corresponds to the leftmost bit) is defined as follows, where the words are listed in the order C''_0, C''_1, ..., C''_{79}.

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817
```

These values are the first 64 bits of the fractional parts of the cube roots of the first 80 primes. They are the constant sequence used in SHA-512.

6.4.7 Dedicated hash-function 6 (SHA-384)

⟦AC1⟩ The 128-bit constant strings T_i for dedicated hash-function 6 are defined as follows (in hexadecimal representation): ⟨AC1⟧

T_0 = 33bfc7a7db2d833c1fa120f248ea0c68

T_1 = 0f53e26170ddedf90aa666a58accf8c4

T_2 = f9371fddd155caefbd989e1270066c7c

A sequence of constant words C_0, C_1, ..., C_{79} is used in the round-function of dedicated hash-function 6. It is defined as follows:

$C_i = K_1[i \bmod 4] \Psi C''_i \ (0 \le i \le 79)$

where the sequence C''_0, C''_1, ..., C''_{79} is the same as that for dedicated hash-function 5 of 6.4.6.

6.4.8 Dedicated hash-function 8 (SHA-224)

⟦AC1⟩ The 128-bit constant strings T_i for dedicated hash-function 8 are defined as follows (in hexadecimal representation): ⟨AC1⟧

T_0 = F48D1673DF60BE1B00BACDC816CC3E4A

T_1 = A38F38CA4247A2F794F62F3F76460AB7

T_2 = 7AA9B4EF4ADB2BCF85F123B1FDEFAC1A

A sequence of constant words C_0, C_1, ..., C_{63} is used in the round-function of dedicated hash-function 8. It is defined as follows:

$C_i = K_1[i \bmod 8] \Psi C''_i \ (0 \le i \le 63)$

where the sequence C''_0, C''_1, ..., C''_{63} is the same as that for dedicated hash-function 4 of 6.4.5.

6.4.9 Dedicated hash-function 17 (SM3)

The 128-bit constant strings T_i for dedicated hash-function 17 are defined as follows (in hexadecimal representation).

T_0 = 52EA0B36B5A4FA8C8D9403894A7421BF

T_1 = 457E3B1FCE828A8E1442AA01AC83E2BE

T_2 = 740B7A08B7CCB27F54B31B160EF57302

A sequence of constant words C_0, C_1, ..., C_{63} is used in the round-function of dedicated hash-function 17. It is defined as follows:

$C_i = K_1[i \bmod 8] \Psi C'_i \ (0 \le i \le 63)$,

where the sequence C'_0, ..., C'_{63} in a hexadecimal representation is defined as follows:

C'_i = 79CC4519 $(0 \le i \le 15)$,

C'_i = 7A879D8A $(16 \le i \le 63)$.

7 MAC Algorithm 2

7.1 General

This clause contains a description of HMAC[8]. MAC Algorithm 2 requires two applications of a hash-function to compute a MAC value. The hash-function shall be selected from ISO/IEC 10118-3, with the requirement that L_1 is a positive integer multiple of 8.

The key size k in bits shall be at least L_2, where L_2 is the size of the hash-code in bits. If an input key has size larger than L_1 bits, where L_1 is the size of the data input of the round-function in bits, then the hash-code of the key is used as the key which is L_2 bits. Therefore, for MAC Algorithm 2, it is assumed that the key size is at most L_1 bits, i.e. $L_2 \le k \le L_1$.

For MAC Algorithm 2, the length in bits of the input data string D shall be at most 2^{64} - 512 when using dedicated hash-functions 1, 2, 3, 4, 8 and 17, 2^{128} – 1 024 when using dedicated hash-functions 5 and 6, 2^{256} – 512 when using dedicated hash-function 7, and 2^{512} – 512 when using dedicated hash-functions 11 and 12.

Dedicated hash-functions 9 and 10 are truncated versions of dedicated hash-function 5 with different initializing values. They are not specified to be used for MAC Algorithm 2.

7.2 Description of MAC Algorithm 2

7.2.1 General

MAC Algorithm 2 requires the following four steps: key expansion, hashing operation, output transformation, and truncation.

7.2.2 Step 1 (key expansion)

Append (L_1 - k) zero bits to the right of the key K; the resulting string of length L_1 is denoted by $\bar{K}$. The key $\bar{K}$ is expanded to two subkeys $\bar{K}_1$ and $\bar{K}_2$.

— Construct the string *IPAD* by concatenating the hexadecimal value '36' (or the binary value '00110110') to itself $L_1/8$ times. Then compute the value $\bar{K}_1$ as the exclusive-or of $\bar{K}$ and the string *IPAD*, i.e.:

$$\bar{K}_1 := \bar{K} \oplus IPAD$$

— Construct the string *OPAD* by concatenating the hexadecimal value '5c' (or the binary value '01011100') to itself $L_1/8$ times. Then compute the value $\bar{K}_2$ as the exclusive-or of $\bar{K}$ and the string *OPAD*, i.e.:

$$\bar{K}_2 := \bar{K} \oplus OPAD$$

7.2.3 Step 2 (hashing operation)

The string which is input to the hash-function is equal to the concatenation of $\bar{K}_1$ and D, i.e.:

$$H' := h(\bar{K}_1 \,||\, D)$$

7.2.4 Step 3 (output transformation)

The string which is input to the hash-function is equal to the concatenation of $\bar{K}_2$ and H', i.e.:

$$H'' := h(\bar{K}_2 \,||\, H')$$

7.2.5 Step 4 (truncation)

The MAC of m bits is derived by taking the leftmost m bits of the string H'', i.e.:

$$\text{MAC} := m \sim H''$$

7.3 Efficiency

If the padded data string (where the padding algorithm is defined for a specific hash-function) contains q blocks, then MAC Algorithm 2 requires $q + 3$ applications of the round-function.

This can be reduced to $q + 1$ applications of the round-function by modifying the code for the hash-function. It is possible to pre-compute the values $IV_1 := \phi(\bar{K}_1, IV)$ and $IV_2 := \phi(\bar{K}_2, IV)$ and replace the initial value IV by IV_1 in the first application of the hash-function, and by IV_2 in the output transformation (the second application of the hash-function). This also requires a modification of the padding method: indeed, the actual input to the hash-function is now L_1 bits shorter. This means that the value L_1 shall be added to the value L_D.

8 MAC Algorithm 3

8.1 General

This clause contains a variant of MAC Algorithm 1 that is optimized for short inputs (at most 256 bits). To compute a MAC value, MAC Algorithm 3 requires seven applications of the simplified round-function when dedicated hash-functions 1, 2, 3, 4, 8 and 17 are selected, and four applications of the simplified round function when dedicated hash-functions 5 and 6 are selected. But this can be reduced to a single application of the simplified round-function by some pre-computation. The hash-function shall be selected from dedicated hash-functions 1 to 6, 8 and 17 from ISO/IEC 10118-3:2018.

For MAC Algorithm 3, the length in bits of the input data string D shall be at most 256 when using Dedicated hash-functions 1 to 6, 8 and 17 from ISO/IEC 10118-3:2018.

The key size k in bits shall be at most 128 bits, and the MAC length m in bits shall be at most $L_H/2$.

MAC Algorithm 3 can accommodate the maximum of 128-bit key K and therefore provide at most 128-bits security strength.

8.2 Description of MAC Algorithm 3

8.2.1 General

MAC Algorithm 3 requires the following five steps: key expansion, modification of the constants of the round-function, padding, application of the round-function, and truncation.

8.2.2 Step 1 (key expansion)

〚AC1〛 If K is shorter than 128 bits, concatenate K to itself $\lceil 128/k \rceil$ times and select the leftmost 128 bits of the result to form the 128-bit key K': 〚AC1〛

K': = 128 ~ (K || K || ... || K). If the length (in bits) of K is greater than or equal to 128, K': = 128 ~ K.

Compute the subkeys K_0, K_1, and K_2 as follows:

K_0:= $\bar{h}$ (K' || U_0 || K')

K_1:= 128 ~ $\bar{h}$ (K' || U_1 || K'), when using dedicated hash-functions 1, 2 and 3

K_1:= 256 ~ $\bar{h}$ (K' || U_1 || K'), when using dedicated hash-functions 4, 5, 6, 8 and 17

K_2:= 128 ~ $\bar{h}$ (K' || U_2 || K')

Here, U_0, U_1 and U_2 are 768-bit constants that are defined in 6.4.1.

Padding and length appending are omitted because in this case the length of the input string is either L_1 bits or $2L_1$ bits.

When deriving K_0, truncation is omitted and the length of K_0 is always L_2 bits.

When using dedicated hash-functions 1, 2, 3, 5 and 6 the derived key K_1 is split into four words denoted $K_1[i]$ ($0 \le i \le 3$), i.e.:

$K_1 = K_1[0] \,||\, K_1[1] \,||\, K_1[2] \,||\, K_1[3]$

When using dedicated hash-functions 4, 8 and 17, the derived key K_1 is split into eight words denoted $K_1[i]$ ($0 \le i \le 7$), i.e.:

$K_1 = K_1[0] \,||\, K_1[1] \,||\, K_1[2] \,||\, K_1[3]|| \, K_1[4] \,||\, K_1[5] \,||\, K_1[6] \,||\, K_1[7]$

For conversion of a string into words, a byte ordering convention is required. The byte ordering convention for this conversion is that which is defined for each dedicated hash-function in ISO/IEC 10118-3.

8.2.3 Step 2 (modification of the constants and the *IV*)

When using dedicated ash-Fufnctions 1, 2, 3, 4, 5, 6, 8 and 17, the additive constants used in the round-function are modified by the modulo 2^w addition of a word of K_1, e.g.:

C_0: = C_0 Ψ $K_1[0]$

In 6.4, it is indicated which word of K_1 is added to each constant.

The initial value IV': = K_0, when round-function ϕ' is computed.

8.2.4 Step 3 (padding)

The padding bits that are added to the original data string are only used for calculating the MAC. Consequently, these padding bits (if any) are not stored or transmitted with the data. The verifier shall know whether or not the padding bits have been stored or transmitted. The data string D to be input to the MAC algorithm shall be right-padded with as few (possibly none) "0" bits as necessary to obtain a data string $\bar{D}$ of length 256 bits.

If the input data string is empty, the padded data string $\bar{D}$ consists of 256 "0" bits.

8.2.5 Step 4 (application of the round-function)

The bit string $\tilde{L}$ is computed as the binary representation of the length (in bits) L_D of the data string D, left-padded with as few '0' bits as necessary to obtain a 128-bit string. The rightmost bit of the bit string $\tilde{L}$ corresponds to the least significant bit of the binary representation of L_D. The string which is input to the round-function ϕ' (with modified constants) is equal to the concatenation of K_2, $\bar{D}$, and the exclusive-or of K_2 and $\tilde{L}$.

For dedicated hash-functions 1, 2, 3, 4, 8 and 17:

$$H' := \phi'(K_2 \,||\, \bar{D} \,||\, (K_2 \oplus \tilde{L}), IV')$$

For dedicated hash-functions 5 and 6:

$$H' = \phi'(K_2 \,||\, \bar{D} \,||\, (K_2 \oplus \tilde{L}) \,||\, K_2 \,||\, \bar{D} \,||\, (K_2 \oplus \tilde{L}), IV')$$

8.2.6 Step 5 (truncation)

The MAC of m bits is derived by taking the leftmost m bits of the string H', i.e.

MAC: = $m \sim H'$

8.3 Efficiency

MAC algorithm 3 requires seven applications of the simplified round-function, when dedicated hash-functions 1, 2, 3, 4, 8 and 17 are selected, four applications of the simplified round function when dedicated hash-functions 5 and 6 are selected. This can be reduced to a single application of the round-function by pre-computing the values K_0, K_1 and K_2.

9 MAC Algorithm 4

9.1 General

This clause contains a description of the Keccak message authentication code (KMAC) algorithm.

KMAC has two variants: KMAC128 and KMAC256. The two variants differ somewhat in their security properties. Both variants can support any security strength up to 256 bits of security, provided that a long enough key is used.

Parameter m is the requested output length of the MAC algorithm in bits. Some applications of KMAC can not know the number of output bits they will need until after the outputs begin to be produced. MAC Algorithm 4 is specified with two situations.

a) KMAC128 and KMAC256 include m as input to form the MAC input data *newD* to generate output.

b) KMACXOF128 and KMACXOF256 do not include m as input to form the MAC input data *newD* to generate output.

For the input to MAC Algorithm 4, the order of bits in a byte is least significant bit first.

9.2 Description of MAC Algorithm 4

The input to MAC Algorithm 4 involves concatenating a padded version of the key K with the input D and an encoding of the requested output length m, in case of KMAC128 and KMAC256. The result is a data string labelled *newD*.

The data *newD* is prepended with an encoding of a constant N = "KMAC" = 11010010 10110010 10000010 11000010 and an additional input, S, an optional customization string, a string specified by the application. If no customization is desired, S is set to the empty string.

The variants of MAC Algorithm 4 are defined using sponge function SPONGE[f, *pad*, r] used in dedicated hash-functions 13, 14, 15, 16, SHAKE128 and SHAKE256 of ISO/IEC 10118-3, where the function f is the permutation KECCAK-p and r is the rate corresponding to the number of bits in each message block. In particular,

$$\text{KMAC}v\ (K, D, m, S) = \text{SPONGE}[f, pad, 1\,600 - 2v]\ (X, m)$$

where v = 128 or 256 and X is padded input of key K and D with additional information.

The length in bits of X should be less than $2^{2\,040}$.

9.3 Encoding and padding

9.3.1 Integer to byte encoding

Two internal functions, *left_encode* and *right_encode*, are defined to encode integers as byte strings. Both functions can encode integers up to an extremely large maximum, $2^{2\,040}$-1.

left_encode(x) encodes the integer x as a byte string in a way that can be unambiguously parsed from the beginning of the string by inserting the length of the byte string before the byte string representation of x.

right_encode(x) encodes the integer x as a byte string in a way that can be unambiguously parsed from the end of the string by inserting the length of the byte string after the byte string representation of x.

For an integer i ranging from 0 to 255, $enc_8(i)$ is the byte encoding of i, with bit 0 being the least significant bit of the byte. Using the function $enc_8()$ to encode the individual bytes, these two functions are defined as follows:

***right_encode*(*x*)**

For $0 \le x < 2^{2\,040}$, perform the following steps.

a) let n be the smallest positive integer for which $2^{8n} > x$;

b) let $x_1, x_2, \ldots, x_n$ be the base-256 encoding of x satisfying:

$x = \sum 2^{8(n-i)}\, x_i$, for i = 1 to n;

c) let $O_i = enc_8(x_i)$, for i = 1 to n;

d) let $O_{n+1} = enc_8(n)$;

e) return $O = O_1 \,||\, O_2 \,||\, \ldots \,||\, O_n \,||\, O_{n+1}$.

***left_encode*(*x*)**

For $0 \leq x < 2^{2\,040}$, perform the following steps.

a) let n be the smallest positive integer for which $2^{8n} > x$;

b) let $x_1, x_2, \ldots, x_n$ be the base-256 encoding of x satisfying:

$x = \sum 2^{8(n-i)} x_i$, for $i = 1$ to n;

c) let $O_i = enc_8(x_i)$, for $i = 1$ to n;

d) let $O_0 = enc_8(n)$;

e) return $O = O_0 \,||\, O_1 \,||\, \ldots \,||\, O_{n-1} \,||\, O_n$.

As an example, *right_encode*(0) yields 00000000 10000000, and *left_encode*(0) yields 10000000 00000000.

9.3.2 String encoding

The *encode_string* function is used to encode bit strings in a way that can be parsed unambiguously from the beginning of the string, S. The function is defined as follows:

***encode_string*(*S*)**

For $0 \leq L_S < 2^{2\,040}$, perform the following step:

Return *left_encode*(L_S) || S.

As an example, when S is the empty string "", $L_S = 0$. The *left_encode*(0) is 10000000 00000000. In this case, *encode_string*(S) yields 10000000 00000000 || "".

Note that if the bit string S is not byte-oriented (i.e. L_S is not a multiple of 8), the bit string returned from *encode_string*(S) is also not byte-oriented. However, if L_S is a multiple of 8, then the length of the output of *encode_string*(S) is also a multiple of 8.

9.3.3 Padding

The *bytepad*(X, w) function prepends an encoding of the integer w to an input string X, then pads the result with zeros until it is a byte string whose length in bytes is a multiple of w. In general, *bytepad* is intended to be used on encoded strings — the byte string *bytepad*(*encode_string*(S), w) can be parsed unambiguously from its beginning, whereas *bytepad* does not provide unambiguous padding for all input strings.

The definition of *bytepad*() is as follows.

***bytepad*(*X*, *w*)**

For $w > 0$, perform the following steps:

a) z = *left_encode*(w) || X;

b) while $L_z \bmod 8 \neq 0$:

$z = z \,||\, 0$;

c) while $(L_z/8) \bmod w \neq 0$:

$z = z \,||\, 00000000$;

d) return z.

9.4 KMAC128

9.4.1 General

The input to KMAC128 includes a MAC key *K*, input data *D*, an output length *m*, and an optional customization string *S*.

Verify that $L_K < 2^{2\,040}$ and $0 \leq m < 2^{2\,040}$ and $L_S < 2^{2\,040}$, before performing the steps in 9.4.2 to 9.4.4.

9.4.2 Step 1 (Prepare *newD*)

For MAC key *K*, input data *D*, and output length *m*, set:

a) *K'* = *encode_string*(*K*);

b) *m'* = *right_encode*(*m*);

c) *newD* = *bytepad*(*K'*, 168) || *D* || *m'*.

9.4.3 Step 2 (Prepare *X*)

For *newD*, *N* = "KMAC" = 11010010 10110010 10000010, and customization string *S*, set:

a) *N'* = *encode_string*(*N*);

b) *S'* = *encode_string*(*S*);

c) *X* = *bytepad* (*N'* || *S'*, 168) || *newD* || 00.

AC1 The characters 00 in item c) specify two zero bits.

NOTE The number 168 is the rate (in bytes) of SPONGE[*f*, *pad*, 1 344], where *f* = KECCAK-*p*[1 600, 24]. SPONGE[*f*, *pad*, 1 344] is referred to as KECCAK[256] in Reference [12]. AC1

9.4.4 Step 3 (Generate MAC output)

For *X* and output length *m*, compute

a) KMAC128 (*K*, *D*, *m*, *S*) = SPONGE[*f*, *pad*, 1 344] (*X*, *m*), where *f* = KECCAK-*p* [1 600, 24].

For dedicated hash-functions 13 to 16, the input data to the hash-function appended with suffix "01" as input to KECCAK-*p*, while for MAC Algorithm 4, the input data to KECCAK-*p* includes a two-bit suffix "00".

9.5 KMAC256

9.5.1 General

The input to KMAC256 includes a MAC key *K*, input data *D*, an output length *m*, and an optional customization string *S*.

Verify that $L_K < 2^{2\,040}$ and $0 \leq m < 2^{2\,040}$ and $L_S < 2^{2\,040}$, before performing steps in 9.5.2 to 9.5.4.

9.5.2 Step 1 (Prepare *newD*)

For MAC key *K*, input data *D*, and output length *m*, set:

a) *K'* = *encode_string*(*K*);

b) *m'* = *right_encode* (*m*);

c) *newD* = *bytepad*(*K'*, 136) || *D* || *m'*.

9.5.3 Step 2 (Prepare *X*)

For *newD*, *N* = "KMAC" = 11010010 10110010 10000010, and customization string *S*, set:

a) *N'* = *encode_string*(*N*);

b) *S'* = *encode_string*(*S*);

c) *X* = *bytepad* (*N'* || *S'*, 136) || *newD* || 00.

AC1 The characters 00 in item c) specify two zero bits.

NOTE The number 136 is the rate (in bytes) of SPONGE[*f*, *pad*, 1 088], where *f* = KECCAK-*p*[1 600, 24]. SPONGE[*f*, *pad*, 1 088] is referred to as KECCAK[512] in Reference [12]. AC1

9.5.4 Step 3 (Generate MAC output)

For *X* and output length *m*, compute:

a) KMAC256 (*K*, *D*, *m*, *S*) = SPONGE[*f*, *pad*, 1 088] (*X*, *m*), where *f* = KECCAK-*p* [1 600, 24].

For dedicated hash-functions 13 to 16, the input data to the hash-function appended with suffix "01" as input to KECCAK-*p*, while for MAC Algorithm 4, the input data to KECCAK-*p* includes a two-bit suffix "00".

9.6 KMACXOF128

9.6.1 General

The input to KMACXOF128 includes a MAC key *K*, input data *D*, an output length *m*, and an optional customization string *S*.

Verify that $L_K < 2^{2\,040}$ and $0 \le m < 2^{2\,040}$ and $L_S < 2^{2\,040}$, before performing the steps in 9.6.2 to 9.6.4.

9.6.2 Step 1 (Prepare *newD*)

For MAC key *K* and input data *D*, set:

a) *K'* = *encode_string*(*K*);

b) *o'* = *right_encode* (0);

c) *newD* = *bytepad*(*K'*, 168) || *D* || *o'*.

NOTE AC1 When used as a XOF, KMAC is computed by setting the encoded output length to 0, as in shown *right_encode*(0) in item b). AC1

9.6.3 Step 2 (Prepare *X*)

For *newD*, *N* = "KMAC" = 11010010 10110010 10000010, and customization string *S*, set:

a) *N'* = *encode_string*(*N*);

b) *S'* = *encode_string*(*S*);

c) *X* = *bytepad* (*N'* || *S'*, 168) || *newD* || 00.

AC1 The characters 00 in item c) specify two zero bits.

NOTE The number 168 is the rate (in bytes) of SPONGE[*f*, *pad*, 1 344], where *f* = KECCAK-*p*[1 600, 24]. SPONGE[*f*, *pad*, 1 344] is referred to as KECCAK[256] in Reference [12]. AC1

9.6.4 Step 3 (Generate MAC output)

For X and output length m, compute:

a) KMACXOF128 (K, D, m, S) = SPONGE[f, pad, 1 344] (X, m), where f = KECCAK-p [1 600, 24].

For dedicated hash-functions 13 to 16, the input data to the hash-function appended with suffix "01" as input to KECCAK-p, while for MAC Algorithm 4, the input data to KECCAK-p includes a two-bit suffix "00".

9.7 KMACXOF256

9.7.1 General

The input to KMACXOF256 includes a MAC key K, input data D, an output length m, and an optional customization string S.

Verify that $L_K < 2^{2\,040}$ and $0 \leq m < 2^{2\,040}$ and $L_S < 2^{2\,040}$, before performing the steps in 9.7.2 to 9.7.4.

9.7.2 Step 1 (Prepare *newD*)

For MAC key K and input data D, set:

a) K' = *encode_string*(K);

b) o' = *right_encode*(0);

c) *newD* = *bytepad*(K', 136) || D || o'.

When used as a XOF, KMAC is computed by setting the encoded output length to 0, as shown in *right_encode*(0) in item b). That is, the output length is not an input for MAC computation.

9.7.3 Step 2 (Prepare *X*)

For *newD*, N = "KMAC" = 11010010 10110010 10000010, and customization string S, set:

a) N' = *encode_string*(N);

b) S' = *encode_string*(S);

c) X = *bytepad* (N' || S', 136) || *newD* || 00.

〚AC1〛 The characters 00 in item c) specify two zero bits.

NOTE The number 136 is the rate (in bytes) of SPONGE[f, pad, 1 088], where f = KECCAK-p[1 600, 24]. SPONGE[f, pad, 1 088] is referred to as KECCAK[512] in Reference [12]. 〚AC1〛

9.7.4 Step 3 (Generate MAC output)

For X and output length m, compute:

a) KMACXOF256 (K, D, m, S) = SPONGE[f, pad, 1 088] (X, m), where f = KECCAK-p [1 600, 24].

For dedicated hash-functions 13 to 16, the input data to the hash-function appended with suffix "01" as input to KECCAK-p, while for MAC Algorithm 4, the input data to KECCAK-p includes a two-bit suffix "00".

Annex A
(normative)

Object identifiers

This annex specifies the object identifiers for the MAC mechanisms specified in this document.

```
MechanismsUsingADedicatedHashFunction {
   iso(1) standard(0) message-authentication-codes(9797) part2(2)
      asn1-module(0) mechanisms-using-a-dedicated-hash-function(0) version2(2)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
   OID, ALGORITHM, HashFunctions, HashFunctionAlgs
    FROM DedicatedHashFunctions {iso(1) standard(0)
hash-functions(10118) part3(3)
asn1-module(1) dedicated-hash-functions(0)};

-- OID assignments
-- ===============

is9797-2 OID ::= {iso standard message-authentication-codes(9797)
part2(2)}

id-mac-1 OID ::= {is9797-2 macAlgorithm-1(1)}
id-mac-2 OID ::= {is9797-2 macAlgorithm-2(2)}
id-mac-3 OID ::= {is9797-2 macAlgorithm-3(3)}

-- MAC algorithm identifier type and the set of recognized MAC algorithms
-- ======================================================================
AlgorithmIdentifier {ALGORITHM:IOSet} ::= SEQUENCE {
algorithm ALGORITHM.&id({IOSet}),
parameters ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}

MessageAuthenticationCode ::= AlgorithmIdentifier{{MacAlgorithms}}

MacAlgorithms ALGORITHM ::= {
   {OID id-mac-1 PARMS MacParameters} |
   {OID id-mac-2 PARMS MacParameters} |
   {OID id-mac-3 PARMS MacParameters} ,
   ... -- additional algorithms expected --
}

-- MAC parameter type definitions
-- ==============================

-- The optional parameters may be agreed upon by other means

MacParameters ::= SEQUENCE {
   dhfAlgo HashFunctions OPTIONAL,
   m INTEGER (1..MAX)
}

hashAlgs OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) country(16) us(840) organization(1)
gov(101) csor(3) nistAlgorithm(4) hashAlgs(2)}

id-KMAC128 OBJECT IDENTIFIER ::=  { hashAlgs 19 }
id-KMAC256 OBJECT IDENTIFIER ::=  { hashAlgs 20 }
id-KMACXOF128 OBJECT IDENTIFIER ::= { hashAlgs 21 }
id-KMACXOF256 OBJECT IDENTIFIER ::= { hashAlgs 22 }
```

```
CustomizationString ::= OCTET STRING

KMAC128Parameters ::= SEQUENCE {
   customizationString  CustomizationString DEFAULT ''H,
   kMACOutputLength     INTEGER DEFAULT 256 -- Output length in bits
}

KMAC256Parameters ::= SEQUENCE {
   customizationString  CustomizationString DEFAULT ''H,
   kMACOutputLength     INTEGER DEFAULT 512 -- Output length in bits
}

alg-KMAC128 ALGORITHM ::= { OID id-KMAC128 PARMS KMAC128Parameters }
alg-KMAC256 ALGORITHM ::= { OID id-KMAC256 PARMS KMAC256Parameters }
alg-KMACXOF128 ALGORITHM ::= { OID id-KMACXOF128 PARMS
CustomizationString DEFAULT ''H  }
alg-KMACXOF256 ALGORITHM ::= { OID id-KMACXOF256 PARMS
CustomizationString  DEFAULT ''H }

END -- MechanismsUsingADedicatedHashFunction --
```

Annex B
(informative)

Numerical examples

B.1 General

This annex gives examples for the computation of MAC Algorithms 1, 2, 3 and 4. The examples of MAC Algorithms 1 and 3 use dedicated hash-functions 1 to 6, 8 and 17. The examples of MAC Algorithm 2 use dedicated hash-functions 1 to 8, 11, 12, 13 to 16, and 17. Dedicated hash-functions 1 to 17 are specified in ISO/IEC 10118-3. Nine examples of hash-code calculations are given for MAC Algorithms 1, 2 and 4. The input strings numbered 1 to 9 are contained in Table B.1. Only the first 5 examples in Table B.1 are given for MAC Algorithm 3.

In this annex, ASCII coding of data strings is used for character encoding; this is equivalent to coding using ISO/IEC 646.

Table B.1 — Input strings for the test values

no.	input string
1	“” (empty string)
2	“a”
3	“abc”
4	“message digest”
5	“abcdefghijklmnopqrstuvwxyz”
6	“abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq”
7	“ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789”
8	“1234567890” repeated 8 times to give 80 characters
9	“a” repeated 1 000 000 times to give 1 million characters

The following two 128-bit strings represent the keys unless otherwise specified.

key 1 = `00112233445566778899AABBCCDDEEFF` and

key 2 = `0123456789ABCDEFFEDCBA9876543210`.

B.2 MAC Algorithm 1

B.2.1 General

For the examples in B.2, the value $m = L_2/2$ has been selected, that is:

— m = 80 for dedicated hash-functions 1 and 3;

— m = 64 for dedicated hash-function 2;

— m = 128 for dedicated hash-functions 4 and 17;

— m = 256 for dedicated hash-function 5;

— m = 192 for dedicated hash-function 6; and

— m = 112 for dedicated hash-function 8.

B.2.2 Dedicated hash-function 1 (RIPEMD-160)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	B7F4508111EB8C3B5229C6AED406DE9ECA640133
2	BC78F55933BCEB1EE85A906F9E18374F23E310F9
3	6300DC20E97A5AA29DB9C7D607D23D126FA36863
4	3A2AC89B78EEAB8759F5112BCAD4CD405EEB5D35
5	16DC174925BBC27E0C93D426C346846F97F8BC69
6	E062210BA5C9C94737BF3A6E85B3B5664FBD1D4E
7	9B462D5CBDAE1485FFE10BC001EF9E3AF6D128B5
8	88E73A01A1DE36C92D6F9E41F7278D407B4A4CCD
9	E7B128E4A1842B750F1E61A486C867C4887A4B21

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	B45D6CA84CFB9020E0D5ABA2A7609D3D81F3F57F
2	8844375992037D1BCD0D118EE548D70C3F19CBBB
3	917C59B8AC7FC19DC25BEF82766412FA16BBC6A7
4	E0737CC7976D8F424390CB8798D623D751AFE15A
5	D57FAE836870718EFA4BD4A5F2F322A179A8735E
6	42B20D4C8FD5E8672760CF83C0478D7BF8021404
7	63DEA9DD7B52CC8C058B2D55B63E1874F8D85C96
8	10441DF4F68CE8815818DC0FB370ABF87BCA4464
9	E06AD21D2AF04DD4217AB03B1A578F036997D01A

B.2.3 Dedicated hash-function 2 (RIPEMD-128)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	64-bit MAC result: leftmost 64 bits of
1	A47A64E9EDE0741B3FDDE33E5C1C6D78
2	51355051852FDC79FB228EAC905633AD
3	D83940DAFFBD4CBBE6BA30A6F9E63F5F
4	1A7CFE2BB26E973E213C1CB96FA4C2EF
5	798AEAC6046B31907C197BD68E59D376
6	0B8E1D4A571F32657189E22A1F2F4A53
7	B814730F482300C6E474FD255A66D680
8	9060A30758EBE3368D939AC168F1A9FD
9	20763FDEDF01E56FF5756954302C7DE0

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	64-bit MAC result: leftmost 64 bits of
1	35FA3AC39F50F2A4E3FFC7AF5776B4EB
2	A89E25E6796747B630A2A00B802EA53E

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	64-bit MAC result: leftmost 64 bits of
3	66339027A36608EBD932DD551616E7B2
4	1F8779BAD84B50373931211A2761EAD3
5	31BF5B5B7ABAC2567DC0E02F1C3A25D7
6	B5B8BA3B8EA895FBC83CB7588FBD2656
7	8D27BBEC257C848D5CF375EB5EDA4CC7
8	B40B5BF6727DE90B26F770850F059C89
9	76C7BC831B0BCE593DFD44E8E054A373

B.2.4 Dedicated hash-function 3 (SHA-1)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	C8A8B3C75E6CE7C6C4F79CC19853CCD54ABCB079
2	8DD9AE643BF10BBB7B978EF13EE6C0F480618FB0
3	A738B26A8BD318184E76707A99CAE14C670B9711
4	1EBFE413E55D6B288A2BD01D294A21FD8D4B20BF
5	0CE7BF40A73D977AB4999CF3A9BD1C5BEDC442E9
6	12A6823CC181294F95109073A6AA0C8961B14386
7	9369EE4A043AF1CA6E078D0B8A9CE5C1545440BA
8	B00D37D70A84B762FC0A8A9BC1B15F0E517B5EDF
9	DDDF44613E8559D12C150D022D5FE33F9E0FBACE

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	C3A5ECD1E715C7272CFE78BC278086587B040422
2	D5D50FFA7EFDF1B17E96E2EC14DBC4412F7B771F
3	01BFDD568008D412158F5B0C90AE2730DCFB77FB
4	9982E0EE91DB89AE7E7618AD1D649BA43406DBDD
5	ACD04E1004FCE53DECA9EE7AB95DAF97B7C44AA8
6	FADF62DCE789E86E60756AA819EF62C3E5C25E94
7	46DB9A49FB4976D007B14B1574843D019CA99445
8	4EF5BED3E816C530B23F491583C038596BB76FDB
9	BAC6BE6BE6153FECE2891F9DA03824DD4D535D19

B.2.5 Dedicated hash-function 4 (SHA-256)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	76D91CA2337FF25EF66DF2AE7172626C5544428822B9E1B9C94121D384489C09
2	479DA7965233CB2133C6B949AD82CBC5B29F528DB90FF04A1496323D77D15FFD
3	BE6E923798F594BC529C87DF5A42333EE18BE88FED984B0EFE092BF31D570FAE
4	664AA91CFF68C786E943FEB0E6BB465213FFC57AF5C8F973827DF67956FC21D6
5	D7A7A1D1007CB2D3DC578BB4FDA4A5B1B2EBF7B27C9CC43BDF7A382851DF91AB
6	BC9853B4E7AE574B3DCF4728BF44FC27A3C04C5AB90EA189DD175B6F5ECB4335

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
7	66F3238A2C2B6A3AB86089B9DF33BA6420F7E66F5DE6856D79CCA908DFE57BFE
8	B1A59E0905F8EE9ACF5C77E67883C8C3CA10DA965BE31F75A47AD85015CD478B
9	15FC09FABB62AADEE831B9988E2DE2F41A3C685D28E4C06720ED6E8493CD060A

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	F0400A79DD136B4E83EB507E23F98C1A54E7DCA33A38C75902008F90B003C37A
2	407FA9A8170113C1C0B06B8FDC32AB5914343D0CBEBEE5B1C84008182794CD8C
3	7E02C2AB8B8115B446E80C70CE4A0126E83E0208420E39965272D6497EE16B86
4	BEFC08E950BF7CD6B6BA0026A910328ACF45551DB93180099D0893C8415314F1
5	00019D3C8D5D563A9A24462029171C6D4CF29BCF8CC6D60BF76584A6B4F696ED
6	383F7B8050D7B08DEAEEB3B5B04496669815277968D5A2ACDEF04D37C596E2E7
7	FFEE6E137909EF2140A87619B51CF6C7FBAFC6EF5B8388C8ACF0F7DE5AA8E7BB
8	511855ADB1B5FE79C1C04B565CD40359B5DFA474AC52BE7F4CB2753285B90D0C
9	8F6D5B1C7CC360DC4E4320755684B24726B8C4312A12B329ADC8C2550C3FEB08

B.2.6 Dedicated hash-function 5 (SHA-512)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	256-bit MAC result: leftmost 256 bits of
1	1064FD19C17F8E6FCCAAF9C23CA91E773CA9BD611378B25B2E6B873AD60641B6 E86846C511102008C3972248A0698E0165F03E2CC63D8CECDBA04FD3198EB2BC
2	5A093A41C4E7A01CF19D6E5AEA66B7B7266C200CFE4E7395C6C78CEA539909A4 A0FEEA06858CB8124FC1BD40CC59C3A386066740801E5F532663C3476B9D9508
3	6362967AE7AF0C3E214BA9CCBAABE836EE2ECC9A23F1F3A99B68F46CBC639701 F6848EBFDA1A0C65BEA7CB9E21E8D46178C45E78F6B0477BA753963FC57697EA
4	70322F2B6F1BC5D2CEB4ED9513BA1F596C208B94F0952754657F16DEB4D3FC0E A4DCEDD31A7B04CFF6972B1A4492AE2851A89E9CE48BC603CB3CA86F4934E12E
5	44B71222A187B28033A470D9706AC369D8F9B9A7D0CC1901D1C4A0C91BF08647 C7886B825294BEB500D24D0022EFCB5C276D8A634D5D71C0AA3388DDE35C85D4
6	B55517C950CDBD787F323F8B5AB53AF9BE72847963E6A33C504630466FBC5665 D2EA0D5648C68A7F15101A3B708E58B456BEEC787D6981228A859EE473D186E4
7	BE8EBFEFA2A7C6B9069DA3B6E1ACB4E145DFA3E9E207D36E7A23333930FCCB38 4132C24493262B6C31866576582F964F5210A5D510F3F4569CD2815D570949BE
8	A99D73180AFE77AB36E39C787FE2A99F2BE7FF8A84D37C0FA78E86089D6ED832 FA18F6CA9D43AFA8A9B2C0396D487BF202F0F346AEEA2973A3A3169CE048FB75
9	57AD9E76F0B0C1D645717A244ADC5CAF92330B5A6031530E84F7835260FAD630 BE130ABB504C913C46F91123381C163D40E0F57C5595DF94B81BE280A64E25F0

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
1	46FD8E2F9AEB2846A06D299C63D9E0F8B9A8C78C0B590BC1DAA928A91570757B 4BC5324E7F9823B018AE15131968962D624DEFEDF5259C4555A3AF5E5807398D
2	13B87CA393C6FDCA4F2CEA4E85EFE887E5C8F87AB8C6E51438483BC7EB53FB3C 07BDB4FD6CB817E7BAA71F33BF1BE3C8BC330642834118B90BE57CEED262C6C1
3	282F5C6F3572D91757F50935B5C811B17D1A56AB9E89F8E983B6C760A9975933 DD1930524AE09D479AFC3A00F74CA8E1E397F4F76B51FF635833A802A134F359
4	6339002EE73BF689936FFA48E5196306C0CB810DECCF86C6D17366ECAC4F2ECD 6D3DD6294AE16C2D82A7E2EC5CE18190E9D587758737CE56EC5C848C87BDE0BA
5	D84F86C90E9216E98CCB7EF804FDA09875E2757305D3954DA4FF872DAF849E2F C915F08C1792CBE6A434ACBD9DB1FA94FFA73F566A66D4B4245EEB01BEF47174
6	2D5181E2582F1D8144234AAE0BE4F9D4EF85BFE19D2BA25B131D83892A50FC38 BB479EB69B6C519666ECF8386DF53DAB3F379192076CEBCBE172607B43F38161
7	D6E472517FA6E4053E13613FED434193205483FE7FBA847FF08E62637E7565BE 9DB12650460B193399B09E77C69FD1CCE70AC0C9C3616D4945C8741B3C668410
8	163413FEA2AB10585049821165F5CF04BEFA59AA0DFBB9D44DFC3504FCEA33F9 A89B6FF19DB362950599114F0B7AA0EC91FFDF98A5DFAFDF5E94226E8AF86E30
9	D8B8A9DA12D4A2BB9AE9C4669DB506A7BB4176A3450A572206E57E6E9E9C88AF 3E4409BA11DF8DBC304077F25991CA308C3EC2474AC30FD46D7AE8814DCD8D3D

B.2.7 Dedicated hash-function 6 (SHA-384)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	⟦AC1⟧ 192-bit MAC result: leftmost 192 bits of ⟨AC1⟧
1	15B5D007B61A79EA68EBAA0CB41AE163047A964560B986A4 EC093CBB8F75C4F561989E43292C5E4E89856928896F72E6
2	30DF703C9857253311CD025BA4F2FA16D261EC650C3FA817 BC68A0E2B892F3D191C4589DFC9AE24F175FA2383822F74F
3	B3132BF7E26CA350732457CF47AD35E0075B0A1F862CB8C9 4443F7406AC8DFD785CD912DF786F5CBF6C485AAD25C7F77
4	0A88ABF35B4D4A0F22A47A8D9263DD321F02E19976636F3F 4224AF52FE1C90F4991C8029573685F2D8A617CA177BDCDE
5	35B3DB1E226172815E5F0585A070C6614C9B3D9E3B25C616 B1EB7EC86431C907E97BCE48324182A186B4019EF79B099D
6	E334E6286A0B7A412DFDC7467C4433F1AE6B682EADC102C1 C0924A8D8D322ADF1C14320B67A628777634D6538A549A30
7	A90D42B7FD1027B9A96C2DD3FCDE00F32DB679FB6B2D7152 8D05FE0584487DED9169B9A9BB7A2143182807BCA11B3B59
8	79493626F781C9D38BBC1D313C384F16565089F085AD1892 49B2D4B7FA293B57C28BCA623DB8668FFEB0A5425CBB22F5
9	341228BDD6AC469EFEC34174BAEEC3992DA3E9C3A4A6AB77 D763FECA0DE021EBD1EF1C956AA0A15E29FEDA8461CDD0D7

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	192-bit MAC result: leftmost 192 bits of
1	0FD970CFD0707F73F3D5496968B4CFC913B464DA5D197457 BE0D1A8D5A00FAFE672F107F43C56D3A90C7649400D177E5
2	ED7C428E4E0F9ADED211EEF2D4B3BF5F5E804013A8BFA4AF C4E07A13CC9EB52B788E05D0E8ACBC9CEEC4ABAEFF039987
3	8E3EA04E736A1984DFBCB825DA8C8A9E7FD7C526E522009F 69AEA64436731326D2C5A01AB1ED843A51C5487E7E8AD582
4	AD317904FDB3A4074EEDF48D74F309725778913E91C75299 0634BF30B675C9B19DD04694ED71B726FC88986ED5C81C19
5	9583A3E57837A3E215500B99EF7384EE3290B772C068E496 D176ED22F34F68B96251DC85DD2B1D46E76223BACACC0935
6	43874C75FC95D9228DE74358B3FF5726EF6D35C6E996E07A 6A9F5D83B6DAC03AAC9AEDF61D1586D31E1BCEAFF5934511
7	43D859AD12F5B797CE465A56B28950F5E17A2AE898995753 C7D753EF43E1782E84AB79287E63A4041BDAE89584825A5E
8	29F82232179CAF06AB49446155B52C3AA4D3C11F3A5C2DD9 BD27D09AC76C0CAF1A4CE01108AFAB87062022068791C8DE
9	B3E1B9C71E4D2D661BA97A46473505323E291C6CA6C94CCE CD0851506C55F1A203790F25259B5FB41A9D626D76ABA0CE

B.2.8 Dedicated hash-function 8 (SHA-224)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	112-bit MAC result: leftmost 112 bits of
1	065B86EDEE58E12C4D83C7AD1500EB4CD313DC3AA2147A12F39B7AFE
2	73D5627C9DBDE736841766FE543B7954D59AEE1F5C6823BCE7E77351
3	A4F4EA69DF69D9705D71305817B38AFE1EF6ECF724C3F6743B26A9D2
4	E8E14D49D6C2A88D4A717A276693FF3D03444AC43FE02C99B6919A23
5	38CB47CB00B57B858A0ABB864F05B8E83EE4A250A5794740796FAA05
6	64ACE1CD852195AC765764DBA813749BDECB15498C30FB6CA73E0F71
7	570E89F76E8435B945CF47AF5B054262C654636AEB955FD951076B91
8	CCC8BDEED3869F5D8E2013EDFCE22A36185C5403103F04E586F987C5
9	E0BEA67230DA03039540FA70CB0FBD69464E9DEE3C3FF80CD5D76646

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	112-bit MAC result: leftmost 112 bits of
1	FD5B8A7CD44B62730BF4DE82D7E2D70135E29FAEB1E66933D54BFF68
2	E70AB63A4D92C3B2A6975BB24B9380B5320D7E510107D27CC97F0086
3	36C19C58F5F13B43544424A085EB5E822AF4BD7F32B7AD190B5CA3BB
4	DC53898B38F96269AAF0890A3A4D7E00B03B931C3AF7C80C8BBCF6EC
5	446B9988D8C5B35A20DD6B71B5F1E3E048144CD082C801449B5497EA
6	48E72C0EDC3E52370EF2DAE859A5462D60C735DA3F0B7494AFB0D5AB
7	3E048D2F615BC681D1B2FD59A37F7D8972AE7C8096603B859F771223

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	112-bit MAC result: leftmost 112 bits of
8	E2CC9DFD0A2945F8F2C3003ADD8AAA493BB0C72BFAA82B7CA8B1F289
9	552A67693AB02EC7D0AF18075DA9875B8B5D1DB89F6CFC73EA7151DE

B.2.9 Dedicated hash-function 17 (SM3)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	4AECB47B01AF2E0B420FE3C9B24A0FD8C5EF6D82F41C8009B9133E398BD4A8B5
2	AC5FE9BA773DAF4A9235C858BFC03E6863052E3C58B0C125AED70A999AB775E7
3	F321D3C152400A44CB98D8096084823ADFDDBB57A3B2E947A4024B581020E404
4	CFF373D995C6112F16A585184595E0A99FAD55AF4A55494E37D47B6766C28A05
5	2A139ABD14AFD36C4C483816BA188A2444B3B5B08D95E2B64FA946A1604128AE
6	389066AB696C54D77BBD168A1B5D1D177DB6233751E3994B0804C3F1CC378075
7	D0DAD6CB29EE0D393547D1F716BF83111E00A6513FA213DB522F9E0E1AFF96B4
8	9C1C16CEE95C7BBFEE01C054A72169A0F99A1472B9060074AA5366CC18EBB0D4
9	ED73BAE5E7FA51284FC2704C98DA12FB20992564BF1D8976970E9E07586D1783

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	80DFB76DBF659EDBCA4E703FF92D522B00BECE4B8D698BF644837F2F88EEC613
2	8F319C8AC77E38786B6DDA88CEE90388597CC2FE7772FA3BCF14DA16F1D11C50
3	01B474B0BA28ECBA6EF0A7A1339C96A734351D59D4BB5EC192EDC59BBF4FBC4B
4	9841877F2A1E0E04813B4258ED16FDD080476CC7D6B2C51D7BD2EEBD7F93FA5D
5	57CD60810A8A106EFE2B4B16F3D16F4EF89CB49E0EC5D9C07300BC127603DD35
6	1B3BEF507A9D57B95D2B41E8A56EE88BDC8539D04B224D6BEDA2CE1098E4CBB2
7	271AF8DC5A4DA1AF692306CC198EFE2BC9923BD6D382FCE5C1284D19E6A2721C
8	BFDD8620B52EB6FF3FCF74EF0EE93652161C816DEE86DDF5C822867A2901ADE4
9	883827DE1F419CC0E8502F962A052DF5913212A9AD2DA3BFF00DA80A7CA8DFE4

B.3 MAC Algorithm 2

B.3.1 General

For the examples in B.3, the value $m = L_2/2$ has been selected, that is:

— m = 80 for dedicated hash-functions 1 and 3;

— m = 64 for dedicated hash-function 2;

— m = 128 for dedicated hash-functions 4, 12, 15 and 17;

— m = 256 for dedicated hash-functions 5, 7, 11 and 16;

— ⟦AC1⟩ m = 192 for dedicated hash-functions 6 and 15; and ⟨AC1⟧

— m = 112 for dedicated hash-functions 8 and 13.

The examples are generated using 128-bit keys key 1 and key 2 given in B.1 unless listed otherwise, even for dedicated hash-functions with output length L_2 > 128. In practice, the key length should be at least L_2 bits.

B.3.2 Dedicated hash-function 1 (RIPEMD-160)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	9EBEA41FBC24CD80BF2ECFD5B8C8CC8181D3FCAE
2	75CB722C50024C0E8A7A0DBA7D5C36B86D9D1DD5
3	5B48C1749DDED71EDFE0ADE2B944E808E4A65820
4	F9033064567F541235C3944EE95CB476055985D1
5	B37885405B71E025AF0CB574021A562A62733628
6	5C6429B982C8054B5B3348A0D7D2CE24D7032BC1
7	B0A4A451D0926855E52428E16D1FEAA241C4DD9B
8	1CCEEC5122F08A76EBCD8E3DE88610D942D8A5F6
9	45D61908BFF6039E6DE3C037FDCE6191F19F6410

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	2FDE5DAF7050D14E6D7ACD2254D17FA3A8CBFCDD
2	239C4020610429A8662BF81A2CAAEA47F8EA0A44
3	89EFFB9F5A6BCEAE3C65D0C9803F3464E5E9E349
4	F5FC87FD5702F5D4E7BB634DA4CB4B41CD505B6C
5	5686C00F69E6C868732C67402AA107CEAB513439
6	525EC4893A221EFD9B6DD351059B40C05B4CE2D3
7	B975ED3893FC8D535376EF49211E2E6B1BB30B90
8	BC201FFA581357C271DAE25104167F3DCC97BADC
9	95A875A1D64D55E677D8E4455E1445E7E940F758

B.3.3 Dedicated hash-function 2 (RIPEMD-128)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	64-bit MAC result: leftmost 64 bits of
1	AD9DB2C1E22AF9AB5CA9DBE5A86F67DC
2	3BF448C762DE00BCFA0310B11C0BDE4C
3	F34EC0945F02B70B8603F89E1CE4C78C
4	E8503A8AEC2289D82AA0D8D445A06BDD
5	EE880B735CE3126065DE1699CC136199
6	794DAF2E3BDEEA2538638A5CED154434
7	3A06EEF165B23625247800BE23E232B6
8	9A4F0159C0952DA43A8D466D46B0AF58
9	19B1B3AF333B894DD86D09427116D0AD

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	64-bit MAC result: leftmost 64 bits of
1	8931EEEE56A6B257FD1AB5418183D826
2	DBBCF169EA7419D5BA7BD8EB3673FF2D
3	2C4CD07D3162D6A0E338004D6B6FBC9A
4	75BFB25888F4BB77C77AE83AD0817447
5	B1B5DC0FCB7258758855DD1840FCDCE4
6	670D0F7A697B18F1A8AB7D2A2A00DBC1
7	54E315FDB34A61C0475392E5C7852998
8	AD04354D8AA2A623E72E3594EE3535C0
9	6F9B1C0FC06753618D6DB4B007733795

B.3.4 Dedicated hash-function 3 (SHA-1)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	86C2962E58B3498A2608935AF7726311F2BFB538
2	0497FF21DAE3251DA0ED2F47F5A3B74ABA6B2560
3	6EE2A25F943E3F3EC05225FBB86BA73E2E5D51D2
4	CD4C0D1328DC4A8DC2801001B129AEFC6E0CF9CE
5	89ECE303FAD1E4313950CC3B008CB239B5B85844
6	9DF741057D075D3C4E1533E38A5FF469647194B4
7	188A58390A6EF9827035B81CDF1B5069211F0EE5
8	98A98D6A81FD361030856D2C19742AD8DBC468E7
9	D2986310BA18A78786534882F9C6BCBF06CCE9E3

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	2739B6BE63F539EB70FE250346F6382A2DFA345F
2	A0C2711A6B1DA4CD8F85EF1E6FF7BF70B412B477
3	18F570E864FF903D2773D53C2E114E1A62152953
4	A80845A89BA15E941A2457084BC431F3E47759E1
5	14143EA1057B02D20C0157216190A006E30F3D41
6	DAB4B41BA639B4715889406FE18E0C037017E063
7	AEAEA5415B4F266CB15CBEB844E56AEC2DABAD6D
8	3DBA11471EB4FCCF21BAEB0BFF7E20150132C6CF
9	3BB917B8BD8560E89FF9054FBE096CBACA109D5F

B.3.5 Dedicated hash-function 4 (SHA-256)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	E8A06537F096CCF1A3C425A56CEA054072C4A8DB67BD28CFB02FBEAF84B35F6C
2	DDABFDF46CE93311868B7275E05730AD3E23192A575CC291AE3785289B94A2F3
3	02581EA39A6CF2D752793FD782CFB9CF965BE72B32B322C9551D03510645FB31

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
4	1F12288F42F42661349E5DB741CE19F3B8C3A8149FD4B8981237FA200FEB104F
5	EA4A04E76EEC57D6906098AFA7AE0264072C09F0DB34269B117C68C3ED989C5E
6	6EB683218305A862A1C1EFBA04A2A62DC4EC27886D3C79AFF7C493C2D6DFB080
7	6DC64AC5C5F197EB5463474AA6B329DA9D5B3C6A3324B147469E06F21EB53C41
8	8F4B417527DA9533408D95951ED6504525C9683B45637B246CE25C99ACC64698
9	5E2E0579A26517B06D2933CF62DEA20347A0A8DF9D7C3D200FA5E894ED9C5EDF

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	DCC3C81236AAD92043D1478DF7926E78205F7BBD0C3001854BF9087261ACCE47
2	AEE154DCF83568248DC228C8C3513E9BEEA268B4979FF17CDE5BE484F4919DDA
3	C3B53B9897D72197B240F08715E5C830886FE2F2EFC2E5A8ACD9D5405098863B
4	60CD78CAED2CC9BD3F5BDA6AAA81596B55556660B19A2DF2FF6C48F89C52CD7E
5	6283D8BA031EE52E2D7EBA96287025F161A5219EF1FB59CEBE6133007B35A146
6	3BB625768D0900710F0EF7E854990BBBA35AA9B7BD4B0133656D290992A9BF79
7	98FF69D0048FF552843CB8D5DC686EB2FEC3600D664A464F7B88F7289CC41A78
8	5893F4AD6CEBB85BB90CD4107BF85EEEBAB621C6EEB4EC487780A45DED09F5B2
9	781BFEC8396C6268E5413D76EDAE0C90E6592B624BB4E0FB6137F4DF33FB91D1

B.3.6 Dedicated hash-function 5 (SHA-512)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	256-bit MAC result: leftmost 256 bits of
1	EEB2E7DC1EA7C75966552A7F45C9F30E3FB9A7EA1362BA6D7324DD7DEF461F88 B2B5E433CD7CA25A08554605B0B020C3A865434BABFC140DE5D55C8A94C7FDC6
2	8A507C281F9155A086C97E3BDB14B658F6C901B1948674139389853F55B453C4 83DF9AB807269B286AFD6FBDB6A59E3CF190BB61951D8A89BB0F3D611DC012DC
3	F5B41F81B56D9CEF4BFFFBDD659470CA9DE7348A23DAC136790B028986D13E7D 74DC59759FAEA253D5342ABD56CF6F5859145AD54BCA62E0C45245B7E4FA5C53
4	2EEC2E512FFDAC27444A563B40BFE9EAF594D0C83947A374AC82797DA811466E 8DF8380836446B4B392E4E815B8E84695DB8253230635C18CFEB19CBE1CF012B
5	DCD40B28C684428F83D1E7D457A906DFCAB55C2298B7A242C4F208F205E948A1 BA081A39C6DF60E2279DE4C3D6F82A4490ACCD6679AAF7FEA90DE4BAE06F2C32
6	E78F92E31D7410DD16EC830B477FE703B79925811758F0D3A11F3E4F48DA0C26 87A797EB2E3D7E20026936A87E6903F9D8C93EF3E8FECF2CB2A42A720F301821
7	49BCFE57FA600FDF68562B91E686B02DE81DE25CF4466C707298538980880BFD 339264B48F2BD712127A1C66D97D1B367DDD8656B996CBD8D5B7EDD561328CD5
8	C4979A98F32B6DCAC7718B6089694DBFC6E6ACDF82724F1EFFB277593B716389 ECEA6F4C40EA524C84A1324C6AFABB78C0FE0AA008C1EA3427AF179540FDFBF3
9	376DD55BA616E59FCD6249267577608563C168CBF82CC6A89B83BC9224641B28 CC944C6DFEEDE8CCF7FC6F61FF0D322B3183449677330B6EBD83BDD7B57FB846

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
1	42A0B3DC6AF1D40CF4D58E2E35832C5824AA77E6B685B3E3BBBE69A82C726DD8 B46C861BBE0DADFA359207426187A1675A054CE82905F5A2FCE5495D6BBB6957
2	DAED898D6A86AFD1C622C96D33521175690365330EA0CEEEDB85292F8BBADFD8 67A7C1327356DFD75FA0C38B099BD229C39CF7CF07F479308F52A5C29CD284DA
3	41A98A3AD2B5BF46C000DEB754D93C2D41C4EFBE272163346E8D78A1FA222BD8 046551C4FFE81E8BA0A0DBD746A25066DBFAE79B4040D964D8F2DE181E71212D
4	1296CD85141D1D3BAA6C52ED6A1569925112AAB7820883B3369D278A5711C62C F483045B5F4172841BC917BF92D45EAFC448D975FCAF58D95E8E9AFD127BB7A6
5	C0941AAEA51B8539592403C1CCBB98736F470F5F07C2FBB2374CEFDDE6BF0A34 EAA164B430B59E6921422D6E0C5BEA969FB9F6A7381AC9A8E0107D5BBDD11E3A
6	B87B9328C0041DC4492C5F0AA613EBDC9E1D01156E4E0628705A27B3DCB6EEEC 20E862DA7971DC2B999B6C6952ADB7D8615E68384289076C4B752D0DF393F2E3
7	CB102567BDB4AAB8833419ED3BB95F1875488439B5416FDD466B79CC73BD26E0 9690A3E94CF611D7532128224E225C671B50FC2BED4934516A4955931774B30E
8	41B7CD308733F10CCCC0AE5CAD8AF9E0740317A0EE874489872EC640CC0CB16F 101A12E446F55585555E7AB5128B8D370C006FFA151C7FA35EE10144E1FDD16B
9	2FFE4FEF445D76A2A41E5EDB715170D02ACAC44A580144C17ED65434A876CC99 568E39E97CE78E5325EE376B113C6D7C5247D96AA0DC1D91A0932C81B58D956F

B.3.7 Dedicated hash-function 6 (SHA-384)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	192-bit MAC result: leftmost 192 bits of
1	7BEE4557700A1D8ECE045E8A6FC980F456D2FF4F5AA77BEA 3147F54DC77E8F6A2FA016E9BAA4E105D53B4631CE088CBE
2	33B764BFC7E4A95BB432BD7766C3EB0F0AB686CAC8FD40CD A53A0B0D623D48373F36B321412C4D01E4AAF8BA53C77751
3	67BF47CD4B410564245D335985B5DD404D085E2DB88F2A35 B0782C7FA4AEF3407D489D66EA8914E74752CD1913963139
4	1522B5D65022C1CEBF2425EC914320CCDBE198251CFEA79F 499179A2025185B5CE6241E84A6FF0F9C83820FA83597E62
5	98E3EA52031575D96489C7DB7AE3B4A4AE7D80E789958CE7 99350E2E07D7B852FC8BB3EFF3F2954A2C158BC3C70E8ED1
6	34B80B9F2775DABB019819277302ADAEEC0CA6F0963B2979 314A44724B6318D9213DDFFA2E04B175E1E7D6778FC8A8A5
7	5BF44F677E77FBEBE31516D80CED014DC99E7F51AC4CC41F 6401292990E3668319B137C2F1626C67BB92A1CED7BE15AA
8	7A8610621AB18CA0C87AD25A984C333D3B4BE12E85DEB8E3 88E4656133115FD4710DF4B81D0A526E56553C25E6279131
9	63490CBECD7350ACD6D9D5F485D323440A271555CC3C1E51 F245E0FE4D8DFE6340C6146D2EEB46DFF90A0D1970A30C52

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	192-bit MAC result: leftmost 192 bits of
1	6760086DFFB66B3AA619569BF567D6ACEFD52E2F3D90CC40 103346CB1EE0A493E159785876F8C310B44BF05B64E6B2FF
2	6575DC9AF2E0A9D32D619465C95FED1FF5B1D3E65A2EAD93 405BD2DA82896EE3F9D336B374E5D015594EC44872EBF8C2
3	A6685B72C7545F84405CF20CF17CB67B246FA914C59F335E 7F95B5B11963072777FA3B635AABF86D0D75B83D6365211D
4	0D52B84209956EFD9F39DE27821D328CB0B3FCDEFCF64B99 ED2B65C4DE7A753BCA2361CBB26043649FDFCD8C757E700C
5	9A50FF272D08AB3AD03911B4FB042E0CB8080C18F5938F0C 93340DA508722DBB799C72EA1274B67AD30EAE22E86213B3
6	21BC7FA9F9E23536084CB65EE28727DDD378A1FD6D316D29 BFC3C8A39851EDE817A392D460A628E79A018989249A0CC0
7	F291C145D2F9A10C76C5E40CD4C4027E2688799C95FE4C45 3042DED3EEE4C4331CDC5F571B8642C615DA2A1A854C6EA4
8	23A9161DE7B21284446B49D5038F0D2823A0F05619B243F3 D0E114E3AA9AC905C506A9546E9EEB41F1DD1ABCE8F43B71
9	D056C9491A84401387A18E6953C7157E86C3AD4D3E2B0971 5E91B486EE89C7FE17CE40A10C78EF819433F006F7779443

B.3.8 Dedicated hash-function 7 (WHIRLPOOL)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	256-bit MAC result: leftmost 256 bits of
1	5A77B599D2DB9B6B 8C8E5112DD5F0B88 719D60A4866688C2 DFF624A6EA4ADB62 47556A7EC5191745 4AAD7C63F5F9A7A9 439C7887DDD47DBF E45B0A68ABE62A40
2	177B98F9F215046E 640B8EFE3E723C4E 7233C5E745B72DE9 381D6A3F47E30F95 CCE9FDDCD3853303 B84821F0B070296C EC039B7C86432B34 90520D5FB2585A19
3	F92ECF9FB82E39EC B2D3EC8CFEF76317 A8C4E6F835BD4994 D4D156B68F640D37 61D32FA9C99DCBC9 A7FF32E7643F6B00 8B8D1510FB2A95D9 A1AEC1CC15508634
4	078067C14C1E3930 11BFE58E1E94F03F 9062DA01378760A6 5F7BE1FF041B8087 ECB7BA5D3225B39D EE3B63A616368829 02EB3B4999CB227B 60F1613C174DBBD3
5	05411A95D2A5CDA0 CB4A7339A70E62FF 790D945F25963F15 95E39486BAD88B2F C37973E0FE0EFCB1 5F68EB1E06AA4E3C F15DEFA7E2A4A129 B18F4C5C8B300B63
6	8E2A8C15E9611E57 5BF67165B38B0425 9A30C8C15F9DE729 97391B32575D9C78 653C4CA5CD6FC863 E6A6E0CFDAD1F5B7 60E657B5365ADA57 04BA92ED92277DDF
7	9A7D93D28BA451CD E57570C1CC41E943 D288F3FD112C7E32 22185F2163AE9328 C2528ABA3661909A 91044C6C1C8EA914 2F1FCFB07CE394A9 C0F31862B5A2328B
8	A3676A07D9E79CAB DAA1DA6EAB3FBAD1 28114F4D7E00050A B7167400203585B6 D08E538430F7A050 2E3504C51A5B2449 2BE90403BA08B1A3 D7C1E9E8B70B50E5
9	521EA57548F1068E C0364330ABEEAC85 9E008D976323B1BA 13ECFB405E0909EB B6D4A01127D67966 157DEC38F1D77A35 361BACDA62E506C4 59DE28298AEC9CCE

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
1	FDB6120AFEDAEB87 A4DDC952FE02C1EC B17DDD6647D0FAB6 7194CAA506EED1DB C29A8B3524BA23F3 512765C83734D2A2 6D65139B49FE5A0D 8682E868353DA04B
2	8B738011A43BD813 63C38B941E81975B C2562EC9185B70B5 503D34FEA89B0E3B 66D15236D52C5FF7 35CAA40ED5A49561 D57A5FB202386A21 E226335C78928E0D
3	C97109474261CEDB 4FE524CE8319BD1E 4FAD2DCA54348400 30238EB26812644D 16D1F15A20C9844E AEAE04EBA11D6D67 F1405C58A1779C5C 0E4C2D6C42E7252A
4	A320497D440E9452 846B80EFD4578628 ACD969D64A6EC42E F350F05BE6F604E8 FCE87CB6215E5ADD 711D76EF1C0573CD D5C6E1F133F31472 776D37CE86E9B369
5	E1C734A8E6301FD2 70655F5E6DACCE51 15083D3DA974D411 82C219F74F357E48 1B29C21F864E60AB 49FA8143BE4CF0F0 F12622BA4C6E2325 D09BDA7F634518BB
6	66E060BF156AEC45 4058E4D4A88A0DA8 8FAD6D118D5B7310 60FA0BB68B673DDB 236D7C34E465105D 5AD7C6D7206801CE CC20CB08109486F6 0E6B48FEAC3D5B77
7	608FB970FD10D1BB CEAEE1FA02E44C06 2F1711A214E2594B E57A71FCC419042F DC7313D3D2C5FEC0 32A3DCA366CF7FBF B0EBE2504F2C749C D89E914D87292F11
8	D3B314AD10D07CC4 5708D35526B165A8 9B5AE596D24ABEAC FCD3C0EF2DCCF196 CBB66497472C2E50 A91EE7705695D1AC 0750F2237CD78E8C D01C9891429CA501
9	024F0B3B7A403417 B8191F8383DFFE55 F23F5B1A29E3FC24 BB29097E294FE798 1B0F8AFBC605AD78 3F5BE4F1D2B19C51 CEA40F91A5239B70 4A65ABF1DB054FC9

B.3.9 Dedicated hash-function 8 (SHA-224)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	112-bit MAC result: leftmost 112 bits of
1	B5BC4C46AE5F4E335792CE51304F986D3DD6EAB862933AB3EE7DB5D7
2	63C3DB8305A361388FC52CF4567521303ADA68C31A6FA0638113D594
3	B176AB2549522FD0B93EE32B99BD43C00388DF17FE2B827CE91FD603
4	CD676B859E48A06EC59AE4BF8F341997D9E9FDBA4922ABC983D787A1
5	4CA41164F2AC8F994CA036C8FF516142EED491D82FA6A9C51B44F817
6	3BC5924DA588B993389429D0146FCCD64320CB69E1057C16CDA57AE1
7	184AD4D6C9B1C5BC8FE7D4184C20F724E66C4F158DB41E3025B022D2
8	39FC2867E4979919B1EE5A03D1B15571D69BABA4FED9891D9F97FBF1
9	63859486E22F8C2E90E5F5BF510E732543414F6FB731B0A8E9807249

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	112-bit MAC result: leftmost 112 bits of
1	13247FB84F20471BEC77D777E9648D2C2C4C7B7E132782FC3D1262A4
2	F07713FCB295F3C9997E823B63D0C0B2170D7A6EF533E0F93076D235
3	6F072241CED9E423F04B10F89D656CE36AC7AA6027CB22A6E1216A42
4	9582A2C75DECBCD7E5378D583559B33F74AC61A4A3CF5AA25CD6E3A0
5	47CA643A560DD0A7D06E86218A7E80256CA87FE6A29CCD2081F40EC4
6	02358052E6DF1510771267593C2682814A8E4362E6DF0BDEB59D4DDD
7	B08BCB38E974C3972451DA9805C977DF001CEA225FE3D8EA105938AE

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	112-bit MAC result: leftmost 112 bits of
8	EB50337CE04D4131ABF837780BBBD9674473EA029A08E774F1DC6876
9	6774049ADA46BCC6AD6BCAE615404A22704885B0627F1BDD74D2F8DA

B.3.10 Dedicated hash-function 11 (STREEBOG-512)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	256-bit MAC result: leftmost 256 bits of
1	4F7E2CA889424AEAE71322C57E6EE62AD8C83DCE35732638E08D6ADC976173DE E8E63C545E048660DAF26F73A2C1F8EAFA742AC0B43D6AB7C5DF35FFA890C0CA
2	B89ECC7C524DCA2B11330789F5EA11F602D3B23B634AEEB1D1698BEFD801DD04 EA57D579802AAEF3B7672F06D296CB5E281FD01B5262D1D9B5B465CAA5E971CC
3	02BD67EAEDE2457A81C2CF7E1E11ED14EA606AA0DB7EBF50A720FEB8732A801D 720622DA6685672AB61A94BF1D8AE676CD97CFB8309440AAD4533C20A624F646
4	F776222F39C37AA7E594875C03317DD396B99448E678294982A7F88DD28EE2DB 69DD00514B544B7AE06030C93266392B9BF63B332EDE6AC10647444E4DC795A2
5	7BDB2F9CB9C2184BE466617FC8399C5D574407304CFB42EA45DACD73CB467E27 9F3A1BEEA29CC63063018C9A834FA6F87B2D640DDFD912E3B7E5C6981555E466
6	ADA0F966AEFF2DAE6F080D3496095C17C32EBD22690ADD5224EDE044D6000E81 C478669C7F7CF548B33A64398ED3F8701075AFD5FE880CD5B60FD0E532CE65EC
7	268FEF7EDAECDC4EC446F2D6ACC421330B6F825188F032B0338469D0AC3BA93E 85DE74D03C25455B5BCA6A6FE1E64B304E03094CFDE2E35BA55086CA2515A992
8	FBB4602FFF774425FA118629AC72694C7A0F1CAF5FC29757A032C0E2684E27D9 4AF84D9354F52C32790DF97EA507080802553EA1FFDFE45A7B665D0B47000B44
9	81134E8FC975C523B0AF7AD92CC7FDE5713ADB5729DB31DFA25C2F4C4C631029 3A3D61D2BEE406A64E2C6CA9CC244BF995EE2C3FA732CFAA4F6FD6AAF4E48885

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
1	7A4964EB0E54B581C9EAA219B076F1621CF4E09F7C0B75164D715775CFB9E4B1 51050F3398727AF867C750906AD0E5AA4670ADC172E3F72AE6188C37E9D46914
2	8CB66C29746CA80DAD223765F8070FB7D522C5BE5BB9D63907BE6311F8D4A0DA 54DE8537487DED11E478D79E4B6DC57EB6196647F7EE2C444D68D74ADBF5D516
3	A92AA271649E3DD82BA9AA62577D43F8F018FF137CA599E6BB4AEC4BDE46921C 0DD049DFE2FA8C3A8506FB10282802233540B6501B1218A85C963C22625B0CDA
4	0C74FE09D165B0BC1FB33672FF568E3691502B616864B6277A17FEDC44953910 7ACD74D99AA174FC73F13FA3E77C063C5B33582A414D22FEF1A3946807E6BA81
5	487FB9AD68B5B2CC0995D825D4CE7E4FB505307562BB455CA36B2C0B1CBB7FEA DD749BD35776F1C0B80F315BBE670C7DB70C1932C51306BDC819873751002DD0
6	BEA0D516D1596E28BEF7628CB87D4A5F6F0B9F563FF2F5328082FB463A3EB6FA B9C254EE089F74B9CCA472288A6FF68CEE6510FB2EA651ACB184FF1ADF666D23
7	2580DAC2885EB51745173B6743FFAA654C52CE93E261D44BA0D8DA7903F0C497 379A207EC2A5F9847881DD9E73AA38867A5DA7DF2FD49FCDBECC4C8A957D2A17

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
8	7A301CB9E173C00CDE6CE48F33475070BA2EAAE9CD44B6EA8E94321D208DB042 9B67437387975318933C95130ECBA1D8073B4C9520085612FD51FC42B72303C2
9	C8B49FCDE5BFF171EE7F5B06D857725CE6C081C4C6B43541E4510DF58F5A8E14 0A39820A74CC4D626F4FD978DE2A53477698D200EEADE9FDCF7A513C0882A8F6

B.3.11 Dedicated hash-function 12 (STREEBOG-256)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	20E8626D9A746890B8147B0E9A92655FB552D8F563012E895B1C8C3EAF4E435D
2	5DB2BAA1F8566EAD696ACD2934AA66F02CDC5BB538D0A37C8BBE42259B629C92
3	29C74A370EF9E990ECBD81D96F7326320F56EDA8AC57FC7135A680B738D14BE6
4	BCE21050CD78264EE932DC52AE6DE90C0F0A79B4680A42715434B32DB85A47DE
5	83272FAFA6861BC227CA04B67617020E1B33305B85C61411D4110AAC46214913
6	9ED06C892D7507E9A26CC283678A8970A8E0C427C9DADB02CBF1912B4DA562C6
7	E8515AA95848C54A8AC067833F2D299D0D19162F778567A0EE9E88AD752896F3
8	670EED4C909B0A6A8EEC494DF90E0D1C6EB560F7A7968EB40708B7EC1CEC07BB
9	CE87EBB0C17AE24F81B83127E2B8CD0737405361B3300B52FCC5C5499F5A5080

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	0EAF380211F04DBCB7C0C6F55055DFFDCEAE465586D29BECB74B1C7B9980E6F0
2	47803329FACB8639A8FF1615FDB65BFAEF3DA3CC605B7D9FF77FF67DDC27F54C
3	575C49EEFF380288E250FF8C5DAFAD91D3BBADA3EC99C4A792BBBE426908D470
4	05A932CDB31D5579B838546DDBC8C63AE0A9E8CD5CE2ABD2D79D350C9A9ACF7C
5	0C31B53DBBE494B944B18FD42790816D423A4609DF8A4B7A20700382C147CFAE
6	EEC9A789C459ECA4D681DE03892F8D9126255A0DF31BBB22E15628ADD543DD92
7	BDB41060022CB36AE36BE2A0D3822D69A232E785E9742B7F50693FFB34999354
8	65CA0F004B2F8E85EACE91DE87BCBE38910B56F1898ACF78DD22D1C0890B2FEE
9	7702D5B830CE6CDC5CE1F5AC424A9BEEB2011D21600D7F5C585D09D8C3F4F4D4

B.3.12 Dedicated hash-function 13 (SHA3-224)

For dedicated hash-function 13, the input block length in bits L_1 = 1 152.

key 1: 00112233 44556677 8899AABB CCDDEEFF	
no.	112-bit MAC result: leftmost 112 bits of
1	6EDFB4AA4C6A4A9DE3B72BDCFE3400F172AB2C7B58528B65FA53A0C9
2	99E165D4681E26A3204B308D9A7D7310FEBBE10BA26CC73000D477D8
3	1E37242E0078AF7A01EF36E5B6D69DAC70883D2FC6A7275A13164583
4	939C99EDC6C050A8A72C74B114F59C4AFDEB1BBF35A327EB82586312
5	48FC29D6EF2C8910F381B1D9B0BEDBC29DBD575AE204276B5EA7BEFF
6	EE8ADC45A7181019015AD8054414C6442E4C0FB7F6D5BBAF3C8B0F6D
7	A182680F2D1E31E431F37A52589C12C49C5F850E903F6CAE5973CE94
8	DF67C88ED508121EF023D917804AD526EA8A61F0A9197B28B370662D

key 1: 00112233 44556677 8899AABB CCDDEEFF	
no.	112-bit MAC result: leftmost 112 bits of
9	A2618F749D4D8176C438692775CEFE697AC30479183C0B1C0DC59C87

key 2: 01234567 89ABCDEF FEDCBA98 76543210	
no.	112-bit MAC result: leftmost 112 bits of
1	D9F5665FF92A920E15BE5394BB4B866161B4EA25B25432CEB0D42143
2	1E9F288FAC84AF5D08CB1701FB7C00D08083F33CA37EC47AC06DB9A1
3	617A13DE5D2FB7686C7C39DC8F36F577503DAF14A650FDAE2A1FFA58
4	BDE98D29090D7484E301C6319744D0536BD56FF86A1C00D711D9BDE6
5	0FF43DA7CB865D01899B0198CF8374117356FC980927C71BB7E50A01
6	A0C76202454AA4DD340398B2783D31F85FE314D8D3015C5839AB89C3
7	FDE8EDC09C56A8E6022BE63FF25E2FA332C9C22A69E750A1F88F8D19
8	3547E3BFF8F453919BFA80A032ACDEE13681CBE80CBBCE6F06A75952
9	D6117DF9C940A59340AC07FA87A1B0D561A8B7B62910146EF3FF4665

B.3.13 Dedicated hash-function 14 (SHA3-256)

For dedicated hash-function 14, the input block length in bits L_1 = 1 088.

key 1: 00112233 44556677 8899AABB CCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	EBA7EE20E139E620BDFB13CE48D852EA2AE75D805C8DDB2C79E32F1BA0181420
2	82239EE02055AFD6FED8E15C02E469FA6C0D85F2DAA7E1C71B388B1806D2E4B9
3	E5EFEB277789B2B594858239E161FF69CDD48F4561E46323BFC927C46AAA287B
4	3B954DB9F75358791832CBFACC3AF711E9DBC76F96273517BACE80C86EBA851B
5	A53359920E75374A9999AC4A16205E096491F32E120BD8B0BB2DB37A601D0ED7
6	EE86C6BED38BEDDFCF5AFBBD6189C7B5B60D27616B90300BFCC30172B76986F4
7	0429FA7DE4ED6CEB3BFA60CCFC2C3BC9E752001A949F1A960BC30A57607CB1DB
8	954609D9626AEDD3FD9839140ADD73534B9FFDEB54D738B0DB6DAB29FB4CF3FE
9	A8B37DD9722A304593BD666CEE644D66C56E2E5F90EF5681DC7A16F6CB45F872

key 2: 01234567 89ABCDEF FEDCBA98 76543210	
no.	128-bit MAC result: leftmost 128 bits of
1	D2DAF269764D82F016E406810A6EDAC62B16F9ECA0607A1C33E1B7E208DAA1DD
2	B6685B21408A6BA1187E6D317F83D437BDAA35F0EDD5A08077D6FBF4610B5B3D
3	2E834B9CBD18B8A6183BD542AB8E633532688ECAA59631E73537D4FBBB8F9A51
4	2BBFF566BB69A514B8C7CADD72266FD4E6CE3752C21577A3D9CB456E03A94B17
5	51D8F81119FB49433E7AE2F3B49A94CED6E8A323FEC2E4059D7B5F322852E2FB
6	17AAE63A8F2CF6895DEC9C5C333E7EC613F288803F941CBAA55EFA68248B528A
7	E4BB9C0B333BCEC920AEF770B6D3C9FE34D1E2F67B9D0193B7D87ED13EB53470
8	0A50FE45FC2C53FE65BACCE1CA5B63B2E45BF8F567BD7C80B1C6B4DEEFEAA3FD
9	EC731C01F856EAD2B52ACF11D2231B6AC3779888EAD809E78F824BDF5FC00BD8

B.3.14 Dedicated hash-function 15 (SHA3-384)

For dedicated hash-function 15, the input block length in bits L_1 = 832. The examples below use two keys. Both keys are 256 bits long.

Key 1:	00112233 44556677 8899AABB CCDDEEFF 01234567 89ABCDEF FEDCBA98 76543210
no.	192-bit MAC result: leftmost 192 bits of
1	B355E1175B41864342C2F3987D8C6A838344E7829134C229 FE3EB49CB46881BFEF35B30F980238705F7CF8AACC3A1787
2	9C3422D429F36BD8BBC376DD05370BFEB6542E6CA7C8DA76 3EB805BB88EA5AF5631BDBF85273793A692303E36AB0BDFD
3	E0B79791864AEC24E501F1E11D5E529554CD10642CCC5DF1 B8C34A22B83803A38164DC2A6D23B98A650FF3AFAAF82034
4	0957F6895AE36D93DA8295078484F6D4491EA8D0DA96E62E E6E88323EE7950976228A7F2219BECD7DCF243DA0A02F2E8
5	4515480AC2E41CADC15F62258C45C61F5494A90A8A338016 DE7B7E931831154FFD9AF06391A8DE20867C2D7E34BAAFC2
6	DD0B5C3DA5EBD4332710780A6344A3BD0A933F932D658ED1 AA6ADC5420077E1E27DA01A1EE63DE096780DD2970FA5741
7	519CB4B60AFE8073AED719D76C03085722A279576863C498 AAB26683492F52FC371B0C5AD4D8CDE68DD23DC672C95D2E
8	104A0E2946470EA88C12C203BF558A2C84519C883DE24725 862DB4D3AEBB86E742935A907985AC52A0E2C4740F1F8A69
9	FBA286F8D65C103D7E2CC6E9C098EDE83FD6B5E471253A6F 616B1C3D2079676FC7F3EA9B893B1B88F54E03E33EB4D52C

key 2:	01234567 89ABCDEF FEDCBA98 76543210 00112233 44556677 8899AABB CCDDEEFF
no.	192-bit MAC result: leftmost 192 bits of
1	62C632BF859E4F7AD6EE669AFA1DA9B7AB18F9F7157EAAD1 03814848BE3B2DD1783702A798DCEC76CB61EE3EA46AA9C4
2	1701C53ECCE7E68AE76CBF6E1BE55482D5A47E108F773817 5D9B2B2E978326525F80D6C79BFD92EA6C4EFFDD5089A693
3	5CCFDDDC95C900D0C8D5B15B69DE8486026354A43069159D 5A052F347CA216B98105B1AE767FC18F19E436E660987783
4	21B00A8932AEF4FE4F27DEB22E0B701DC69EE43E2155B6B2 0198735065B32D5EAE4BCEB866F8A48CE7FA9C291D08C0C4
5	E9128A6B5DFC7009B4CCF2B2A90655CEAD746E65FB708659 E4051930C3E4C1BAFDF67EA82539CFB04A6274BFA3F206C8
6	4799F1E6007DA5D1AE348333C80B51742729D421D30ECD5A 142134D2F5A1E770420CF461FC0F64557F25FB1EA674D511
7	C3714C7EC5C13C0F3D4D853BA1E083014109E357D7C589D7 59ED7D2C1F895EAEAA5BADFB7C3F342846B95DD3476F2599
8	B7C929E891097BFAC4413A21572D68BD7A413BEA88277893 0FF4EFB150E02BEBDEBC82748921F695B6270B973375C2FD

key 2:	01234567 89ABCDEF FEDCBA98 76543210 00112233 44556677 8899AABB CCDDEEFF
no.	192-bit MAC result: leftmost 192 bits of
9	2D4CDACD3DDDDF900D176D919229CBF7F780ABB8E71743EC C80A4FA2A0557685C922354A242384CDA93CACB566D2C456

B.3.15 Dedicated hash-function 16 (SHA3-512)

For dedicated hash-functions 16, the input block length in bits L_1 = 576. The examples below use two keys. Both keys are 256 bits.

key 1:	00112233 44556677 8899AABB CCDDEEFF 01234567 89ABCDEF FEDCBA98 76543210
no.	256-bit MAC result: leftmost 256 bits of
1	F79F7322CAAEFD0C28C5C11C222A96148766A0313A4BFD54C5F6BF8A8DC6A784 79C14A97F3EBB062185869B8A608A2047DC5AA7D23211D47CAC2A50365C07C5A
2	716014879E3B9CA1ADA8715E268CC47904A36FAF1F7B0E53C560AF582DA87F89 D9C8AB0EA9D77E39A9339637515901BCC826DAA226FA90968C974B3CD4F24942
3	81DB1FF706C6DE22A7FCF14F6C3F71CD47C97D6C0FA40D02EC21545299F835D6 73B77093EA47E20102DB752C2C8C7BB152C951D06D8DB864625047D375AC1DA5
4	DB1419E1C144DD0DB95AD48A574131DBDB0AF420F9C2DC09C9AE76A3205E1FFB 9114F87637E88CEEBB1BDA839712BFECB2DA1EF65DFBF9D9F85F6A4E5A862685
5	38C5CE56A4D085F3CB7E1E85A216B2B4D9FE325DE4DA69839B377E7B6E2ED248 2923E7D8E55B6282694E6E3DB09E954112A6077F12B83B87C5F0E202F4185696
6	778EC69097AC0EFD413C6C294F68926EAB29C639CD375A00147B007A7FB5D64A 3E514E613D1F2909FAE47CA6B0BB8C3FBE123F4CFEC3AF3C2487975D1D79A8AA
7	71481E181FAFFCDCC3944B1B458667B8FCCFE2FCFE91595BDD11D584AEA63DB9 F28F1EF73DC798DE3F76353C7B61ECA34DD7DB81636A1A6CE4650435563E0E1F
8	354172D69AA40BEA5987CA42505942EF91336591E36780744B219063669AD679 8CF7490A535AE73B58D385F8814D092603841EF8E9827852005B95F51E623CE8
9	48A8EB2822A31FC075D7DF8D86E31BD3272A1C9D52E4B722713A1A565943534D FBE3780E17051DDC276F1CB03470D86290ACF721EE3C7F85523FF5B217D830DE

key 2:	01234567 89ABCDEF FEDCBA98 76543210 00112233 44556677 8899AABB CCDDEEFF
no.	256-bit MAC result: leftmost 256 bits of
1	E2FA5CE28916DE1030AF4BABA110AA1A506FC0A12DF8674FE9CC752598AEE027 B0AEB6C2419C3CFA16CC77C256AE614AC589FB1A8B8FF2411EEB8F347001FD51
2	A830CAC0CF0391D28B0FD90D754E9158A47EE7844B7E5F05211F4AA8985A62F9 2851A91A281B9AF04FA583E64D9B2C0F6B97FE4593CCE47A632147AECEC5C0ED
3	18B4F1DBC8E906F95B9CBB3C3409C7B83DC7F673307C2A0DF21639A6F4A902E9 6170C328F5EF67620DF83EE8B4B8476E1DE3F10381B0655B8744BB2E57C1082D
4	91BE9AFA71F27E8CEB374F02ED66A74CFD3556E8F9D9C5BDDAE9B3F6D2CC732B 26E26244128AABAF41746557C79F449D4D431D5892C426DEE534F095655FC463
5	5EA1271718F679716C6C4856BB6C39798E7DB7195811F5E039C2D50789D6A994 77888B679B4A0F7A0B13DA985DC27C994F22DA02A3B54E949DFEA877D83FF7E7

key 2:	01234567 89ABCDEF FEDCBA98 76543210 00112233 44556677 8899AABB CCDDEEFF
no.	256-bit MAC result: leftmost 256 bits of
6	A8D3E1D7ECCCC5AD85FDFD2231256E1557D25CD67966F80F20017B0CC799702F 5C51271930563CB97DF5ECAA8FB9F96DEDFE1E01251BDBCEDE69699F6407E8C1
7	D9C3521CF57FE20F7E28D006D8E0930E27BF6EFB792860ABED916D89179A1F2C 0C9F36CDC63682246F0A2E744E921942A5574C483F3DDD2E646782EC2AF92AE3
8	18C6A86BC213EA83E5F6AAC6BB0DECFF2E6E8166A71EB36DCE7BDFDE7C787351 044FA903E2BEE0971D3B03B10AD18035511C3F6194BBDB45E199591ED71BED9A
9	F14E0F2E99AD5CC590C9F769D6AB589DD32E9555BD23421A31A7E4AF820488C9 D6DB63421716859902F662DF75F954B10DA1C0E0EC723B93E06DF713CF2544E9

B.3.16 Dedicated hash-function 17 (SM3)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	C8E4E95012EB3D449B5DD0691947986E469E08A3506BB55CCB94A96EBFADA654
2	5FD9F7568A24C438F14B7A22E799B0689FE053ABB76D316202E3C9D10E9EEBE2
3	0933617A88D312F6F9FB4B5F200E31A64D655E92F7FA2A43F55DFEEB8AB6788D
4	9C9A22E8B5797B82CFF9BABA56893CC1D75811C334D198F3AF43401740B824F7
5	A51CE58C52AE29EDD66A53E6AAF0745BF4FEDBDE899973B2D817290E646DF87E
6	DC813339153491AD81477754EB3DF00DBB3CC3E6A69F9CACCE737DB7E61342FF
7	BCA6FA751AECAC5BA3AC49963F6A58F7C2293C6E6923802BC52117A741A49FEE
8	25E034DF9A3AC81599C233440CA6F68F38CA5166438BFA620210EC2F59880C0D
9	34DB1B0452359EA54DA16932E42A662BE88C19C5AD4FE9073867C05A92752024

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	F14B797B559216B73D3816ADFB790250AF3F21198A1AE867123762BB63A00945
2	5BD1836B97C74F88A77BC309E77A269481F53BE9D5C4CE1E40B1C50FE574762E
3	28D8A61BE67D8BF7652C4EDA7092B612F88BE62184F55005C57DDF076E764199
4	E0ACCC4DA77E77D135F17F5CA1EE3E600DAB444FC23ADD6F7E6A54E1B34B26BC
5	429D9030B1D992AD8198E01C13141C2859A913D69DE00CCE9E4A60F00BF276CB
6	AAB294F80562AB234E6226BF7FC3B03F839C7759E60F69735B7E99E50EB94A24
7	08F457B37E5E062AFAFB24DE8D48B92246F1788BAAD4D7B3D11E5F627E33A0D3
8	9F85C779D718A33BDEC2D6E0C1F280FE6A8C12FF2521530A44D168DD4080BC14
9	ED3057AB0DB1E826240FCF8E8760C3DB9338E9AABDAD8B11BB0C040D73E74441

B.4 MAC Algorithm 3

B.4.1 General

For the examples in B.4, the value $m = L_2/2$ has been selected, that is:

AC1 m = 80 for dedicated hash-functions 1 and 3;

m = 64 for dedicated hash-function 2;

m = 128 for dedicated hash-functions 4 and 17;

m = 256 for dedicated hash-function 5;

m = 192 for dedicated hash-function 6; and

m = 112 for dedicated hash-function 8. ⟨AC1⟩

B.4.2 Dedicated hash-function 1 (RIPEMD-160)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	6606EF2D3BBD010F516C65372C3CF0ACF111B3F7
2	F0BC0C81307E17A71F4C40AE0B2AC39FCB23CE12
3	7720FD23925B854F963E8812573CD86EBA61EB66
4	2683D6CE053BA0420E76130EAE2367734B7D2D53
5	DE532D156CBE12464BB6147E99470C471D91F1C6

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	4BD390E9EC460AD4866CCB32D091AFBF73E5B6DA
2	CD2847BAB4636C9BCEADCF3D187122A9199DA670
3	15C3910C42638E5EE6DEBD506BD8C4DB94713A3A
4	04148DCB47728E3E57B836A66043D5145879796E
5	829A24010704DBD0EE34A6D607F7B34829E04E95

B.4.3 Dedicated hash-function 2 (RIPEMD-128)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	64-bit MAC result: leftmost 64 bits of
1	AEB2C45F13C0C6F5F10BE2F1E3E9C322
2	16874D0E17E4F1C290DD749CCEFF7834
3	A289AA06AEB8FC99B989C377BAADD9D4
4	0D80DB68BBF99442DC3D6B83D038DEF3
5	11DC4A6BD375C64F78BB78AD265ED7CD

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	64-bit MAC result: leftmost 64 bits of
1	7248481816B8D3AF29F5C002FF769EA5
2	DFE1E36CE9792476E0889F1BECEF18A9
3	9B4F1D21320F4A327F023947554BFC3B
4	3D2D658D0196E4339F42ADA50DFCFA6F
5	0A34452D9DA70C70183DFFDDB8EEC056

B.4.4 Dedicated hash-function 3 (SHA-1)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	80-bit MAC result: leftmost 80 bits of
1	708F4A226CDE708820643CBEEFDCBE6CB36FB269
2	EAB87BE709D1E5CB62C7489C2B1407130B772760
3	C1BD6F9C908132FEF5187CBE681B42A8C785FBF6
4	F34DEB241D46C6448D67ACE6B8CD4DF00DA23EBC
5	669DED2BD6A1AE0BCFF7D3B74494C1D8161FA0D8

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	80-bit MAC result: leftmost 80 bits of
1	EAF6F9DDBAFD299320FF0FC8E02E2BE62879F341
2	2AAE9DE0A555E7CD7383C27506A467B8DF4E3A33
3	FE6031710329D12090F73F55CFFCC6215F9BEAE9
4	0CCDD9DAB6B0126800EC1CC7A02656E12EDEA42C
5	ABDBC8AAAE4A8CE734432188740A149BDF2D215F

B.4.5 Dedicated hash-function 4 (SHA-256)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	B5FA90A26AC41FF4260BC38142032D5745C2635AEE274846DDAAAAB27ED9E77D
2	3859B97FEF3B4FFF77813AA8E9310BFBB5A015D03564557EFAF3AAFA2888E830
3	8800151E003F4956B4F351862587FCC40DF84D3EC691459DA9E6A8E55E8C3F60
4	5708BD75C1B763B6ACE40E91E1B165C33ECB3EEA3D63B95FD31D895FBF6D46E9
5	3865F84EE189730BB4FC387D42F8A86281DBE3687341C83BB7A5ED362D8094FE

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	5977BDB720997D17297835A553017D478A7F514990215A73A661089A9C5ABC7D
2	E5C2A8C6328E749FAB5E983FCC9D213CB968B2CBFC8634A28DE38C17336DE017
3	CBE66212BD2BF930C303399B2836C8A246FAFF28D34C2B389C6A16DD7F101A48
4	6F7C08BA9AB6BFC73C157B55EEB892F11819B09915818F5B515E1486CB0167E0
5	F519F962E848583855E2FAEFC5D89238B53126719CD62792FE1BA0039C51B3D3

B.4.6 Dedicated hash-function 5 (SHA-512)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	256-bit MAC result: leftmost 256 bits of
1	851274D11F7EEB79F957561C333D89F89F3D5E175E3081F9E607E2A411E0459F FC2526D0BC6FA891DA3B167A2F026E8547120A52B47E8008ED47E1DEFE3F4E7B
2	ED1A70D45DDE8C4CD9FFCF185BD02F2CF6C8EC2DA1E4559EF76A3BA7BD4F5C9E 0F62E7054D78C464667B8F566B19B720E4F00F016928B3E7BB16A85C3144F96D
3	0DC53F9D3046491CFC9CACDEC530EAF1E72124A7AEAB22637726CBC558747538 ABB16565C2E5C786BA5421016B295DA5BEE82C5D8D95CFE7F61C98A91DEB5CD6
4	E02857769CD9A3DDEB560F623A369D52683DE000F6A40474C803D71980262CCC 57E6DF46E85B527848E72AA609B3E9FB8087EFB2BCA73699E9837E2DB5A9849A
5	5ABF362C5C3E4BF2575A4A81D6F4B14D821287AEEC73CE16DA1E728C2CDBB6DC 9F1035C671F27C1D7E14DBA8892B91F3FD49102CAA2AE4FCEAE35B44E210A119

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	256-bit MAC result: leftmost 256 bits of
1	58A261DE6A3B16B5B0704162C4E20EE1AA217C1AC6715EEE850FF24E3F60BCB8 405FCE0DD407B9C690BDFA6ADD431E3AEFEB8C4DA7EACB22245FB598E5A5D09E
2	F4B2B1B1103C653FB97DA019A35D673327F377D8EFCC303F2D71C31858A7C0A1 6CC04FF114052F8A1D2D3CE1AFF87653E5DF05C8EABB0F09E62C97D726E1CADA
3	C3A07A51F53A150363DAC620582936BF20E04717A7C9DADC3334371E25B046A0 4DA428BB257E119EA259ECE2686748535C7067FF0344A42BC3F4E463461B6E8C
4	8C2B5FB4BE793A0516DE3BF6B7220ACD75631AEDCF4D6DC3A3F22EA80782D7C3 47E56C9AA6F09AA310B72056F862866474CE594EDAF0E314CA05775C0F7FCA13
5	F989FCE9ADE63ED03CF66375E9D29793AE753BF216D760D18FE0CA7EF2B113A6 9D3EBF216047A3A9B5CC293728530BD2B8ED5C040762769504210CC7375CC598

B.4.7 Dedicated hash-function 6 (SHA-384)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	192-bit MAC result: leftmost 192 bits of
1	C718D056BE669B442A923C9FF26A5492BF18F8F8B66AE38F 517FB5298FA3C9175DC90B731D9500176A1CE4D8A815D158
2	92086BE3FD7BD4F6922856CBAE4CAB642AB8952508FF942A 3A0584379FE70EC567F49F186B68D1801A3F1B02226CD092
3	25E6E1749494D26165FE3E754CCC72FBCAE97FD79E3C0156 E22E842D814C0F5DAFB36879929F2D8618EDCAF95818055E
4	DBA35A4B7B4925E658F1F98149F349B0F8CAE49FEAB9CDCE BEE3F891B97AF16A892956D80077753D1902627EA9AB32CC
5	D71DFDD1A89F025F0705C73BF3A95351693AD12409038478 3ECAFA9F652718230455FB215CF687A1A072E96D19B49B05

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	192-bit MAC result: leftmost 192 bits of
1	DEF0203A553FB3AA73381296A072E08988AF6ED1BE0C8DF5 8D4EFDEAB720F6EBEE2332C7D4F66266A64F4E1772CD1EE2
2	D7882765D9FC1028C24F40EFB3B7C05524F5A1E398AA84CF 9E7AC78712501EA3C4F4B4184390066B7F925CF3900920A5
3	E43C4311FD59ED61C1FC697B8C6B8C9FBE90C6DB51ACFC38 44BAFA16907714EA7A7CB7520424AB93E7467359418303F5
4	D9BA729CCCE3E987EF4B382876A42FCC7C72A20DA2696CEB 7452C470E890C068C83713CAD018810E402C8576EA9AF993
5	3F2E9ED7E9C6FE6B7A547B3823DFC2BEF98F4792767CD14B 946CEA443CB8B6F873E6F8945D0A6657731E0A2BEE29B0E8

B.4.8 Dedicated hash-function 8 (SHA-224)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	112-bit MAC result: leftmost 112 bits of
1	2E47E46A24AB4E7DF363DCC7A418B27F0DE8D8FB63183AA2CACC34BB
2	BDE7A7F350661ABA5585F0B32C430874D12637E4AE7D2164AA112A6D
3	073B15C8448DF8D65A0BC23546BF02C3A527884297DD2519BA170791
4	18A89EC81098140110E97905BBCC072311929A19DD214118AB636702
5	E7D1F07468AEFE9DC2E5AD9549523213F25043845A3CEA3E9E6A10C6

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	112-bit MAC result: leftmost 112 bits of
1	559000BB6220FEF550B35EE119EA561CC4C393F03BB52E9267DE560D
2	3CFF9B831C517828ADDBFAB6228BE80A1346F905BE32D109E34110FA
3	73774106EA714A5CBD6682D83A6832E80C788E209174807F3273D8F5
4	CF69464136119BD6B0E8C0D7037214F1CD61FA1EBBAF4BDFA4BDFAC3
5	9517EEF881B9D989C402836D84AEDCF93037E70B6AEBEC1D4E311B37

B.4.9 Dedicated hash-function 17 (SM3)

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit MAC result: leftmost 128 bits of
1	B7DA221372494909407DAE9BAA89EC6F97B4B5FF44453C1C634B0CBB642612F2
2	C9333F511344032C70CA41DAB3E335A2AB28DF5B933344E90B4EFC91BA90DC6B
3	336202E1213B63AF5A141FDFCD2B2213C6EDC56CEE7EC12B8A8878182C530FB3
4	E6A3BC8769F5DC27131F4799AD710C8B933347FA65D3A7EFDF354472B81F0238
5	4602CD8E9DA630CB863A14362B0D7DD025D2F5855ABC283DFEEFB1E720B32467

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit MAC result: leftmost 128 bits of
1	94160AE33F288BFB23235E507CB9C9E46C4E074E91E7F4962F1F09C341BE7A53
2	D2D2A779931DFC33B5231422A243C78F012264BA933B1DF1D34D6D6AF2FD34A9
3	1EF7683A13569F6F6596C31E16811B54A51073848527573DB6F685F3AE8D0BC7
4	C60D7367FC8F6715D563A52C50D1DD9FA1E2EDBAB832F4C2D8873A8EED77264B
5	605BF340130372153AA4BBDC9C10E7783D0143C6B7E1D186072FF062D04D8DDD

B.5 MAC Algorithm 4

B.5.1 General

For the examples in B.5, the value m = 128 for KMAC128 and KMACXOF128 and m = 256 for KMAC256 and KMACXOF256.

B.5.2 KMAC128

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit KMAC output
1	78 FC 2B 43 56 E4 93 73 8C 58 AC E6 4B DF 1E 5A
2	F7 27 0E 5F 80 DA D1 D0 AB 6B 0E 3D 1D B8 53 8B
3	A7 F1 D8 D8 12 D3 91 0F D4 EA A4 1D 03 8C 8A C2
4	43 5E 3A D9 B2 D9 D3 58 0E 84 30 AC C0 66 87 EC
5	49 4D C8 12 D7 34 42 08 1D 70 DC 45 C1 F8 7A 9F
6	08 29 6F D8 F7 CC FB A9 58 CC BF BF 81 91 5F 58
7	C0 D7 EC A8 DB 90 A2 C1 FA A1 09 A0 E7 4E 97 D4
8	D7 CE DF 71 06 B4 27 B8 E3 8B 20 8E 2D 41 C9 C9
9	E2 40 1B 13 7D 56 C2 EC 0B 0E 14 95 8D 3C 77 FF

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit KMAC output
1	22 35 21 24 47 88 50 95 C8 F8 62 A1 6B C7 0A 8A
2	98 7F 28 C3 9E 82 43 93 66 D8 04 21 47 FF 71 41
3	FA 13 8E C5 DD C1 1D F6 A0 E6 A7 C1 FF 79 86 39
4	9A 5B B0 58 81 A6 A3 9D 3D 2F 22 57 CC D3 A6 82
5	D3 39 C0 41 78 DE 3D 3D 64 24 D2 5A A5 87 F1 E6
6	30 A9 DF D7 FF A3 2B 7A 3A 2F 85 CC 66 8D 57 A7
7	F6 31 BE 84 63 4A 1E 1D 29 44 77 93 7F 35 B6 47
8	19 11 86 87 EA AD 86 F1 EC 0A EC 33 25 B9 7C 3B
9	42 81 46 19 E9 28 30 DB 57 2C 4D 3F 7D 87 08 02

B.5.3 KMAC256

key 1:	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
no.	256-bit KMAC output
1	66 DD CC 67 27 BC 22 7E 72 57 6C 20 FA C1 C0 30 27 17 72 57 F4 25 C6 A4 74 34 D6 62 E8 C4 CD A4
2	7A 59 00 CF D7 79 F6 2C 30 3A 33 BF B6 55 70 DE 29 52 A7 E0 79 CF 1A 35 AC A8 66 FD 6D 97 D1 F4
3	D2 CE 0D FD 73 1E 17 25 74 0C B7 89 51 A4 3C FD 31 EF D2 91 D5 0A 30 44 FC 0D EA DD 0B 46 30 4B
4	94 D6 A3 E4 AC 54 4D 68 91 0E A3 5A CC 03 79 80 27 F1 FA F7 F9 2E 5E 93 99 25 99 46 63 3F E3 76
5	5C B5 99 0F AB A3 70 7B 59 BD 17 F0 63 EC 37 AA FB 60 36 6A 4E BC 48 21 99 01 6B C2 10 2F AB DC
6	C8 0A D7 70 F6 CB 4F A4 8A 56 82 19 7C 05 A2 80 F8 85 92 2C FE 0D 28 3F 29 D8 12 84 78 4D 37 94
7	C7 AF FD 6B CE 5E 47 31 FD 27 B8 39 36 28 AE 93 06 A2 7A 4D E0 45 81 69 DE 92 E2 F6 C4 6A 2A B3

key 1:	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
no.	256-bit KMAC output
8	4C F8 B6 FE 58 81 AE 32 A7 D4 C4 00 A3 6E DD 12 00 03 1C 38 34 3D 78 6D DF 8F EE 3A 87 68 D5 25
9	B4 DD AC FA B5 56 7B 61 68 34 8E F4 9F FD A7 DC E9 B0 D7 E7 BB 0C 0F 26 3B 48 EB 2C FD 01 1F 39

key 2:	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
no.	256-bit KMAC output
1	1E 3B 05 94 3E FF 46 5C 80 DC 0B 63 60 05 A3 87 1E D0 88 4E A2 B5 02 84 78 98 F9 48 A5 36 61 57
2	E4 B5 55 16 E4 95 B3 D3 34 2D F2 FF 1E 43 06 3B 5B E2 E2 A0 2A 93 BB 87 19 1E 4E CE 72 44 16 BD
3	96 FB 5A B2 28 E2 60 3A EB F6 38 82 CA 7C 07 D1 B4 76 27 DD 4D 31 13 E9 5E A3 FF A4 36 FF 4B 5B
4	B7 0D 6D 12 DF 92 8B F1 61 9F 00 B2 0D BE A4 AE BC 18 39 BF 31 60 33 BA C8 28 E9 46 ED 82 A2 38
5	58 D2 D5 78 62 D6 BC 7A 0C 86 FD 86 F0 FD 4A 53 F6 8B 3B D9 A3 51 74 27 5F E3 B8 35 2E 70 85 09
6	6D D9 82 AB 75 0C CE 6A CF 98 39 2E 3D 9E AF 1D EE 8C 57 49 FB 61 6A BA 1E 15 6A 32 30 A6 3C C7
7	15 DB 3C 5A 39 8E B1 ED 08 50 88 CC 3A B3 81 50 E6 96 A2 CC F8 37 CB 95 6C F8 BF 5D E6 05 5D 3E
8	B3 76 18 BD 7C D3 EE 52 55 8E B4 27 D2 F5 C1 CA 16 3A B3 D3 F6 83 8B 31 58 15 B9 3A F0 DD 41 7C
9	3A C1 88 4C 84 A3 C6 F7 76 B7 22 CB 71 72 AA A7 80 9E D7 75 70 1C 0F C3 BA 5C DA 92 AD 7C 40 CF

B.5.4 KMACXOF128

key 1: 00112233445566778899AABBCCDDEEFF	
no.	128-bit KMACXOF output
1	76 01 31 4E A3 24 54 CA 7C 63 45 27 C2 9B 27 B8
2	E2 12 22 1C 8F 47 C3 7D AC F5 D2 E8 35 CF 4C FC
3	FE F6 EF 32 04 F5 31 78 7E 39 68 84 A4 4B 74 7C
4	E5 12 D6 CC 97 64 B4 E2 C8 16 94 ED 57 80 77 1E
5	8F FB D4 07 75 D2 AB 36 34 84 F1 2F 99 B7 92 C4
6	D5 C2 C2 39 1F 88 37 94 2C 02 25 E0 33 47 DE 11
7	20 5F 33 B3 9B 4F E1 57 39 48 A8 E9 FE 5C EB 50
8	2F 5A 5C A6 CF F4 D3 25 A5 A9 D6 35 B3 06 F0 66
9	5F 8D 7E B7 B9 BC 9C 9A 54 8E AD F8 1D E3 5C 85

key 2: 0123456789ABCDEFFEDCBA9876543210	
no.	128-bit KMACXOF output
1	00 38 D4 48 18 18 99 7F 7C B5 C8 EF E3 7C AD 86
2	25 56 62 E2 9C CC 0E 90 DA 3B 51 B0 C9 84 48 58
3	FF 9E E2 6E 87 EA 6B 58 0F 61 90 F8 06 AD 58 60
4	BD DD C0 A3 99 CE 8B A9 74 CB 52 DC 9B 37 40 FA
5	01 D4 0F C3 45 10 6B 80 40 F3 FB 21 1E FF 96 8E
6	9D 9D 7F 1C 3B C7 30 CD 6B 3C 5D C2 C3 77 7B 02
7	AC 45 A7 E6 F5 C5 6E 6C 68 59 BA 0B 86 3C 6D DC
8	4E 03 05 03 9C FD FF 5A D9 DB E4 64 19 54 20 06
9	44 A7 97 35 A8 B7 E0 63 55 9F 59 AC ED 1F 8E 55

B.5.5 KMACXOF256

key 1:	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
no.	256-bit KMACXOF output
1	FC 21 7D AD EE 0D 90 F8 28 5D D6 24 54 47 9C D8 AA D2 18 6D 80 71 6F 5E 1B 23 EB 2F 63 12 21 4B
2	BF 79 43 B8 20 C2 6F C5 8A 58 E1 B3 02 93 47 21 6D E5 6B E5 75 B0 FB F5 37 D6 87 E7 55 5F 59 D4
3	09 66 FE 45 CE 0D E9 77 ED 3C 29 E6 F0 99 13 F8 7C CA 93 AA AF 16 72 8E F3 58 B1 51 99 29 E2 65
4	98 C7 15 4A 9C 12 C6 F4 C5 3F 69 12 9A 5A 34 79 E0 D0 7B 61 FF 7E A3 C3 B8 A2 11 C5 70 66 F9 F0
5	C6 22 E5 D1 8B 6F 67 5C 8E 29 03 2D 7D 36 17 A3 6C 3C 0C 02 F0 0D AD ED 0E 6B 45 44 E1 99 B8 B5
6	F6 59 86 1A F4 5B 5B 6F 95 36 76 57 DB 3B 44 31 A8 6E 9A B2 C7 12 A7 76 8F C9 96 6F D5 B2 40 54
7	3D 75 B7 20 AA CC DB 31 74 9A E7 7B B9 83 FF D1 1E 9A 56 A2 50 E9 42 5A 30 DE 2E D8 AD EE 79 F8
8	91 CD C5 85 E4 9D 36 68 98 26 5D 13 EF 04 30 09 61 A3 F2 67 01 C6 54 94 A7 E6 9F 56 20 F0 83 06
9	F2 B2 B7 8A 1F C0 15 BF 5D 47 69 73 7B 9E 4D A5 D6 00 57 78 7E FE 84 86 1E CA 8D 18 9A B3 EA E7

key 2:	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
no.	256-bit KMACXOF output
1	B2 9C 77 0D D2 44 83 9F 40 9B BE B0 D7 B2 04 28 FE 79 FC 7E 6B B5 6E 0F C8 99 B6 37 8D C7 33 5A
2	E8 24 5A 77 94 1D 10 72 A4 08 3D D3 61 F3 46 62 FA 2C DF BA EC D4 B6 9D E7 1C 42 4A 81 83 F8 19

key 2:	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
no.	256-bit KMACXOF output
3	EC 28 38 2D AE 41 D9 78 6A A3 D2 4D C4 A1 BC B3 87 32 B6 46 47 D9 C1 55 09 CC 6D C1 BD F5 8B 3E
4	38 08 B3 37 4B 4C 34 D7 29 60 FA 73 87 E0 64 0E C8 9D 5E 2B F2 51 CE EE E1 85 9D AC 52 8E 73 46
5	69 EE 1C 79 13 45 2A 83 71 A1 A7 84 DC 63 21 4F D3 69 06 BB 3F DA FD 87 05 17 DE B9 8F 10 64 01
6	8A 15 0F 5A 9B B1 43 47 79 8A 2B DA D3 AE F6 2A 4C C9 EE B4 28 30 C6 3F 66 11 91 30 84 21 CE DA
7	28 3D 7F B0 C3 D2 38 0F 47 8C 4A F4 B1 E3 4C 10 3D B3 73 6E 70 E4 27 60 FC 8D 7A C6 9E 71 47 57
8	37 5F 28 ED C6 97 41 B4 05 7E A1 7D E1 32 C9 24 A9 1B 8C 1E A2 90 6A 25 04 30 05 9D 02 59 F6 E7
9	62 2B 09 99 22 09 B7 55 C6 52 62 68 22 24 E5 B5 86 B1 1C 60 86 DD 22 D0 1A A2 46 5F E8 97 A0 FD

Annex C
(informative)

Security analysis of the MAC algorithms

This annex discusses the security level of the MAC algorithms in this document. Its goal is to assist the user of this document in selecting one of the mechanisms and the parameter values.

In this annex, $MAC_K(D)$ denotes the MAC for a string D computed using the MAC algorithm key K.

In order to determine the security level of a MAC algorithm, two attack strategies are considered:

— forgery attack: this attack consists of predicting the value of $MAC_K(D)$ for a data string D without initial knowledge of K. If the adversary can do this for a single data string, they are said to be capable of a *forgery*. Practical attacks often require that a forgery is *verifiable*, i.e. that the forged MAC is known to be correct beforehand with probability near 1. Moreover, in many applications the data string has a specific format, which imposes additional constraints on the data string D;

— key recovery attack: this attack consists of finding the MAC algorithm key K itself from a number of data string/MAC pairs. Such an attack is more powerful than forgery, since it allows for arbitrary forgeries.

The feasibility of an attack depends on the number of known and chosen data string/MAC pairs required, and on the number of off-line MAC calculations.

Possible attacks against MAC Algorithms are described below. This list is not exhaustive. The first two attacks are generic, i.e. they apply to any MAC algorithm. The last two attacks apply to any iterated MAC algorithm (for more details, see Reference [10]).

— guessing the MAC: this is a forgery which is not verifiable, and which has a success probability of $\max(1/2^m, 1/2^k)$. This attack applies to all MAC algorithms, and can only be precluded by a judicious choice of m and k.

— brute force key recovery: this attack should require on average 2^{k-1} operations. Verification of such an attack requires of the order of k/m data string/MAC pairs. Again, this attack applies to all MAC algorithms. It can be precluded by a judicious choice of the value k. Alternatively, someone can be prevented from obtaining the k/m data string/MAC pairs which are necessary to identify the key uniquely. For example, if $k = 128$ and $m = 64$, approximately 2^{64} keys correspond to a given data string/MAC pair. If a different key is used to compute each MAC, a brute force key recovery is no more effective than guessing the MAC value.

— birthday forgery[10]: if an adversary knows a sufficient number of data string/MAC pairs, they expect to find two data strings D and D' such that $MAC_K(D) = MAC_K(D')$ *and* the input values of the output transformation in both computations are equal; this is called an internal collision. If D and D' form an internal collision, $MAC_K(D \,||\, Y) = MAC_K(D' \,||\, Y)$ for any string Y. This allows for a forgery after one chosen data string, as an adversary can predict the MAC for $D'||Y$ after having observed the MAC corresponding to $D||Y$. This forgery is on data strings of a specific form, which might not be a concern in all applications, but it should be noted that extensions of this attack exist which allow for greater flexibility in the data strings. The attack requires one chosen data string, and approximately $2^{n/2}$ known data strings and 2^{n-m} chosen data strings.

  The birthday forgery attack can be precluded by prepending a block to the data string which contains a serial number and by making the MAC computation stateful. This means that the implementation has to guarantee that each serial number is used only once for a MAC calculation during the lifetime of the key. This is not feasible in all environments.

— shortcut key recovery: some MAC Algorithms are potentially vulnerable to key recovery attacks based on an internal collision. No shortcut attack has been reported for the MAC algorithms described in this document.

— proofs of security:

  — It has been proven that MAC Algorithm 1 is secure if the following assumption holds[9]:

  The round-function, ϕ, keyed by the initial value, *IV*, and by the additive constant is a pseudorandom function.

  NOTE A pseudo-random function is a function with a secret key that appears to behave as a random function (i.e. is hard to distinguish from a random function) for someone who does not know the secret key.

  — [AC1] It has been proven that MAC Algorithm 2 is secure if the following assumption holds:[8] [AC1]

  The round-function, ϕ, keyed by the initial value, *IV*, is a pseudorandom function (i.e. its output is hard to predict) and the iterated hash-function is second preimage-resistant.

According to Reference [12], the state recovery workload for MAC Algorithm 4 is, $\frac{2^c}{q-1}$, when $m > r$ and $\frac{2^{b-m}}{q-1}$ when $m < r$. where c is the capacity, r is the rate, m is the bit length of MAC value, and q is the number of message and MAC value pairs. Note that $c + r = b$.

For KMAC128, c = 256 and r = 1 344. In most cases, the bit length of MAC value m is much smaller than r. That is, b - m is much larger than c. That is, the state recovery workload is larger than 2^c, even allowing attackers obtaining $q = 2^{64}$ message MAC pairs. For a properly selected key size, i.e. if the bit length of the key is at least c = 256, KMAC128 can provide up to 256-bit security strength.

[AC1] For KMAC256, c = 512 and r = 1 088. [AC1] In most cases, the bit length of MAC value m is much smaller than r. That is, b - m is much larger than c. That is, the state recovery workload is larger than 2^c, even allowing attackers obtaining $q = 2^{64}$ message and MAC pairs. For a properly selected key size, i.e. if the bit length of the key is at least c = 256, KMAC128 can provide up to 256-bit security strength.

Bibliography

[1] ISO 7498-2, *Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture*

[2] ISO/IEC 646, *Information technology — ISO 7-bit coded character set for information interchange*

[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 10181-6, *Information technology — Open Systems Interconnection — Security frameworks for open systems: Integrity framework*

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

[7] AC1 *Text deleted.* AC1

[8] Bellare M., Canetti R., Krawczyk H., "Keying hash functions for message authentication," Advances in Cryptology, Proceedings Crypto'96, LNCS 1109, Koblitz N., Ed., Springer-Verlag, 1996, pp. 1–15.

[9] Bellare M., Canetti R., Krawczyk H. "Pseudorandom functions revisited: The cascade construction and its concrete security," *Proc. 37th Annual Symposium on the Foundations of Computer Science, IEEE*, 1996, pp. 514–523. Full version via http://www-cse.ucsd.edu/users/mihir.

[10] Preneel B., van Oorschot P.C., "MDx-MAC and building fast MACs from hash functions," Advances in Cryptology, Proceedings Crypto'95, LNCS 963, Coppersmith D., Ed., Springer-Verlag, 1995, pp. 1–14.

[11] Kelsey J., Chang J., Perlner R., R. "SHA-3 derived functions: cSHAKE, KMAC, TupleHash And ParallelHash" NIST Special Publication 800-185. December 2016. http://csrc.nist.gov

[12] Bertoni G., Daemen J., Peeters M., Van Assche G., Cryptographic sponge functions. https://keccak.team/files/CSF-0.1.pdf

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Copyright in BSI publications

All the content in BSI publications, including British Standards, is the property of and copyrighted by BSI or some person or entity that owns copyright in the information used (such as the international standardization bodies) and has formally licensed such information to BSI for commercial publication and use.

Save for the provisions below, you may not transfer, share or disseminate any portion of the standard to any other person. You may not adapt, distribute, commercially exploit or publicly display the standard or any portion thereof in any manner whatsoever without BSI's prior written consent.

Storing and using standards

Standards purchased in soft copy format:

- A British Standard purchased in soft copy format is licensed to a sole named user for personal or internal company use only.
- The standard may be stored on more than one device provided that it is accessible by the sole named user only and that only one copy is accessed at any one time.
- A single paper copy may be printed for personal or internal company use only.

Standards purchased in hard copy format:

- A British Standard purchased in hard copy format is for personal or internal company use only.
- It may not be further reproduced – in any format – to create an additional copy. This includes scanning of the document.

If you need more than one copy of the document, or if you wish to share the document on an internal network, you can save money by choosing a subscription product (see 'Subscriptions').

Reproducing extracts

For permission to reproduce content from BSI publications contact the BSI Copyright and Licensing team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member.**

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email cservices@bsigroup.com.

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Useful Contacts

Customer Services
Tel: +44 345 086 9001
Email: cservices@bsigroup.com

Subscriptions
Tel: +44 345 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK