



BSI Standards Publication

IT Security techniques — Hash-functions

Part 3: Dedicated hash-functions

National foreword

This British Standard is the UK implementation of ISO/IEC 10118-3:2018. It supersedes BS ISO/IEC 10118-3:2004+A1:2006, which is withdrawn.

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 secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2018
Published by BSI Standards Limited 2018

ISBN 978 0 580 93283 0

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 November 2018.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

IT Security techniques — Hash- functions —

Part 3: Dedicated hash-functions

*Techniques de sécurité IT — Fonctions de brouillage —
Partie 3: Fonctions de brouillage dédiées*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2018

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

	Page
Foreword	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Symbols	2
4.1 Symbols specified in ISO/IEC 10118-1	2
4.2 Symbols specific to this document	2
5 Requirements	4
6 Models for dedicated hash-functions	4
6.1 Use of models	4
6.2 Round-function model	4
6.3 Sponge model	5
7 Dedicated Hash-Function 1 (RIPEMD-160)	6
7.1 General	6
7.2 Parameters, functions and constants	7
7.2.1 Parameters	7
7.2.2 Byte ordering convention	7
7.2.3 Functions	7
7.2.4 Constants	8
7.2.5 Initializing value	10
7.3 Padding method	10
7.4 Description of the round-function	11
8 Dedicated Hash-Function 2 (RIPEMD-128)	12
8.1 General	12
8.2 Parameters, functions and constants	12
8.2.1 Parameters	12
8.2.2 Byte ordering convention	12
8.2.3 Functions	13
8.2.4 Constants	13
8.2.5 Initializing value	13
8.3 Padding method	13
8.4 Description of the round-function	13
9 Dedicated Hash-Function 3 (SHA-1)	15
9.1 General	15
9.2 Parameters, functions and constants	15
9.2.1 Parameters	15
9.2.2 Byte ordering convention	15
9.2.3 Functions	15
9.2.4 Constants	15
9.2.5 Initializing value	16
9.3 Padding method	16
9.4 Description of the round-function	16
10 Dedicated Hash-Function 4 (SHA-256)	17
10.1 General	17
10.2 Parameters, functions and constants	18
10.2.1 Parameters	18
10.2.2 Byte ordering convention	18
10.2.3 Functions	18
10.2.4 Constants	18
10.2.5 Initializing value	18
10.3 Padding method	18

10.4	Description of the round-function	19
11	Dedicated Hash-Function 5 (SHA-512)	20
11.1	General	20
11.2	Parameters, functions and constants	20
11.2.1	Parameters	20
11.2.2	Byte ordering convention	20
11.2.3	Functions	21
11.2.4	Constants	21
11.2.5	Initializing value	22
11.3	Padding method	22
11.4	Description of the round-function	22
12	Dedicated Hash-Function 6 (SHA-384)	23
12.1	General	23
12.2	Parameters, functions and constants	24
12.2.1	Parameters	24
12.2.2	Byte ordering convention	24
12.2.3	Functions	24
12.2.4	Constants	24
12.2.5	Initializing value	24
12.3	Padding method	24
12.4	Description of the round-function	24
13	Dedicated Hash-Function 7 (WHIRLPOOL)	25
13.1	General	25
13.2	Parameters, functions and constants	25
13.2.1	Parameters	25
13.2.2	Byte ordering convention	25
13.2.3	Functions	25
13.2.4	Constants	27
13.2.5	Initializing value	27
13.3	Padding method	27
13.4	Description of the round-function	27
14	Dedicated Hash-Function 8 (SHA-224)	28
14.1	General	28
14.2	Parameters, functions and constants	28
14.2.1	Parameters	28
14.2.2	Byte ordering convention	28
14.2.3	Functions	28
14.2.4	Constants	29
14.2.5	Initializing value	29
14.3	Padding method	29
14.4	Description of the round-function	29
15	Dedicated Hash-Function 9 (SHA-512/224)	29
15.1	General	29
15.2	Parameters, functions and constants	29
15.2.1	Parameters	29
15.2.2	Byte ordering convention	29
15.2.3	Functions	30
15.2.4	Constants	30
15.2.5	Initializing value	30
15.3	Padding method	30
15.4	Description of the round-function	30
16	Dedicated Hash-Function 10 (SHA-512/256)	30
16.1	General	30
16.2	Parameters, functions and constants	30
16.2.1	Parameters	30

16.2.2	Byte ordering convention.....	31
16.2.3	Functions.....	31
16.2.4	Constants.....	31
16.2.5	Initializing value.....	31
16.3	Padding method.....	31
16.4	Description of the round-function.....	31
17	Dedicated Hash-Function 11 (STREEBOG-512).....	31
17.1	General.....	31
17.2	Parameters, functions and constants.....	32
17.2.1	Parameters.....	32
17.2.2	Byte ordering convention.....	32
17.2.3	Functions.....	32
17.2.4	Constants.....	34
17.2.5	Initializing value.....	34
17.3	Padding method.....	34
17.4	Description of the round-function.....	35
18	Dedicated Hash-Function 12 (STREEBOG-256).....	36
18.1	General.....	36
18.2	Parameters, functions and constants.....	36
18.2.1	Parameters.....	36
18.2.2	Byte ordering convention.....	36
18.2.3	Functions.....	36
18.2.4	Constants.....	36
18.2.5	Initializing value.....	36
18.3	Padding method.....	37
18.4	Description of the round-function.....	37
19	Dedicated Hash-Function 13 (SHA3-224).....	37
19.1	General.....	37
19.2	Parameters, functions and constants.....	37
19.2.1	Parameters.....	37
19.2.2	Byte ordering convention.....	37
19.2.3	Functions.....	37
19.3	Padding method.....	43
19.4	Description of a round-function.....	43
19.5	Output transformation.....	44
20	Dedicated Hash-Function 14 (SHA3-256).....	44
20.1	General.....	44
20.2	Parameters, functions and constants.....	44
20.2.1	Parameters.....	44
20.2.2	Byte ordering convention.....	44
20.2.3	Functions.....	44
20.2.4	Constants.....	44
20.2.5	Initializing value.....	44
20.3	Padding method.....	45
20.4	Description of round-function.....	45
20.5	Output transformation.....	45
21	Dedicated Hash-Function 15 (SHA3-384).....	45
21.1	General.....	45
21.2	Parameters, functions and constants.....	45
21.2.1	Parameters.....	45
21.2.2	Byte ordering convention.....	45
21.2.3	Functions.....	46
21.2.4	Constants.....	46
21.2.5	Initializing value.....	46
21.3	Padding method.....	46
21.4	Description of round-function.....	46

21.5	Output transformation.....	46
22	Dedicated Hash-Function 16 (SHA3-512)	46
22.1	General.....	46
22.2	Parameters, functions and constants	46
22.2.1	Parameters.....	46
22.2.2	Byte ordering convention.....	46
22.2.3	Functions.....	47
22.2.4	Constants.....	47
22.2.5	Initializing value.....	47
22.3	Padding method.....	47
22.4	Description of round-function.....	47
22.5	Output transformation.....	47
23	Dedicated Hash-Function 17 (SM3)	47
23.1	General.....	47
23.2	Parameters, functions and constants	48
23.2.1	Parameters.....	48
23.2.2	Byte ordering convention.....	48
23.2.3	Functions.....	48
23.2.4	Constants.....	48
23.2.5	Initializing value.....	48
23.3	Padding method.....	49
23.4	Description of the round-function	49
Annex A (normative) Object identifiers		51
Annex B (informative) Numerical examples.....		55
Annex C (informative) SHA-3 Extendable-Output Functions.....		245
Bibliography		399

Foreword

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

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

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

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

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

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

This fourth edition cancels and replaces the third edition (ISO/IEC 10118-3:2004), which has been technically revised. It also incorporates the Amendment ISO/IEC 10118-3:2004/Amd1:2006 and Technical Corrigendum ISO/IEC 10118-3:2004/Cor1:2011.

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

- SHA-3, STREEBOG and SM3 hash-functions have been included;
- SHA-3 extendable-output functions have been included;
- cautionary notes for hash-functions with short hash-codes have been added.

A list of all parts in the ISO/IEC 10118 series can be found on the ISO website.

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

IT Security techniques — Hash-functions —

Part 3: Dedicated hash-functions

1 Scope

This document specifies dedicated hash-functions, i.e. specially designed hash-functions. The hash-functions in this document are based on the iterative use of a round-function. Distinct round-functions are specified, giving rise to distinct dedicated hash-functions.

The use of Dedicated Hash-Functions 1, 2 and 3 in new digital signature implementations is deprecated.

NOTE As a result of their short hash-code length and/or cryptanalytic results, Dedicated Hash-Functions 1, 2 and 3 do not provide a sufficient level of collision resistance for future digital signature applications and they are therefore, only usable for legacy applications. However, for applications where collision resistance is not required, such as in hash-functions as specified in ISO/IEC 9797-2, or in key derivation functions specified in ISO/IEC 11770-6, their use is not deprecated.

Numerical examples for dedicated hash-functions specified in this document are given in [Annex B](#) as additional information. For information purposes, SHA-3 extendable-output functions are specified in [Annex C](#).

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 10118-1, *Information technology — Security techniques — Hash-functions — Part 1: General*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 10118-1 and the following apply.

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1

block

bit string of length L_1 , i.e., the length of the first input to the round-function

3.2

word

string of bits

3.3

circulant matrix

matrix with the property that each row, apart from the first, consists of the right cyclic shift by one position of the row immediately above it

3.4

abelian group

group $(G, *)$ such that $a*b = b*a$ for every a and b in G

3.5

field

set of elements S and a pair of operations $(+, *)$ defined on S such that: (i) $a*(b + c) = a*b + a*c$ for every a, b and c in S , (ii) S together with $+$ forms an abelian group (with identity element 0) and (iii) S excluding 0 together with $*$ forms an abelian group

4 Symbols

4.1 Symbols specified in ISO/IEC 10118-1

B_i	byte
D	data
H	hash-code
IV	initializing value
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 the IV
L_X	length (in bits) of a bit string X
$X \oplus Y$	bitwise exclusive-or of bit strings X and Y (where $L_X = L_Y$)
$X Y$	concatenation of strings of bits X and Y in the indicated order
Φ	a round-function, i.e. if X, 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

4.2 Symbols specific to this document

A^i	sequence of constant matrices used in the specification of the round-function defined in Clause 16
A^n	concatenation of n instances of the word A
a_i, a'_i	sequences of indices used in specifications of a round-function
C_i, C'_i	constant words used in the round-functions
C''	8×8 circulant matrix with entries chosen from $\text{GF}(2^8)$ used in the specification of the round-function in Clause 16
c_0	function taking a string of 64 elements of $\text{GF}(2^8)$ as input and giving an 8×8 matrix with entries from $\text{GF}(2^8)$ as output, used in specifying the round-function defined in Clause 16
c_1, c_2, c_3	functions taking an 8×8 matrix of elements of $\text{GF}(2^8)$ as input and giving an 8×8 matrix with entries from $\text{GF}(2^8)$ as output, used in the specification of the round-function defined in Clause 16

c_4	function taking two 8×8 matrices of elements of $\text{GF}(2^8)$ as input and giving an 8×8 matrix with entries from $\text{GF}(2^8)$ as output, used in the specification of the round-function defined in Clause 16
D_i	a block derived from the data string after the padding process
d_i, e_i, f_i, g_i	functions taking either one or three words as input and producing a single word as output, used in specifying round-functions
H_i	a string of L_2 bits which is used in the hashing operation to store an intermediate result
Int_n	an inverse mapping to the mapping Vec_n , i.e. $\text{Int}_n = \text{Vec}_n^{-1}$
$\text{GF}(2^8)$	a field defined as $\text{GF}(2)[x] / p_8(x)$ where $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. The elements of the field are 8-bit strings
M	an 8×8 matrix whose entries are chosen from $\text{GF}(2^8)$
q	number of blocks in the data string after the padding and splitting processes
$R^n()$	operation of right shift by n bits, i.e. if A is a word and n is a non-negative integer then $R^n(A)$ denotes the word obtained by right-shifting the contents of A by n positions
$S^n()$	operation of “circular left shift” by n bit positions, i.e. if A is a word and n is a non-negative integer then $S^n(A)$ denotes the word obtained by left-shifting the contents of A by n places in a cyclic fashion
$S'^n()$	operation of “circular right shift” by n bit positions, i.e. if A is a word and n is a non-negative integer then $S'^n(A)$ denotes the word obtained by right-shifting the contents of A by n places in a cyclic fashion
s	a function, which replaces an element $x \in \text{GF}(2^8)$ with another element $s[x] \in \text{GF}(2^8)$
t_i, t'_i	shift-values used in specifying a round-function
Vec_n	a bijective mapping from $\mathbb{Z}_{2^n}$ to the set of n -bit words, which maps an integer from $\mathbb{Z}_{2^n}$ to its binary representation (i.e. for any integer $z = z_0 + 2z_1 + \dots + 2^{n-1}z_{n-1}$ of the set $\mathbb{Z}_{2^n}$, where $z_j \in \{0,1\}, j = 0, \dots, n-1$, by definition $\text{Vec}_n(z) = (z_{n-1} \ \dots \ z_1 \ z_0)$)
W, X_i, X'_i, Y_i, Z_i	words used to store the results of intermediate computations
W', X'', K_i, Y', Z'	matrices with entries chosen from $\text{GF}(2^8)$ used to store the results of intermediate computations
$\mathbb{Z}_{2^n}$	set of non-negative integers less than 2^n , together with the operations of addition and multiplication modulo 2^n
$\wedge$	bitwise logical AND operation on bit strings, i.e. if A, B are words then $A \wedge B$ is the word equal to bitwise logical AND of A and B
$\vee$	bitwise logical OR operation on bit strings, i.e. if A, B are words then $A \vee B$ is the word equal to bitwise logical OR of A and B
$\neg$	bitwise logical NOT operation on a bit string, i.e. if A is a word then $\neg A$ is the word equal to the bitwise logical NOT of A

- ⊕ addition modulo 2^w operation, where w is the number of bits in a word; i.e. if A and B are w -bit words, then $A \oplus B$ is the word obtained by treating A and B as the binary representations of integers and computing their sum modulo 2^w , where the result is constrained to lie between 0 and $2^w - 1$ inclusive. The value of w is 32 for Dedicated Hash-Functions 1 to 4, defined in [Clauses 7 to 10](#), 64 for Dedicated Hash-Functions 5 and 6, defined in [Clauses 11 and 12](#) and 512 for Dedicated Hash-Functions 11 and 12, defined in [Clauses 17 and 18](#)
- multiplication operation of 8×8 matrices with entries chosen from $\text{GF}(2^8)$; i.e. if A and B are such matrices, then $A \cdot B$ is the matrix obtained by multiplying A and B in the following way. Treat each entry of either A or B as the binary polynomial representation of an integer (for example, the binary polynomial representation of integer 89 (hexadecimal) is $x^7 + x^3 + 1$); treat the multiplication of two of the entries as the remainder when multiplication of the two polynomials is divided by a polynomial $p_8(x)$, where $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$; and treat the sum operation as the operation $\oplus$
- $:=$ a symbol denoting the “set equal to” operation used in procedural specifications of round-functions, where it indicates that the value of the variable (e.g. word or matrix) on the left side of the symbol should be set equal to the value of the expression on the right side of the symbol

5 Requirements

Users who wish to employ a hash-function from this document shall select

- one of the dedicated hash-functions specified below, and
- the length, L_H , of the hash-code H .

NOTE 1 All the hash-functions defined in this document take a bit string as input and give a bit string as output; this is independent of the internal byte ordering convention used within each hash-function.

NOTE 2 The choice of L_H affects the security of the hash-function. All of the hash-functions specified in this document are believed to be collision-resistant hash-functions in environments where performing $2^{L_H/2}$ hash-code computation is deemed to be computationally infeasible.

[Annex A](#) defines object identifiers that shall be used to identify the hash-functions specified in this document.

6 Models for dedicated hash-functions

6.1 Use of models

The 17 dedicated hash-functions specified in this document are defined using two different models. Dedicated Hash-Functions 1 to 12 and 17 are defined using the general round-function-based model defined in ISO/IEC 10118-1, which is further described in [6.2](#). Dedicated Hash-Functions 13 to 16 use the sponge construction model as defined in [6.3](#).

6.2 Round-function model

Dedicated Hash-Functions 1 to 12 and 17 specified in this document are based on the general model for hash-functions given in ISO/IEC 10118-1.

In the specifications of the hash-functions in this document, it is assumed that the padded data string input to the hash-function is in the form of a sequence of bytes. If the padded data string is in the form of a sequence of $8n$ bits, $x_0, x_1, \dots, x_{8n-1}$, then it shall be interpreted as a sequence of n bytes, $B_0, B_1, \dots, B_{n-1}$, in the following way. Each group of eight consecutive bits is considered as a byte, the first bit of a group being the most significant bit of that byte. Hence,

$$B_i = 2^7 x_{8i} + 2^6 x_{8i+1} + \dots + x_{8i+7}$$

for every i ($0 \leq i < n$).

The output transformation for the hash-functions specified in this document is defined so that the hash-code, H , is derived by taking the left-most L_H bits of the final L_2 -bit output string H_q .

Identifiers are defined for each of the 17 dedicated hash-functions specified in this document. The hash-function identifiers for the dedicated hash-functions specified in [Clauses 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22](#) and [23](#) are equal to 31, 32, 33, 34, 35, 36, 37, 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40 and 11 (hexadecimal), respectively. Some hash-function identifiers are also used in the OSI object identifiers assigned in [Annex A](#).

6.3 Sponge model

In [6.3](#), a permutation-based hash-function with sponge construction is specified.

A permutation-based hash-function with sponge construction is defined by a padding method, a permutation and a set of parameters.

The sponge construction[\[4\]](#) is a framework for specifying functions on a binary data with arbitrary output length. The construction employs the following three components:

- an underlying function on fixed-length strings, denoted by f ;
- a parameter called the rate, denoted by r ;
- a padding rule, denoted by pad .

The function that the construction produces from these components is called a sponge function, denoted by $\text{SPONGE}[f, \text{pad}, r]$. A sponge function takes two inputs, a bit string, N , and the bit length, d , of the output string, $\text{SPONGE}[f, \text{pad}, r](N, d)$.

NOTE For further details on the rationale of the sponge construction framework, see Reference [\[5\]](#).

The function, f , maps strings of a single, fixed length, b , to strings of the same length. b is called the width of f . When the underlying function, f , is invertible, i.e. a permutation, it is a permutation-based hash-function with sponge construction.

The rate, r , is a positive integer that is strictly less than the width b . The capacity, c , is the positive integer $b - r$. Thus, $r + c = b$.

In the padding rule, pad is a function that produces padding, i.e. a string with an appropriate length to append to another string. In general, given a positive integer x and a non-negative integer m , the output $\text{pad}(x, m)$ is a string with the property that $m + \text{len}[\text{pad}(x, m)]$ is a positive multiple of x . Within the sponge construction, $x = r$ and $m = \text{len}(N)$, so that the padded input string can be partitioned into a sequence of r -bit strings.

Given these three components, f , pad and r , as described above, the $\text{SPONGE}[f, \text{pad}, r]$ function on (N, d) is specified by $\text{SPONGE}[f, \text{pad}, r](N, d)$. The width b is determined by the choice of f .

$\text{SPONGE}[f, \text{pad}, r](N, d)$

Input: string N , non-negative integer d

Output: string Z , such that $\text{len}(Z) = d$

Steps:

- a) Let $P = N \parallel \text{pad}[r, \text{len}(N)]$.
- b) Let $q = \text{len}(P)/r$.

- c) Let $c = b - r$.
- d) Let $P_0, \dots, P_{q-1}$ be the unique sequence of strings of length r , such that $P = P_0 \parallel \dots \parallel P_{q-1}$.
- e) Let $S = 0^b$.
- f) For i from 0 to $q-1$, let $S = f[S \oplus (P_i \parallel 0^c)]$.
- g) Let Z be the empty string.
- h) Let $Z = Z \parallel \text{Trunc}_r(S)$.
- i) If $d \leq |Z|$, then return $\text{Trunc}_d(Z)$; else, continue.
- j) Let $S = f(S)$ and continue with step h).

Note that the input d determines the number of bits that $\text{SPONGE}[f, \text{pad}, r](N, d)$ returns, but it does not affect their values. In principle, the output can be regarded as an infinite string, whose computation, in practice, is halted after the desired number of output bits is produced.

The parameters of a sponge construction include:

- b , the width;
- r , the rate;
- c , the capacity, such that $b = r + c$; and
- d , the output length.

Here, if notations specified in ISO/IEC 10118-1:2016, Clause 3 are used, r can be considered as L_1 , which is the length of a block of input data (message), while b can be considered as L_2 , which is the output length of the function f . Furthermore, the relation between function f and the round-function Φ as defined in ISO/IEC 10118-1:2016, Clause 3 can be represented as $\Phi(P_i, S_{i-1}) = f[S_{i-1} \oplus (P_i \parallel 0^c)]$, where $P_i = D_i$ is the i th data block, while $S_{i-1} = H_{i-1}$ is the output of the previous execution. The squeezing stage is considered the output transformation.

7 Dedicated Hash-Function 1 (RIPEMD-160)

7.1 General

In [Clause 7](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 1. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{64}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 1 is equal to 31 (hexadecimal).

NOTE 1 Dedicated Hash-Function 1 defined in [Clause 7](#) is commonly called RIPEMD-160[6].

NOTE 2 As a result of a short hash-code length and/or cryptanalytic results, Dedicated Hash-Function 1 does not provide a sufficient level of collision resistance for future digital signature applications and it is, therefore, only used for legacy applications. However, for applications where collision resistance is not required, such as in hash-functions as specified in the ISO/IEC 9797 series or in key derivation functions specified in ISO/IEC 11770-6, its use is not deprecated.

7.2 Parameters, functions and constants

7.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 160$ and L_H is up to 160.

7.2.2 Byte ordering convention

In the specification of the round-function of [Clause 7](#), it is assumed that the block input to the round-function is in the form of a sequence of 32-bit words, each 512-bit block being made up of 16 such words. A sequence of 64 bytes, $B_0, B_1, \dots, B_{63}$, shall be interpreted as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, in the following way. Each group of four consecutive bytes is considered as a word, the first byte of a word being the least significant byte of that word. Hence,

$$Z_i = 2^{24}B_{4i+3} + 2^{16}B_{4i+2} + 2^8B_{4i+1} + B_{4i}, \quad (0 \leq i \leq 15).$$

To convert the hash-code from a sequence of words to a byte-sequence, the inverse process shall be followed.

NOTE The byte-ordering specified here is different from that of [9.2.2](#).

7.2.3 Functions

To facilitate software implementation, the round-function Φ is described in terms of operations on 32-bit words. A sequence of functions $g_0, g_1, \dots, g_{79}$ is used in this round-function, where each function g_i , $0 \leq i \leq 79$, takes three words, X_0, X_1 and X_2 , as input and produces a single word as output.

The functions g_i are defined as follows:

$$\begin{aligned} g_i(X_0, X_1, X_2) &= X_0 \oplus X_1 \oplus X_2, & (0 \leq i \leq 15); \\ g_i(X_0, X_1, X_2) &= (X_0 \wedge X_1) \vee (\neg X_0 \wedge X_2), & (16 \leq i \leq 31); \\ g_i(X_0, X_1, X_2) &= (X_0 \vee \neg X_1) \oplus X_2, & (32 \leq i \leq 47); \\ g_i(X_0, X_1, X_2) &= (X_0 \wedge X_2) \vee (X_1 \wedge \neg X_2), & (48 \leq i \leq 63); \\ g_i(X_0, X_1, X_2) &= X_0 \oplus (X_1 \vee \neg X_2), & (64 \leq i \leq 79). \end{aligned}$$

7.2.4 Constants

Two sequences of constant words, $C_0, C_1, \dots, C_{79}$ and $C'_0, C'_1, \dots, C'_{79}$, are used in this round-function. In a hexadecimal representation (where the most significant bit corresponds to the left-most bit), these are defined as follows:

$$\begin{aligned} C_i &= 00000000, & (0 \leq i \leq 15); \\ C_i &= 5A827999, & (16 \leq i \leq 31); \\ C_i &= 6ED9EBA1, & (32 \leq i \leq 47); \\ C_i &= 8F1BBCDC, & (48 \leq i \leq 63); \\ C_i &= A953FD4E, & (64 \leq i \leq 79); \\ C'_i &= 50A28BE6, & (0 \leq i \leq 15); \\ C'_i &= 5C4DD124, & (16 \leq i \leq 31); \\ C'_i &= 6D703EF3, & (32 \leq i \leq 47); \\ C'_i &= 7A6D76E9, & (48 \leq i \leq 63); \\ C'_i &= 00000000, & (64 \leq i \leq 79). \end{aligned}$$

Two sequences of 80 shift-values are used in this round-function, where each shift-value is between 5 and 15. These sequences are denoted by $(t_0, t_1, \dots, t_{79})$ and $(t'_0, t'_1, \dots, t'_{79})$. Two additional sequences of 80 indices are used in this round-function, where each value in the sequence is between 0 and 15. These sequences are denoted as $(a_0, a_1, \dots, a_{79})$ and $(a'_0, a'_1, \dots, a'_{79})$. All four sequences are defined in [Table 1](#).

Table 1 — Sequences for Hash-Function 1

i	0	1	2	3	4	5	6	7
t_i	11	14	15	12	5	8	7	9
t'_i	8	9	9	11	13	15	15	5
a_i	0	1	2	3	4	5	6	7
a'_i	5	14	7	0	9	2	11	4

i	8	9	10	11	12	13	14	15
t_i	11	13	14	15	6	7	9	8
t'_i	7	7	8	11	14	14	12	6
a_i	8	9	10	11	12	13	14	15
a'_i	13	6	15	8	1	10	3	12

i	16	17	18	19	20	21	22	23
t_i	7	6	8	13	11	9	7	15
t'_i	9	13	15	7	12	8	9	11
a_i	7	4	13	1	10	6	15	3
a'_i	6	11	3	7	0	13	5	10

i	24	25	26	27	28	29	30	31
t_i	7	12	15	9	11	7	13	12
t'_i	7	7	12	7	6	15	13	11
a_i	12	0	9	5	2	14	11	8
a'_i	14	15	8	12	4	9	1	2

i	32	33	34	35	36	37	38	39
t_i	11	13	6	7	14	9	13	15
t'_i	9	7	15	11	8	6	6	14
a_i	3	10	14	4	9	15	8	1
a'_i	15	5	1	3	7	14	6	9

i	40	41	42	43	44	45	46	47
t_i	14	8	13	6	5	12	7	5
t'_i	12	13	5	14	13	13	7	5
a_i	2	7	0	6	13	11	5	12
a'_i	11	8	12	2	10	0	4	13

i	4 8	4 9	5 0	5 1	5 2	5 3	5 4	5 5
t_i	1 1	1 2	1 4	1 5	1 4	1 5	9	8
t'_i	1 5	5	8	1 1	1 4	1 4	6	1 4
a_i	1	9	1 1	1 0	0	8	1 2	4
a'_i	8	6	4	1	3	1 1	1 5	0

i	5 6	5 7	5 8	5 9	6 0	6 1	6 2	6 3
t_i	9	1 4	5	6	8	6	5	1 2
t'_i	6	9	1 2	9	1 2	5	1 5	8
a_i	1 3	3	7	1 5	1 4	5	6	2
a'_i	5	1 2	2	1 3	9	7	1 0	1 4

i	6 4	6 5	6 6	6 7	6 8	6 9	7 0	7 1
t_i	9	1 5	5	1 1	6	8	1 3	1 2
t'_i	8	5	1 2	9	1 2	5	1 4	6
a_i	4	0	5	9	7	1 2	2	1 0
a'_i	1 2	1 5	1 0	4	1	5	8	7

i	7 2	7 3	7 4	7 5	7 6	7 7	7 8	7 9
t_i	5	1 2	1 3	1 4	1 1	8	5	6
t'_i	8	1 3	6	5	1 5	1 3	1 1	1 1
a_i	1 4	1	3	8	1 1	6	1 5	1 3
a'_i	6	2	1 3	1 4	0	3	9	1 1

7.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 160-bit string, represented here as a sequence of five words, Y_0 , Y_1 , Y_2 , Y_3 and Y_4 , in a hexadecimal representation, where Y_0 represents the left-most 32 of the 160 bits:

$Y_0 = 67452301;$
 $Y_1 = \text{EFCDA}89;$
 $Y_2 = 98\text{BADCFE};$
 $Y_3 = 10325476;$
 $Y_4 = \text{C3D2E1F0}.$

7.3 Padding method

The data string, D , needs to be padded to make it contain a number of bits which is an integer multiple of 512. The padding procedure operates as follows.

- D is concatenated with a single "1" bit.
- The result of the previous step is concatenated with between zero and 511 "0" bits such that the length (in bits) of the resultant string is congruent to 448 modulo 512. More explicitly, if the original length of D is L_D and letting r be the remainder when L_D is divided by 512, then the number

of concatenated zeros is equal to either $447 - r$ (if $r \leq 447$) or $959 - r$ (if $r > 447$). The result will be a bit string whose length will be 64 bits short of an integer multiple of 512 bits.

- c) Divide the 64-bit binary representation of L_D into two 32-bit strings, one representing the “most significant half” of L_D and the other the “least significant half”. Now concatenate the string resulting from the previous step with these two 32-bit strings, with the “least significant half” preceding the “most significant half”.

In the description of the round-function which follows, each 512-bit data block D_i , $1 \leq i \leq q$, is treated as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, where Z_0 corresponds to the left-most 32 bits of D_i .

NOTE The concatenation of the two 32-bit strings of L_D in step c) is such that these two 32-bit strings are used directly as the words Z_{14} and Z_{15} of the last data block; based on the byte ordering convention in 7.2.2, the least significant byte of L_D is the left-most byte and the most significant byte of L_D is the right-most byte.

7.4 Description of the round-function

The round-function, Φ , operates as follows.

NOTE In this description, the symbols $W, X_0, X_1, X_2, X_3, X_4, X'_0, X'_1, X'_2, X'_3$ and X'_4 are used to denote 11 distinct words which contain values required in the computations.

- a) Suppose the 512-bit (first) input to Φ is contained in $Z_0, Z_1, \dots, Z_{15}$, where Z_0 contains the left-most 32 of the 512 bits. Suppose also that the 160-bit (second) input to Φ is contained in five words, Y_0, Y_1, Y_2, Y_3 and Y_4 .
- b) Let $X_0 = Y_0, X_1 = Y_1, X_2 = Y_2, X_3 = Y_3$ and $X_4 = Y_4$.
- c) Let $X'_0 = Y_0, X'_1 = Y_1, X'_2 = Y_2, X'_3 = Y_3$ and $X'_4 = Y_4$.
- d) For $i = 0$ to 79, do the following four steps in the order specified:
- 1) $W = S^{t_i} [X_0 \cup g_i(X_1, X_2, X_3) \cup Z_{a_i} \cup C_i] \cup X_4$;
 - 2) $X_0 = X_4; X_4 = X_3; X_3 = S^{10}(X_2); X_2 = X_1; X_1 = W$;
 - 3) $W = S^{t'_i} [X'_0 \cup g_{79-i}(X'_1, X'_2, X'_3) \cup Z_{a'_i} \cup C'_i] \cup X'_4$;
 - 4) $X'_0 = X'_4; X'_4 = X'_3; X'_3 = S^{10}(X'_2); X'_2 = X'_1; X'_1 = W$.
- e) Let $W = Y_0, Y_0 = Y_1 \cup X_2 \cup X'_3, Y_1 = Y_2 \cup X_3 \cup X'_4, Y_2 = Y_3 \cup X_4 \cup X'_0, Y_3 = Y_4 \cup X_0 \cup X'_1$ and $Y_4 = W \cup X_1 \cup X'_2$.
- f) The five words, Y_0, Y_1, Y_2, Y_3 and Y_4 , represent the output of the round-function Φ . After the final iteration of the round-function, the five words, Y_0, Y_1, Y_2, Y_3 and Y_4 , shall be converted to a sequence of 20 bytes using the inverse of the procedure specified in 7.2.2 and where Y_0 shall yield the first four bytes, Y_1 the next four bytes and so on. Thus, the first (left-most) byte will correspond to the least significant byte of Y_0 and the 20th (right-most) byte will correspond to the most significant byte of Y_4 . The 20 bytes shall be converted to a string of 160 bits using the inverse of the procedure specified in Clause 6, i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 160th (right-most) bit will correspond to the least significant bit of the 20th (right-most) byte.

Figure 1 shows steps 1) and 2) of item d) of the round-function Φ in Dedicated Hash-Function 1 (RIPEMD-160) [the other half, i.e. steps 3) and 4) is similar]. In the round-function Φ , steps 1) to 4) of item d) are used 80 times ($i = 0, \dots, 79$).

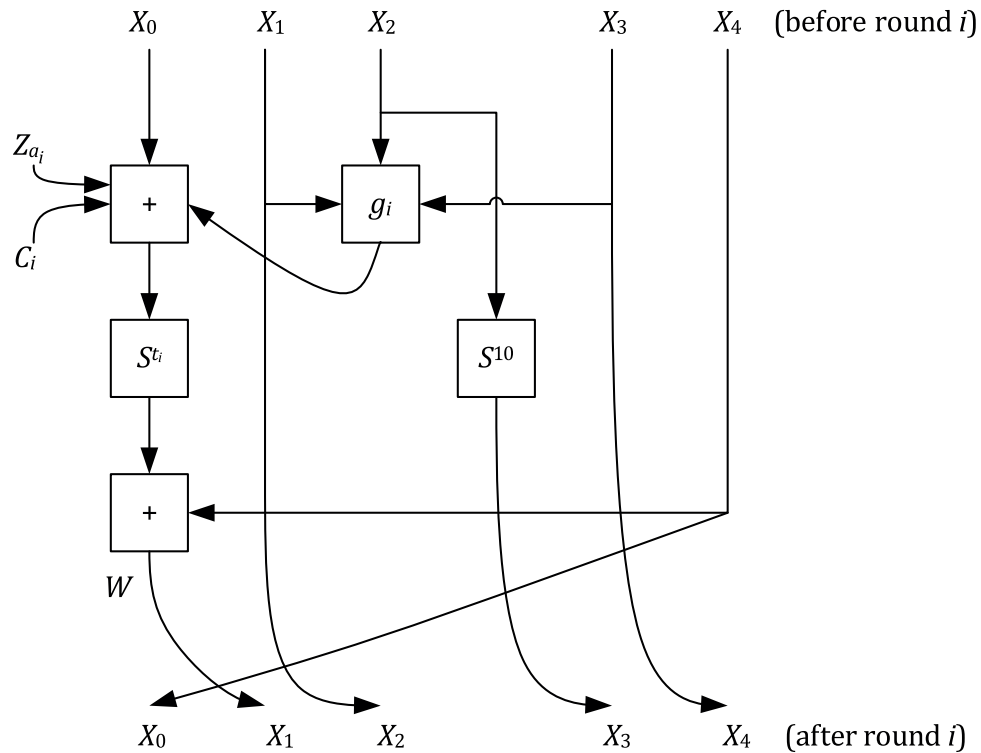


Figure 1 — Part of the round-function in Dedicated Hash-Function 1

8 Dedicated Hash-Function 2 (RIPEMD-128)

8.1 General

In [Clause 8](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 2. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{64}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 2 is equal to 32 (hexadecimal).

NOTE 1 Dedicated Hash-Function 2 defined in [Clause 8](#) is commonly called RIPEMD-128[6]. This hash-function is only used in applications where a hash-code containing 128 bits or less is considered adequately secure.

NOTE 2 As a result of a short hash-code length and/or cryptanalytic results, Dedicated Hash-Function 2 does not provide a sufficient level of collision resistance for future digital signature applications and it is, therefore, only used for legacy applications. However, for applications where collision resistance is not required, such as in hash-functions as specified in the ISO/IEC 9797 series or in key derivation functions specified in ISO/IEC 11770-6, its use is not deprecated.

8.2 Parameters, functions and constants

8.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 128$ and L_H is up to 128.

8.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of [Clause 7](#).

8.2.3 Functions

To facilitate software implementation, the round-function, Φ , is described in terms of operations on 32-bit words. A sequence of functions, $g_0, g_1, \dots, g_{63}$, is used in this round-function, where each function g_i , $0 \leq i \leq 63$, takes three words, X_0, X_1 and X_2 , as input and produces a single word as output.

The functions g_i are defined to be the same as the first 64 of the functions defined in [7.2.3](#).

8.2.4 Constants

Two sequences of constant words, $C_0, C_1, \dots, C_{63}$ and $C'_0, C'_1, \dots, C'_{63}$, are used in this round-function. In a hexadecimal representation (where the most significant bit corresponds to the left-most bit), these are defined as follows:

$$\begin{aligned} C_i &= 00000000, & (0 \leq i \leq 15); \\ C_i &= 5A827999, & (16 \leq i \leq 31); \\ C_i &= 6ED9EBA1, & (32 \leq i \leq 47); \\ C_i &= 8F1BBCDC, & (48 \leq i \leq 63). \\ \\ C'_i &= 50A28BE6, & (0 \leq i \leq 15); \\ C'_i &= 5C4DD124, & (16 \leq i \leq 31); \\ C'_i &= 6D703EF3, & (32 \leq i \leq 47); \\ C'_i &= 00000000, & (48 \leq i \leq 63). \end{aligned}$$

Two sequences of 64 shift-values are also used in this round-function, where each shift-value is between 5 and 15. These sequences are denoted by $(t_0, t_1, \dots, t_{63})$ and $(t'_0, t'_1, \dots, t'_{63})$ and they are defined to be equal to the first 64 values of the corresponding sequences defined in [7.2.4](#).

Finally, two further sequences of 64 indices are used in this round-function, where each value in the sequence is between 0 and 15. These sequences are denoted by $(a_0, a_1, \dots, a_{63})$ and $(a'_0, a'_1, \dots, a'_{63})$ and they are defined to be equal to the first 64 values of the corresponding sequences defined in [7.2.4](#).

8.2.5 Initializing value

For this hash-function, the initializing value, IV , shall always be the following 128-bit string, represented here as a sequence of four words, Y_0, Y_1, Y_2 and Y_3 , in a hexadecimal representation, where Y_0 represents the left-most 32 of the 128 bits:

$$\begin{aligned} Y_0 &= 67452301; \\ Y_1 &= EFCDBA89; \\ Y_2 &= 98BADCFE; \\ Y_3 &= 10325476. \end{aligned}$$

8.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [7.3](#).

8.4 Description of the round-function

The round-function, Φ , operates as follows.

NOTE In this description, the symbols $W, X_0, X_1, X_2, X_3, X'_0, X'_1, X'_2$ and X'_3 are used to denote nine distinct words which contain values required in the computations.

- Suppose the 512-bit (first) input to Φ is contained in $Z_0, Z_1, \dots, Z_{15}$, where Z_0 contains the left-most 32 of the 512 bits. Suppose also that the 128-bit (second) input to Φ is contained in four words, Y_0, Y_1, Y_2 and Y_3 .
- Let $X_0 = Y_0, X_1 = Y_1, X_2 = Y_2$ and $X_3 = Y_3$.

- c) Let $X'_0 := Y_0$, $X'_1 := Y_1$, $X'_2 := Y_2$ and $X'_3 := Y_3$.
- d) For $i := 0$ to 63, do the following four steps in the order specified:
- 1) $W := S^{t_i} [X_0 \uplus g_i(X_1, X_2, X_3) \uplus Z_{a_i} \uplus C_i]$;
 - 2) $X_0 := X_3$; $X_3 := X_2$; $X_2 := X_1$; $X_1 := W$;
 - 3) $W := S^{t'_i} [X'_0 \uplus g_{63-i}(X'_1, X'_2, X'_3) \uplus Z_{a'_i} \uplus C'_i]$;
 - 4) $X'_0 := X'_3$; $X'_3 := X'_2$; $X'_2 := X'_1$; $X'_1 := W$.
- e) Let $W := Y_0$, $Y_0 := Y_1 \uplus X_2 \uplus X'_3$, $Y_1 := Y_2 \uplus X_3 \uplus X'_0$, $Y_2 := Y_3 \uplus X_0 \uplus X'_1$ and $Y_3 := W \uplus X_1 \uplus X'_2$.
- f) The four words, Y_0 , Y_1 , Y_2 and Y_3 , represent the output of the round-function Φ . After the final iteration of the round-function, the four words, Y_0 , Y_1 , Y_2 and Y_3 , shall be converted to a sequence of 16 bytes using the inverse of the procedure specified in 7.2.2 and where Y_0 shall yield the first four bytes, Y_1 the next four bytes and so on. Thus, the first (left-most) byte will correspond to the least significant byte of Y_0 and the 16th (right-most) byte will correspond to the most significant byte of Y_3 . The 16 bytes shall be converted to a string of 128 bits using the inverse of the procedure specified in Clause 6, i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 128th (right-most) bit will correspond to the least significant bit of the 16th (right-most) byte.

Figure 2 shows steps 1) and 2) of item d) of the round-function Φ in Dedicated Hash-Function 2 (RIPEMD-128) [the other half, i.e. steps 3) and 4) is similar]. In the round-function Φ , steps 1) to 4) of item d) are used 64 times ($i = 0, \dots, 63$).

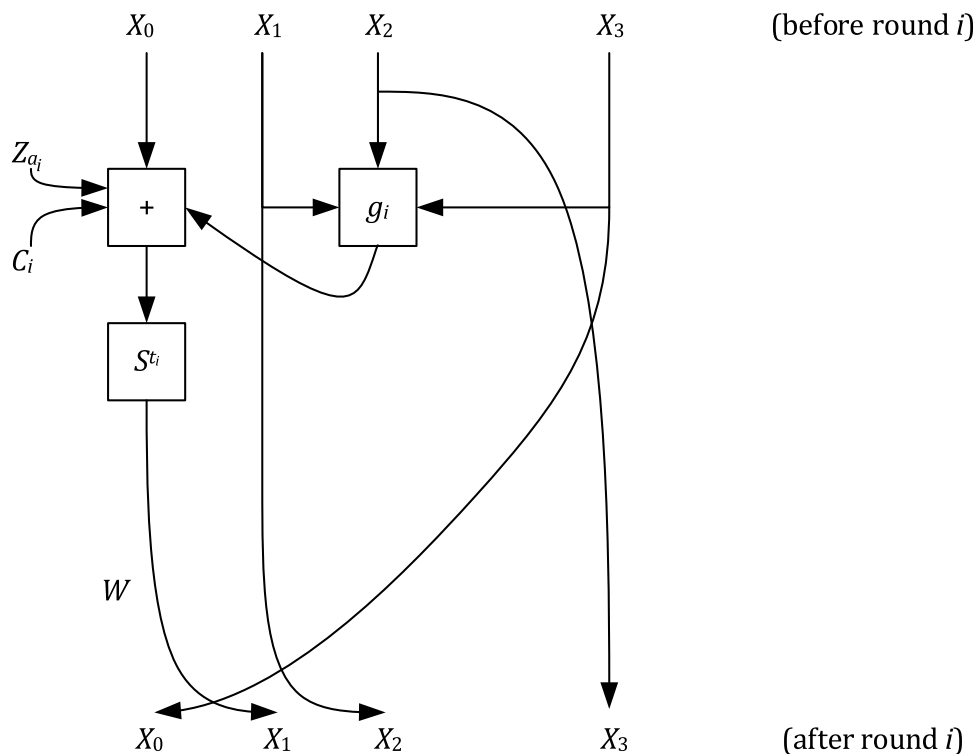


Figure 2 — Part of the round-function in Dedicated Hash-Function 2

9 Dedicated Hash-Function 3 (SHA-1)

9.1 General

In [Clause 9](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define the Dedicated Hash-Function 3. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{64}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 3 is equal to 33 (hexadecimal).

NOTE 1 Dedicated Hash-Function 3 defined in [Clause 9](#) is commonly called SHA-1^[1].

NOTE 2 As a result of a short hash-code length and/or cryptanalytic results, Dedicated Hash-Function 3 does not provide a sufficient level of collision resistance for future digital signature applications and it is, therefore, only used for legacy applications. However, for applications where collision resistance is not required, such as in hash-functions as specified in the ISO/IEC 9797 series or in key derivation functions specified in ISO/IEC 11770-6, its use is not deprecated.

9.2 Parameters, functions and constants

9.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 160$ and L_H is up to 160.

9.2.2 Byte ordering convention

In the specification of the round-function of [Clause 9](#), it is assumed that the block input to the round-function is in the form of a sequence of 32-bit words, each 512-bit block made up of 16 such words. A sequence of 64 bytes, $B_0, B_1, \dots, B_{63}$, shall be interpreted as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, in the following way, where each group of four consecutive bytes is considered as a word, the first byte of a word being the most significant byte of that word. Hence,

$$Z_i = 2^{24}B_{4i} + 2^{16}B_{4i+1} + 2^8B_{4i+2} + B_{4i+3}, \quad (0 \leq i \leq 15).$$

To convert the hash-code from a sequence of words to a sequence of bytes, the inverse process shall be followed.

NOTE The byte-ordering specified here is different from that of [7.2.2](#).

9.2.3 Functions

To facilitate software implementation, the round-function Φ is described in terms of operations on 32-bit words. A sequence of functions, $f_0, f_1, \dots, f_{79}$, is used in this round-function, where each function, f_i , $0 \leq i \leq 79$, takes three words, X_0, X_1 and X_2 , as input and produces a single word as output.

The functions f_i are defined as follows:

$$\begin{aligned} f_i(X_0, X_1, X_2) &= (X_0 \wedge X_1) \vee (\neg X_0 \wedge X_2), & (0 \leq i \leq 19); \\ f_i(X_0, X_1, X_2) &= X_0 \oplus X_1 \oplus X_2, & (20 \leq i \leq 39); \\ f_i(X_0, X_1, X_2) &= (X_0 \wedge X_1) \vee (X_0 \wedge X_2) \vee (X_1 \wedge X_2), & (40 \leq i \leq 59); \\ f_i(X_0, X_1, X_2) &= X_0 \oplus X_1 \oplus X_2, & (60 \leq i \leq 79). \end{aligned}$$

9.2.4 Constants

A sequence of constant words, $C_0, C_1, \dots, C_{79}$, is used in this round-function. In a hexadecimal representation (where the most significant bit corresponds to the left-most bit), these are defined as follows:

$$\begin{aligned}
C_i &= 5A827999, & (0 \leq i \leq 19); \\
C_i &= 6ED9EBA1, & (20 \leq i \leq 39); \\
C_i &= 8F1BBCDC, & (40 \leq i \leq 59); \\
C_i &= CA62C1D6, & (60 \leq i \leq 79).
\end{aligned}$$

9.2.5 Initializing value

For this round-function the initializing value, IV , shall always be the following 160-bit string, represented here as a sequence of five words, Y_0 , Y_1 , Y_2 , Y_3 and Y_4 , in a hexadecimal representation, where Y_0 represents the left-most 32 of the 160 bits:

$$\begin{aligned}
Y_0 &= 67452301; \\
Y_1 &= EFCDA889; \\
Y_2 &= 98BADCFE; \\
Y_3 &= 10325476; \\
Y_4 &= C3D2E1F0.
\end{aligned}$$

9.3 Padding method

The data string, D , needs to be padded to make it contain a number of bits which is an integer multiple of 512. The padding procedure operates as follows.

- D is concatenated with a single “1” bit.
- The result of the previous step is concatenated with between zero and 511 “0” bits, such that the length (in bits) of the resultant string is congruent to 448 modulo 512. More explicitly, if the original length of D is L_D and letting r be the remainder when L_D is divided by 512, then the number of concatenated zeros is equal to either $447 - r$ (if $r \leq 447$) or $959 - r$ (if $r > 447$). The result will be a bit string whose length will be 64 bits short of an integer multiple of 512 bits.
- Concatenate the string resulting from the previous step with the 64-bit binary representation of L_D , most significant bit first.

In the description of the round-function which follows, each 512-bit data block D_i , $1 \leq i \leq q$, is treated as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, where Z_0 corresponds to the left-most 32 bits of D_i .

NOTE The concatenation of the 64-bit string of L_D in step c) is such that the most significant 32-bit string and the least significant 32-bit string of L_D are used respectively as the words Z_{14} and Z_{15} of the last data block. Based on the byte ordering convention in 9.2.2, the most significant byte of L_D is the left-most byte and the least significant byte of L_D is the right-most byte.

9.4 Description of the round-function

The round-function, Φ , operates as follows.

NOTE In this description, the symbols $W, X_0, X_1, X_2, X_3, X_4, Z_0, Z_1, \dots, Z_{79}$ are used to denote 86 distinct words which contain values required in the computations.

- Suppose the 512-bit (first) input to Φ is contained in $Z_0, Z_1, \dots, Z_{15}$, where Z_0 contains the left-most 32 of the 512 bits. Suppose also that the 160-bit (second) input to Φ is contained in five words, Y_0, Y_1, Y_2, Y_3 and Y_4 .
- For $i = 16$ to 79, let $Z_i = S^1(Z_{i-3} \oplus Z_{i-8} \oplus Z_{i-14} \oplus Z_{i-16})$.
- Let $X_0 = Y_0, X_1 = Y_1, X_2 = Y_2, X_3 = Y_3$ and $X_4 = Y_4$.
- For $i = 0$ to 79, do the following two steps:
 - $W = S^5(X_0) \cup f_i(X_1, X_2, X_3) \cup X_4 \cup Z_i \cup C_i$

- 2) $X_4 := X_3$; $X_3 := X_2$; $X_2 := S^{30}(X_1)$; $X_1 := X_0$; $X_0 := W$.
- e) Let $Y_0 := Y_0 \cup X_0$, $Y_1 := Y_1 \cup X_1$, $Y_2 := Y_2 \cup X_2$, $Y_3 := Y_3 \cup X_3$ and $Y_4 := Y_4 \cup X_4$.
- f) The five words, Y_0 , Y_1 , Y_2 , Y_3 and Y_4 , represent the output of the round-function Φ . After the final iteration of the round-function, the five words, Y_0 , Y_1 , Y_2 , Y_3 and Y_4 , shall be converted to a sequence of 20 bytes using the inverse of the procedure specified in 9.2.2 and where Y_0 shall yield the first four bytes, Y_1 the next four bytes and so on. Thus, the first (left-most) byte will correspond to the most significant byte of Y_0 and the 20th (right-most) byte will correspond to the least significant byte of Y_4 . The 20 bytes shall be converted to a string of 160 bits using the inverse of the procedure specified in Clause 6, i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 160th (right-most) bit will correspond to the least significant bit of the 20th (right-most) byte.

Figure 3 shows steps 1) and 2) of item d) of the round-function Φ in Dedicated Hash-Function 3 (SHA-1). In the round-function Φ , steps 1) and 2) of item d) are used 80 times ($i = 0, \dots, 79$).

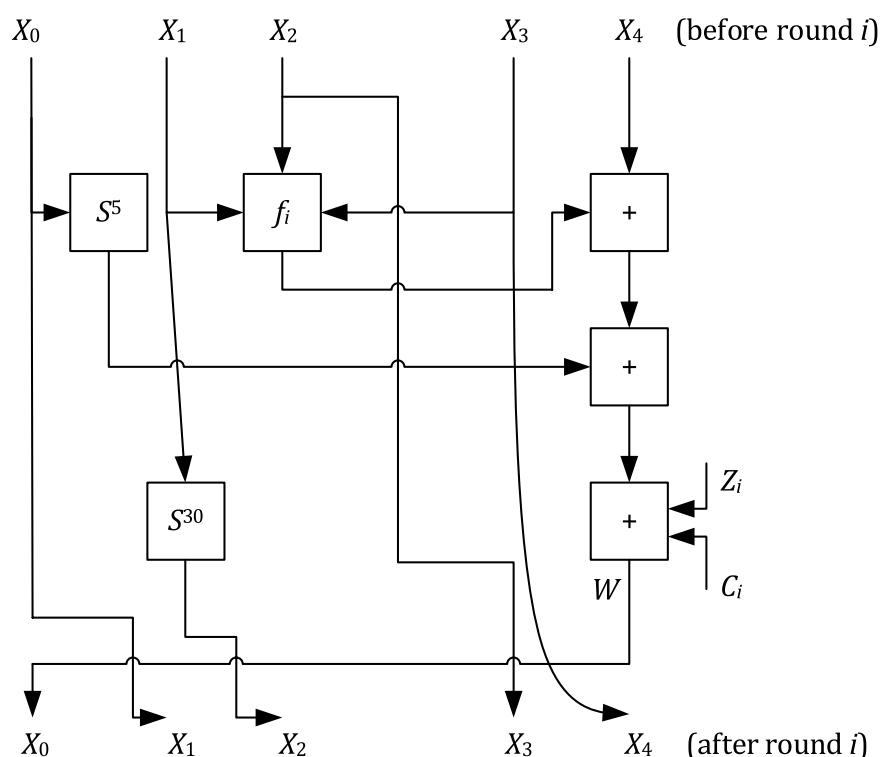


Figure 3 — Part of the round-function in Dedicated Hash-Function 3

10 Dedicated Hash-Function 4 (SHA-256)

10.1 General

In Clause 10, a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 4. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{64}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 4 is equal to 34 (hexadecimal).

NOTE Dedicated Hash-Function 4 defined in Clause 10 is commonly called SHA-256[1].

10.2 Parameters, functions and constants

10.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 256$ and L_H is up to 256.

10.2.2 Byte ordering convention

The byte ordering convention to be used with this hash-function shall be the same as the byte ordering convention defined in [9.2.2](#).

10.2.3 Functions

To facilitate software implementation, the round-function Φ is described in terms of operations on 32-bit words. A sequence of functions, e_0 , e_1 , e_2 , e_3 , e_4 and e_5 , is used in this round-function, where e_0 and e_1 each takes three words, X_0 , X_1 and X_2 , as input; e_2 , e_3 , e_4 and e_5 each takes one word X_0 as input and each of these six functions produces a single 32-bit word as output.

The functions e_0 , e_1 , e_2 , e_3 , e_4 and e_5 are defined as follows:

$$\begin{aligned} e_0(X_0, X_1, X_2) &= (X_0 \wedge X_1) \oplus (\neg X_0 \wedge X_2); \\ e_1(X_0, X_1, X_2) &= (X_0 \wedge X_1) \oplus (X_0 \wedge X_2) \oplus (X_1 \wedge X_2); \\ e_2(X_0) &= S'^{2}(X_0) \oplus S'^{13}(X_0) \oplus S'^{22}(X_0); \\ e_3(X_0) &= S'^{6}(X_0) \oplus S'^{11}(X_0) \oplus S'^{25}(X_0); \\ e_4(X_0) &= S'^{7}(X_0) \oplus S'^{18}(X_0) \oplus R^3(X_0); \\ e_5(X_0) &= S'^{17}(X_0) \oplus S'^{19}(X_0) \oplus R^{10}(X_0). \end{aligned}$$

10.2.4 Constants

A sequence of constant words, C_0 , C_1 , ..., C_{63} , is used in this round-function. In a hexadecimal representation (where the most significant bit corresponds to the left-most bit), these are 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.

10.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 256-bit string, represented here as a sequence of eight words, Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 32 of the 256 bits:

$$\begin{aligned} Y_0 &= 6A09E667; \\ Y_1 &= BB67AE85; \\ Y_2 &= 3C6EF372; \end{aligned}$$

$$\begin{aligned} Y_3 &= \text{A54FF53A}; \\ Y_4 &= \text{510E527F}; \\ Y_5 &= \text{9B05688C}; \\ Y_6 &= \text{1F83D9AB}; \\ Y_7 &= \text{5BE0CD19}. \end{aligned}$$

NOTE These values are obtained by taking the fractional parts of the square roots of the first eight primes.

10.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [9.3](#).

10.4 Description of the round-function

The round-function, Φ , operates as follows.

NOTE In this description, the symbols $W_1, W_2, X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, Z_0, Z_1, \dots, Z_{63}$ are used to denote 74 distinct words which contain values required in the computations.

- a) Suppose the 512-bit (first) input to Φ is contained in $Z_0, Z_1, \dots, Z_{15}$, where Z_0 contains the left-most 32 of the 512 bits. Suppose also that the 256-bit (second) input to Φ is contained in eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 .
- b) For $i = 16$ to 63 , let $Z_i = e_5(Z_{i-2}) \cup Z_{i-7} \cup e_4(Z_{i-15}) \cup Z_{i-16}$.
- c) Let $X_0 = Y_0, X_1 = Y_1, X_2 = Y_2, X_3 = Y_3, X_4 = Y_4, X_5 = Y_5, X_6 = Y_6$ and $X_7 = Y_7$.
- d) For $i = 0$ to 63 , do the following three steps:
 - 1) $W_1 = X_7 \cup e_3(X_4) \cup e_0(X_4, X_5, X_6) \cup C_i \cup Z_i$;
 - 2) $W_2 = e_2(X_0) \cup e_1(X_0, X_1, X_2)$;
 - 3) $X_7 = X_6; X_6 = X_5; X_5 = X_4; X_4 = X_3 \cup W_1; X_3 = X_2; X_2 = X_1; X_1 = X_0; X_0 = W_1 \cup W_2$.
- e) Let $Y_0 = Y_0 \cup X_0, Y_1 = Y_1 \cup X_1, Y_2 = Y_2 \cup X_2, Y_3 = Y_3 \cup X_3, Y_4 = Y_4 \cup X_4, Y_5 = Y_5 \cup X_5, Y_6 = Y_6 \cup X_6$ and $Y_7 = Y_7 \cup X_7$.
- f) The eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , represent the output of the round-function Φ . After the final iteration of the round-function, the eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , shall be converted to a sequence of 32 bytes using the inverse of the procedure specified in [10.2.2](#) and where Y_0 shall yield the first four bytes, Y_1 the next four bytes and so on. Thus, the first (left-most) byte will correspond to the most significant byte of Y_0 and the 32nd (right-most) byte will correspond to the least significant byte of Y_7 . The 32 bytes shall be converted to a string of 256 bits using the inverse of the procedure specified in [Clause 6](#), i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 256th (right-most) bit will correspond to the least significant bit of the 32nd (right-most) byte.

[Figure 4](#) shows steps 1), 2) and 3) of item d) of the round-function Φ in Dedicated Hash-Function 4 (SHA-256). In the round-function Φ , steps 1), 2) and 3) of item d) are used 64 times ($i = 0, \dots, 63$).

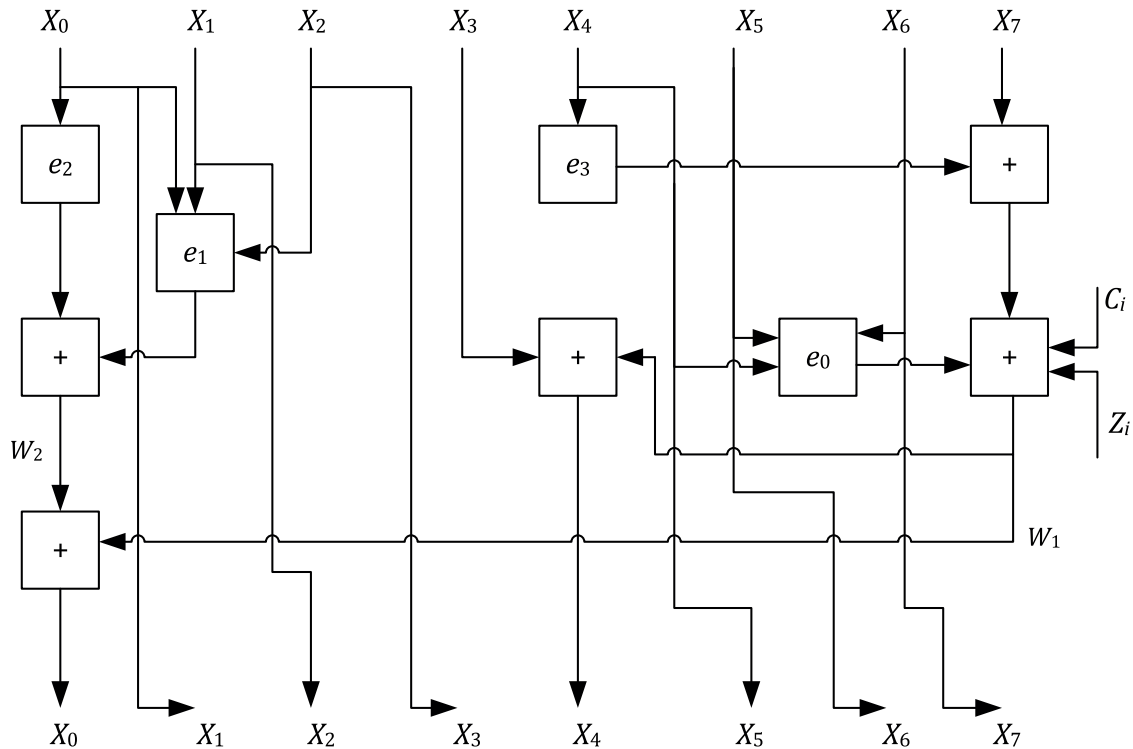


Figure 4 — Part of the round-function in Dedicated Hash-Function 4

11 Dedicated Hash-Function 5 (SHA-512)

11.1 General

In [Clause 11](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 5. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{128}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 5 is equal to 35 (hexadecimal).

NOTE Dedicated Hash-Function 5 defined in [Clause 11](#) is commonly called SHA-512[1].

11.2 Parameters, functions and constants

11.2.1 Parameters

For this hash-function, $L_1 = 1\,024$, $L_2 = 512$ and L_H is up to 512.

11.2.2 Byte ordering convention

In the specification of the round-function of [Clause 11](#), it is assumed that the block input to the round-function is in the form of a sequence of 64-bit words, each 1 024-bit block is made up of 16 such words. A sequence of 128 bytes, $B_0, B_1, \dots, B_{127}$, shall be interpreted as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, in the following way. Each group of eight consecutive bytes is considered a word, where the first byte of a word is the most significant byte of that word. Hence,

$$Z_i = 2^{56}B_{8i} + 2^{48}B_{8i+1} + 2^{40}B_{8i+2} + 2^{32}B_{8i+3} + 2^{24}B_{8i+4} + 2^{16}B_{8i+5} + 2^8B_{8i+6} + B_{8i+7}, \quad (0 \leq i \leq 15).$$

To convert the hash-code from a sequence of words to a sequence of bytes, the inverse process shall be followed.

11.2.3 Functions

To facilitate software implementation, the round-function Φ is described in terms of operations on 64-bit words. A sequence of functions, d_0, d_1, d_2, d_3, d_4 and d_5 , is used in this round-function, where d_0 and d_1 each takes three 64-bit words, X_0, X_1 and X_2 , as input; d_2, d_3, d_4 and d_5 each takes one 64-bit word X_0 as input and each of these six functions produces a single 64-bit word as output.

The functions d_0, d_1, d_2, d_3, d_4 and d_5 are defined as follows:

$$\begin{aligned} d_0(X_0, X_1, X_2) &= (X_0 \wedge X_1) \oplus (\neg X_0 \wedge X_2); \\ d_1(X_0, X_1, X_2) &= (X_0 \wedge X_1) \oplus (X_0 \wedge X_2) \oplus (X_1 \wedge X_2); \\ d_2(X_0) &= S'^{28}(X_0) \oplus S'^{34}(X_0) \oplus S'^{39}(X_0); \\ d_3(X_0) &= S'^{14}(X_0) \oplus S'^{18}(X_0) \oplus S'^{41}(X_0); \\ d_4(X_0) &= S'^1(X_0) \oplus S'^8(X_0) \oplus R^7(X_0); \\ d_5(X_0) &= S'^{19}(X_0) \oplus S'^{61}(X_0) \oplus R^6(X_0). \end{aligned}$$

11.2.4 Constants

A sequence of constant words, $C_0, C_1, \dots, C_{79}$, is used in this round-function. In a hexadecimal representation (where the most significant bit corresponds to the left-most bit), these are defined as follows, where the words are listed in the order $C_0, C_1, \dots, C_{79}$.

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	3C9EBE0A15C9BEBE	431D67C49C100D4C
4CC5D4BECB3E42B6	597F299CFC657E2A	5FCB6FAB3AD6FAEC	6C44198C4A475817

NOTE These values are the first 64 bits of the fractional parts of the cube roots of the first 80 primes.

11.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 512-bit string, represented here as a sequence of eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 64 of the 512 bits:

$Y_0 = 6A09E667F3BCC908;$
 $Y_1 = BB67AE8584CAA73B;$
 $Y_2 = 3C6EF372FE94F82B;$
 $Y_3 = A54FF53A5F1D36F1;$
 $Y_4 = 510E527FADE682D1;$
 $Y_5 = 9B05688C2B3E6C1F;$
 $Y_6 = 1F83D9ABFB41BD6B;$
 $Y_7 = 5BE0CD19137E2179.$

NOTE These values are obtained by taking the fractional parts of the square roots of the first eight primes.

11.3 Padding method

The data string, D , needs to be padded to make it contain a number of bits which is an integer multiple of 1 024. The padding procedure is as follows.

- D is concatenated with a single “1” bit.
- The result of the previous step is concatenated with between zero and 1 023 “0” bits, such that the length (in bits) of the resultant string is congruent to 896 modulo 1 024. More explicitly, if the original length of D is L_D , and letting r be the remainder when L_D is divided by 1 024, then the number of concatenated zeros is equal to either $895 - r$ (if $r \leq 895$) or $1\,919 - r$ (if $r > 895$). The result will be a bit string whose length will be 128 bits short of an integer multiple of 1 024 bits.
- Concatenate the string resulting from the previous step with the 128-bit binary representation of L_D , most significant bit first.

In the description of the round-function which follows, each 1 024-bit data block D_i , $1 \leq i \leq q$, is treated as a sequence of 16 words, $Z_0, Z_1, \dots, Z_{15}$, where Z_0 corresponds to the left-most 64 bits of D_i .

NOTE The concatenation of the 128-bit string of L_D in step c) is such that the most significant 64-bit string and the least significant 64-bit string of L_D are used respectively as the words Z_{14} and Z_{15} of the last data block. Based on the byte ordering convention in 11.2.2, the most significant byte of L_D is the left-most byte and the least significant byte of L_D is the right-most byte.

11.4 Description of the round-function

The round-function, Φ , operates as follows.

NOTE In this description, the symbols $W_1, W_2, X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, Z_0, Z_1, \dots, Z_{79}$ are used to denote 90 distinct words which contain values required in the computations.

- Suppose the 1 024-bit (first) input to Φ is contained in $Z_0, Z_1, \dots, Z_{15}$, where Z_0 contains the left-most 64 of the 1 024 bits. Suppose also that the 512-bit (second) input to Φ is contained in eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 .
- For $i = 16$ to 79, let $Z_i = d_5(Z_{i-2}) \cup Z_{i-7} \cup d_4(Z_{i-15}) \cup Z_{i-16}$.

- c) Let $X_0 := Y_0, X_1 := Y_1, X_2 := Y_2, X_3 := Y_3, X_4 := Y_4, X_5 := Y_5, X_6 := Y_6$ and $X_7 := Y_7$.
- d) For $i = 0$ to 79, do the following three steps:
- 1) $W_1 := X_7 \oplus d_3(X_4) \oplus d_0(X_4, X_5, X_6) \oplus C_i \oplus Z_i$;
 - 2) $W_2 := d_2(X_0) \oplus d_1(X_0, X_1, X_2)$;
 - 3) $X_7 := X_6; X_6 := X_5; X_5 := X_4; X_4 := X_3 \oplus W_1; X_3 := X_2; X_2 := X_1; X_1 := X_0; X_0 := W_1 \oplus W_2$.
- e) Let $Y_0 := Y_0 \oplus X_0, Y_1 := Y_1 \oplus X_1, Y_2 := Y_2 \oplus X_2, Y_3 := Y_3 \oplus X_3, Y_4 := Y_4 \oplus X_4, Y_5 := Y_5 \oplus X_5, Y_6 := Y_6 \oplus X_6$ and $Y_7 := Y_7 \oplus X_7$.
- f) The eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , represent the output of the round-function Φ . After the final iteration of the round-function, the eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , shall be converted to a sequence of 64 bytes using the inverse of the procedure specified in 11.2.2 and where Y_0 shall yield the first eight bytes, Y_1 the next eight bytes and so on. Thus, the first (left-most) byte will correspond to the most significant byte of Y_0 and the 64th (right-most) byte will correspond to the least significant byte of Y_7 . The 64 bytes shall be converted to a string of 512 bits using the inverse of the procedure specified in Clause 6, i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 512th (right-most) bit will correspond to the least significant bit of the 64th (right-most) byte.

Figure 5 shows steps 1), 2) and 3) of item d) of the round-function Φ in Dedicated Hash-Function 5 (SHA-512). In the round-function Φ , steps 1), 2) and 3) of item d) are used 80 times ($i = 0, \dots, 79$).

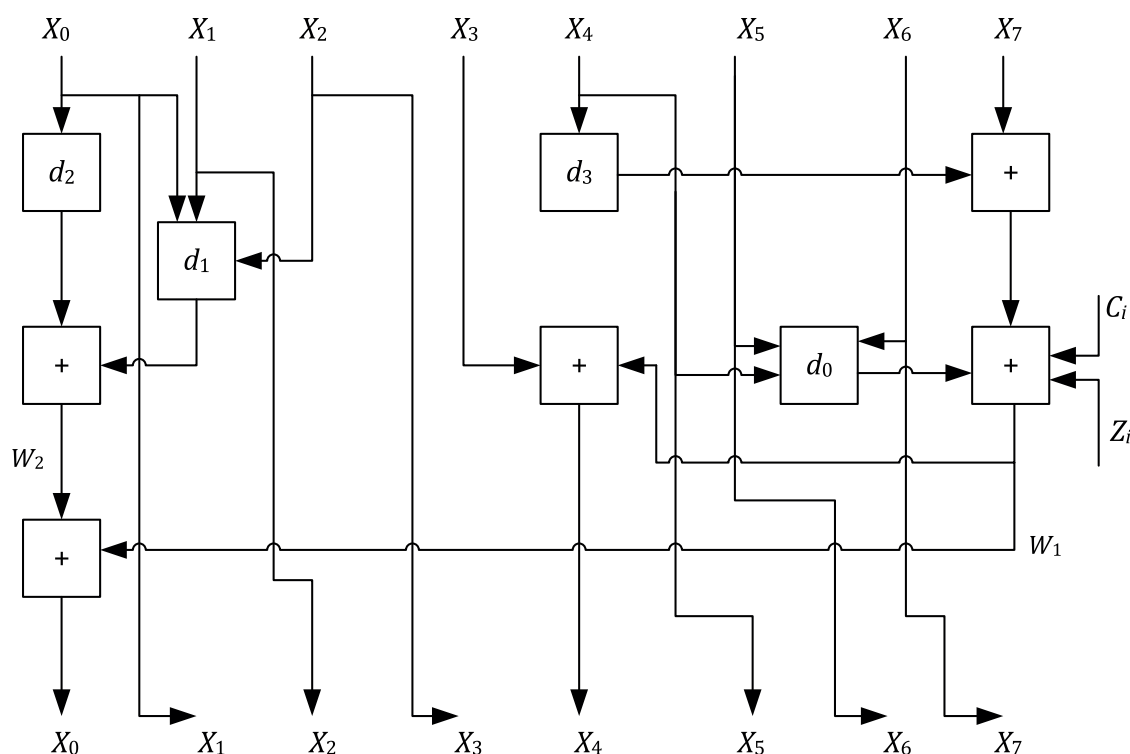


Figure 5 — Part of the round-function in Dedicated Hash-Function 5

12 Dedicated Hash-Function 6 (SHA-384)

12.1 General

In Clause 12, a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value

and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 6. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{128}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 6 is equal to 36 (hexadecimal).

NOTE Dedicated Hash-Function 6 defined in [Clause 12](#) is commonly called SHA-384[1].

12.2 Parameters, functions and constants

12.2.1 Parameters

For this hash-function, $L_1 = 1\,024$, $L_2 = 512$ and $L_H = 384$.

12.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of [Clause 11](#).

12.2.3 Functions

The functions for this hash-function are the same as that for the hash-function of [Clause 11](#).

12.2.4 Constants

The constants for this hash-function are the same as that for the hash-function of [Clause 11](#).

12.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 512-bit string, represented here as a sequence of eight words, Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 64 of the 512 bits:

```

 $Y_0 = \text{CBBB9D5DC1059ED8};$ 
 $Y_1 = \text{629A292A367CD507};$ 
 $Y_2 = \text{9159015A3070DD17};$ 
 $Y_3 = \text{152FECD8F70E5939};$ 
 $Y_4 = \text{67332667FFC00B31};$ 
 $Y_5 = \text{8EB44A8768581511};$ 
 $Y_6 = \text{DB0C2E0D64F98FA7};$ 
 $Y_7 = \text{47B5481DBEFA4FA4}.$ 

```

NOTE These values are obtained by taking the fractional parts of the square roots of the 9th to the 16th primes.

12.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [Clause 11](#).

12.4 Description of the round-function

The round-function to be used with this hash-function shall be the same as the round-function defined in [Clause 11](#).

The final 384-bit hash is obtained by truncating the SHA-512-based hash output to its left-most 384 bits.

13 Dedicated Hash-Function 7 (WHIRLPOOL)

13.1 General

In [Clause 13](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 7. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{256}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 7 is equal to 37 (hexadecimal).

NOTE Dedicated Hash-Function 7 defined in [Clause 13](#) is commonly called WHIRLPOOL[3].

13.2 Parameters, functions and constants

13.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 512$ and L_H is up to 512.

13.2.2 Byte ordering convention

In the specification of the round-function of [Clause 13](#), it is assumed that the block input to the round-function is in the form of a matrix M [where all matrices here are 8×8 matrices with entries chosen from $\text{GF}(2^8)$], each 512-bit block being made up of such a matrix. A sequence of 64 bytes, $B = (B_0, B_1, \dots, B_{63})$, shall be interpreted as a matrix M in the following way. The entry in the first row and the first column of the matrix shall be the left-most byte (where the left-most byte corresponds to the most significant byte) of the sequence B (i.e. B_0), the entry in the first row and the second column of the matrix shall be the second left-most byte of B (i.e. B_1), ..., and the entry in the eighth row and the eighth column of the matrix shall be the right-most byte of B (i.e. B_{63}). This is performed using function c_0 specified in [13.2.3](#).

To convert the hash-code from such a matrix to a sequence of bytes, the inverse process of the function c_0 shall be followed.

13.2.3 Functions

To facilitate software implementation, the round-function Φ is described in terms of operations on a matrix M . A sequence of functions, c_0 , c_1 , c_2 , c_3 and c_4 , is used in this round-function. They are defined as follows.

- a) Function c_0 takes a 64-byte sequence, $B = (B_0, B_1, \dots, B_{63})$ as input and produces a matrix $Z' = (z'_{ij})$ as output where:

$$z'_{ij} = B_{8i+j} \quad (0 \leq i, j \leq 7).$$

This means that $Z' = c_0(B)$, if and only if, $z'_{ij} = B_{8i+j}$ ($0 \leq i, j \leq 7$).

- b) Function c_1 takes a matrix $X'' = (x''_{ij})$ as input and produces another matrix $W' = (w'_{ij})$ as output where:

$$w'_{ij} = s[x''_{ij}], \quad (0 \leq i, j \leq 7),$$

and where s is a function defined below. This means $W' = c_1(X'')$, if and only if, $w'_{ij} = s[x''_{ij}]$ ($0 \leq i, j \leq 7$).

The function s replaces an element $x \in \text{GF}(2^8)$ with another element $s[x] \in \text{GF}(2^8)$. As specified in [Table 2](#), the elements in the first column are the “most significant half” of x and the elements in the first row are the “least significant half” of x . For instance, if $x = 01010110 = 56$ (hexadecimal), $s[x] = 49$ (hexadecimal) = 01001001.

Table 2 — Values of the s-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	18	23	C6	E8	87	B8	01	4F	36	A6	D2	F5	79	6F	91	52
1	60	BC	9B	8E	A3	0C	7B	35	1D	E0	D7	C2	2E	4B	FE	57
2	15	77	37	E5	9F	F0	4A	DA	58	C9	29	0A	B1	A0	6B	85
3	BD	5D	10	F4	CB	3E	05	67	E4	27	41	8B	A7	7D	95	D8
4	FB	EE	7C	66	DD	17	47	9E	CA	2D	BF	07	AD	5A	83	33
5	63	02	AA	71	C8	19	49	D9	F2	E3	5B	88	9A	26	32	B0
6	E9	0F	D5	80	BE	CD	34	48	FF	7A	90	5F	20	68	1A	AE
7	B4	54	93	22	64	F1	73	12	40	08	C3	EC	DB	A1	8D	3D
8	97	00	CF	2B	76	82	D6	1B	B5	AF	6A	50	45	F3	30	EF
9	3F	55	A2	EA	65	BA	2F	C0	DE	1C	FD	4D	92	75	06	8A
A	B2	E6	0E	1F	62	D4	A8	96	F9	C5	25	59	84	72	39	4C
B	5E	78	38	8C	D1	A5	E2	61	B3	21	9C	1E	43	C7	FC	04
C	51	99	6d	0D	FA	DF	7E	24	3B	AB	CE	11	8F	4E	B7	EB
D	3C	81	94	F7	B9	13	2C	D3	E7	6E	C4	03	56	44	7F	A9
E	2A	BB	C1	53	DC	0B	9D	6C	31	74	F6	46	AC	89	14	E1
F	16	3A	69	09	70	B6	D0	ED	CC	42	98	A4	28	5C	F8	86

- c) Function c_2 takes a matrix $X'' = (x''_{ij})$ as input and produces another matrix $W' = (w'_{ij})$ as output where:

$$w'_{ij} = x''_{(i-j) \bmod 8, j}, \quad (0 \leq i, j \leq 7).$$

This means that $W' = c_2(X'')$, if and only if, $w'_{ij} = x''_{(i-j) \bmod 8, j} \ (0 \leq i, j \leq 7)$.

- d) Function c_3 takes a matrix X'' as input and produces another matrix W' as output where:

$$W' = X'' \cdot C'',$$

and where C'' is an 8×8 circulant matrix with entries chosen from $\text{GF}(2^8)$, as specified below:

$$C'' = \begin{bmatrix} 01 & 01 & 04 & 01 & 08 & 05 & 02 & 09 \\ 09 & 01 & 01 & 04 & 01 & 08 & 05 & 02 \\ 02 & 09 & 01 & 01 & 04 & 01 & 08 & 05 \\ 05 & 02 & 09 & 01 & 01 & 04 & 01 & 08 \\ 08 & 05 & 02 & 09 & 01 & 01 & 04 & 01 \\ 01 & 08 & 05 & 02 & 09 & 01 & 01 & 04 \\ 04 & 01 & 08 & 05 & 02 & 09 & 01 & 01 \\ 01 & 04 & 01 & 08 & 05 & 02 & 09 & 01 \end{bmatrix}$$

This means that $W' = c_3(X'')$ if and only if $W' = X'' \cdot C''$.

- e) Function c_4 takes two matrices $X'' = (x''_{ij})$ and $Y' = (y'_{ij})$ as input and produces a single matrix $W' = (w'_{ij})$ as output where:

$$w'_{ij} = x''_{ij} \oplus y'_{ij}, \quad (0 \leq i, j \leq 7).$$

This means that $W' = c_4(X'', Y')$, if and only if, $w'_{ij} = x''_{ij} \oplus y'_{ij} \ (0 \leq i, j \leq 7)$.

13.2.4 Constants

A sequence of constant matrices, $A^r = (A_{ij}^r)$ ($0 < r \leq 10$), is used in this round-function. The round constant for the r th round is a matrix, defined as:

$$\begin{aligned} A_{0j}^r &= s[8(r-1) + j], & (0 \leq j \leq 7), \\ A_{ij}^r &= 0, & (1 \leq i \leq 7, 0 \leq j \leq 7). \end{aligned}$$

13.2.5 Initializing value

The initializing value, IV , is a string of 512 “0” bits.

NOTE 512 “0” bits for the initial value is represented by a matrix Y' with entries in $\text{GF}(2^8)$.

13.3 Padding method

The data string, D , needs to be padded to make it contain a number of bits which is an integer multiple of 512. The padding procedure is as follows.

- D is concatenated with a single “1” bit.
- The result of the previous step is concatenated with between zero and 511 “0” bits, such that the length (in bits) of the resultant string is an odd multiple of 256.
- If the original length of D is L_D , concatenate the string resulting from the previous step with the 256-bit binary representation of L_D , most significant bit first.

In the description of the round-function which follows, each 512-bit data block D_i , $1 \leq i \leq q$, is treated as a matrix $Z' = (z'_{ij})$ ($0 \leq i, j \leq 7$), as specified in 13.2.3, where z'_{00} corresponds to the left-most 8 bits of D_i and z'_{77} corresponds to the right-most 8 bits of D_i .

NOTE The concatenation of the 256-bit string of L_D in step c) is such that the 256-bit string is used directly as the second half of the last data matrix. Based on the byte ordering convention in 13.2.2, the most significant byte of L_D is the entry in the fifth row and the first column and the least significant byte of L_D is the entry in the eighth row and the eighth column.

13.4 Description of the round-function

The round-function Φ operates as follows.

NOTE In this description, the symbols $W', X'', K_0, K_1, \dots, K_{10}$ are used to denote 13 distinct matrices, each with entries chosen from $\text{GF}(2^8)$, which contain values required in the computations.

- Suppose the 512-bit (first) input to Φ is contained in a matrix Z' with entries chosen from $\text{GF}(2^8)$ which is formed by using the byte ordering convention specified in 13.2.2. Suppose also that the 512-bit (second) input to Φ is contained in a matrix Y' with entries chosen from $\text{GF}(2^8)$.
- Let $K_0 = Y'$ and for $i = 1$ to 10, let $K_i = c_4(c_3(c_2(c_1(K_{i-1}))), A^i)$.
NOTE This step expands the matrix Y' onto a sequence of round keys $K_0, \dots, K_{10}$.
- Let $X'' = c_4(Z', K_0)$ and for $j = 1$ to 10, do the following two steps:
 - $W' = c_4(c_3(c_2(c_1(X''))), K_j)$;
 - $X'' = W'$.
- Let $Y' = W' \oplus K_0 \oplus Z'$.

The matrix Y' represents the output of the round-function Φ . After the final iteration of the round-function, the matrix Y' shall be converted to a sequence of 64 bytes using the inverse of the procedure

specified in 16.2.2 and where the entry in the first row and the first column of the matrix shall yield the first byte, the entry in the first row and the second column of the matrix the next byte, ..., the entry in the eighth row and the eighth column of the matrix the last byte. The 64 bytes shall be converted to a string of 512 bits using the inverse of the procedure specified in Clause 6, i.e. the first (left-most) bit will correspond to the most significant bit of the first (left-most) byte and the 512th (right-most) bit will correspond to the least significant bit of the 64th (right-most) byte.

Figure 6 shows steps 1) and 2) of item c) of the round-function Φ in Dedicated Hash-Function 7 (WHIRLPOOL). In the round-function Φ , the steps shown in Figure 6 are used 10 times ($j = 1, \dots, 10$).

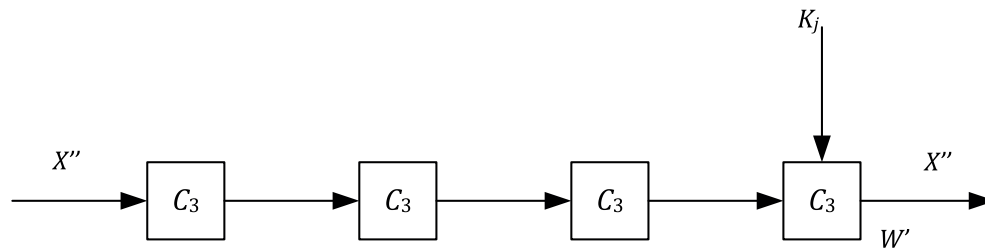


Figure 6 — Part of the round-function in Dedicated Hash-Function 7

14 Dedicated Hash-Function 8 (SHA-224)

14.1 General

In Clause 14, a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 8. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{64}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 8 is equal to 38 (hexadecimal).

NOTE Dedicated Hash-Function 8 defined in Clause 14 is commonly called SHA-224[1].

14.2 Parameters, functions and constants

14.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 256$ and $L_H = 224$.

14.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of Clause 10.

14.2.3 Functions

The functions for this hash-function are the same as those for the hash-function of Clause 10.

14.2.4 Constants

The constants for this hash-function are the same as those for the hash-function of Clause 10.

14.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 256-bit string, represented here as a sequence of eight words, $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 32 of the 256 bits:

$Y_0 = \text{C1059ED8};$
 $Y_1 = \text{367CD507};$
 $Y_2 = \text{3070DD17};$
 $Y_3 = \text{F70E5939};$
 $Y_4 = \text{FFC00B31};$
 $Y_5 = \text{68581511};$
 $Y_6 = \text{64F98FA7};$
 $Y_7 = \text{BEFA4FA4}.$

NOTE These values are the low order 32 bits of the values specified in [12.2.5](#).

14.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [10.3](#).

14.4 Description of the round-function

The round-function to be used with this hash-function shall be the same as the round-function defined in [10.4](#).

The final 224-bit hash is obtained by truncating the SHA-256-based hash output to its left-most 224 bits.

15 Dedicated Hash-Function 9 (SHA-512/224)

15.1 General

In [Clause 15](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 9. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{128}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 9 is equal to 39 (hexadecimal).

NOTE Dedicated Hash-Function 9 defined in [Clause 15](#) is commonly called SHA-512/224[1].

15.2 Parameters, functions and constants

15.2.1 Parameters

For this hash-function, $L_1 = 1\,024$, $L_2 = 512$ and L_H is up to 224.

15.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of [Clause 11](#).

15.2.3 Functions

The functions for this hash-function are the same as that for the hash-function of [Clause 11](#).

15.2.4 Constants

The constants for this hash-function are the same as that for the hash-function of [Clause 11](#).

15.2.5 Initializing value

For this round-function, the initializing value, IV , shall always be the following 512-bit string, represented here as a sequence of eight words, Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 64 of the 512 bits:

```

 $Y_0 = 8C3D37C819544DA2;$ 
 $Y_1 = 73E1996689DCD4D6;$ 
 $Y_2 = 1DFAB7AE32FF9C82;$ 
 $Y_3 = 679DD514582F9FCF;$ 
 $Y_4 = 0F6D2B697BD44DA8;$ 
 $Y_5 = 77E36F7304C48942;$ 
 $Y_6 = 3F9D85A86A1D36C8;$ 
 $Y_7 = 1112E6AD91D692A1.$ 

```

15.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [Clause 11](#).

15.4 Description of the round-function

The round-function to be used with this hash-function shall be the same as the round-function defined in [Clause 11](#). The final 224-bit hash is obtained by truncating the SHA-512-based hash output to its left-most 224 bits.

16 Dedicated Hash-Function 10 (SHA-512/256)

16.1 General

In [Clause 16](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 10. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{128}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 10 is equal to 3A (hexadecimal).

NOTE Dedicated Hash-Function 10 defined in [Clause 16](#) is commonly called SHA-512/256^[1].

16.2 Parameters, functions and constants

16.2.1 Parameters

For this hash-function, $L_1 = 1\,024$, $L_2 = 512$ and L_H is up to 256.

16.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of [Clause 11](#).

16.2.3 Functions

The functions for this hash-function are the same as that for the hash-function of [Clause 11](#).

16.2.4 Constants

The constants for this hash-function are the same as that for the hash-function of [Clause 11](#).

16.2.5 Initializing value

For this round-function the initializing value, IV , shall always be the following 512-bit string, represented here as a sequence of eight words, Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 and Y_7 , in a hexadecimal representation, where Y_0 represents the left-most 64 of the 512 bits:

```
Y0 = 22312194FC2BF72C;  
Y1 = 9F555FA3C84C64C2;  
Y2 = 2393B86B6F53B151;  
Y3 = 963877195940EABD;  
Y4 = 96283EE2A88EFFE3;  
Y5 = BE5E1E2553863992;  
Y6 = 2B0199FC2C85B8AA;  
Y7 = 0EB72DDC81C52CA2.
```

16.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [Clause 11](#).

16.4 Description of the round-function

The round-function to be used with this hash-function shall be the same as the round-function defined in [Clause 11](#). The final 256-bit hash is obtained by truncating the SHA-512-based hash output to its left-most 256 bits.

17 Dedicated Hash-Function 11 (STREEBOG-512)

17.1 General

In [Clause 17](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define Dedicated Hash-Function 11. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{512}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 11 is equal to 3B (hexadecimal).

NOTE Dedicated Hash-Function 11 defined in [Clause 17](#) is one of the functions specified in GOST R 34.11-2012, the national standard of the Russian Federation, commonly called STREEBOG[2].

17.2 Parameters, functions and constants

17.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 512$ and $L_H = 512$.

17.2.2 Byte ordering convention

In the specification of the round-function of [Clause 17](#), it is assumed that the block input to the round-function is in the form of a sequence of 512 bits. A sequence of 64 bytes, $B_0, \dots, B_{63}$, shall be interpreted as a sequence of consecutive bits in the following way. Each byte is considered as an 8-bit sequence, the most significant bit of the byte shall be the first bit of the sequence. Each 512-bit block is treated as a number in the following format:

$$Z = B_0 + B_1 2^8 + \dots + B_{63} 2^{504}.$$

To convert the hash-code from a sequence of bits to a sequence of bytes, the inverse process shall be followed.

17.2.3 Functions

17.2.3.1 General

To calculate the hash-code H of the data string D , functions X, S, P, L and MSB_n are used. They are defined in [17.2.3.2](#) to [17.2.3.6](#).

17.2.3.2 Function X

Function $X[k]$ takes a 512-bit word as input and for the given 512-bit word, produces a 512-bit word as output, where $X[k](a) = k \oplus a$.

17.2.3.3 Function S

Function S takes a 512-bit word as input, produces a 512-bit word as output and is defined as:

$$S(a) = S(a_{63} \parallel \dots \parallel a_0) = \pi(a_{63}) \parallel \dots \parallel \pi(a_0),$$

where $a = a_{63} \parallel \dots \parallel a_0$, a_i , $i = 0, \dots, 63$, are 8-bit words and π denotes a function from the set of octet strings to itself. π is defined as:

$$\pi = \text{Vec}_8 \pi' \text{Int}_8,$$

where $\pi': \mathbb{Z}_{28} \rightarrow \mathbb{Z}_{28}$.

The function π' is defined by the array $\pi' = [\pi'(0), \pi'(1), \dots, \pi'(255)]$:

$\pi' = (252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4, 77, 233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127, 212, 211, 31, 235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253, 58, 206, 204, 181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183, 93, 135, 21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177, 50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13, 87, 223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78, 51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65, 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217, 231, 137, 225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202, 216, 133, 97, 32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190, 229, 108, 82, 89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75, 99, 182).$

17.2.3.4 Function P

Function P takes a 512-bit word as input, produces a 512-bit word as output and is defined as:

$$P(a) = P(a_{63} \parallel \dots \parallel a_0) = a_{\tau(63)} \parallel \dots \parallel a_{\tau(0)},$$

where $a = a_{63} \parallel \dots \parallel a_0$, a_i , $i = 0, \dots, 63$, are 8-bit words and τ is a permutation of the set $\{0, 1, \dots, 63\}$ given by the array $\tau = (\tau(0), \tau(1), \dots, \tau(63))$:

$\tau = (0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25, 33, 41, 49, 57, 2, 10, 18, 26, 34, 42, 50, 58, 3, 11, 19, 27, 35, 43, 51, 59, 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, 21, 29, 37, 45, 53, 61, 6, 14, 22, 30, 38, 46, 54, 62, 7, 15, 23, 31, 39, 47, 55, 63)$.

17.2.3.5 Function L

Function L takes a 512-bit word as input, produces a 512-bit word as output and is defined as:

$$L(a) = L(a_7 \parallel \dots \parallel a_0) = l(a_7) \parallel \dots \parallel l(a_0),$$

where $a = a_7 \parallel \dots \parallel a_0$, a_i , $i = 0, \dots, 7$, are 64-bit words and l is a function equal to right multiplication by the matrix A , given below, over the field $\text{GF}(2)$. The matrix rows are expressed sequentially in hexadecimal notation. The row with number j , $j = 0, \dots, 63$, (specified in the form $a_{j,15}, \dots, a_{j,0}$, where $a_{j,i} \in \mathbb{F}_{16}$, $i = 0, \dots, 15$), is $\text{Vec}_4(a_{j,15}) \parallel \dots \parallel \text{Vec}_4(a_{j,0})$.

8E20FAA72BA0B470	47107DDD9B505A38	AD08B0E0C3282D1C	D8045870EF14980E
6C022C38F90A4C07	3601161CF205268D	1B8E0B0E798C13C8	83478B07B2468764
A011D380818E8F40	5086E740CE47C920	2843FD2067ADEA10	14AFF010BDD87508
0AD97808D06CB404	05E23C0468365A02	8C711E02341B2D01	46B60F011A83988E
90DAB52A387AE76F	486DD4151C3DFDB9	24B86A840E90F0D2	125C354207487869
092E94218D243CBA	8A174A9EC8121E5D	4585254F64090FA0	ACCC9CA9328A8950
9D4DF05D5F661451	C0A878A0A1330AA6	60543C50DE970553	302A1E286FC58CA7
18150F14B9EC46DD	0C84890AD27623E0	0642CA05693B9F70	0321658CBA93C138
86275DF09CE8AAA8	439DA0784E745554	AFC0503C273AA42A	D960281E9D1D5215
E230140FC0802984	71180A8960409A42	B60C05CA30204D21	5B068C651810A89E
456C34887A3805B9	AC361A443D1C8CD2	561B0D22900E4669	2B838811480723BA
9BCF4486248D9F5D	C3E9224312C8C1A0	EFFA11AF0964EE50	F97D86D98A327728
E4FA2054A80B329C	727D102A548B194E	39B008152ACB8227	9258048415EB419D
492C024284FBAEC0	AA16012142F35760	550B8E9E21F7A530	A48B474F9EF5DC18
70A6A56E2440598E	3853DC371220A247	1CA76E95091051AD	0EDD37C48A08A6D8
07E095624504536C	8D70C431AC02A736	C83862965601DD1B	641C314B2B8EE083

Each row of the given table contains four rows of the matrix A . So, the line with number i , $i = 0, \dots, 15$, specifies the rows of the matrix A with numbers $4i + j$, $j = 0, \dots, 3$, in the following left-to-right order: $4i + 0, 4i + 1, 4i + 2, 4i + 3$.

The product of the 64-bit word $b = b_{63}, \dots, b_0$ and the matrix A is a 64-bit word c :

$$c = b_{63}(\text{Vec}_4(a_{0,15}) \parallel \dots \parallel \text{Vec}_4(a_{0,0})) \oplus \dots \oplus b_0(\text{Vec}_4(a_{63,15}) \parallel \dots \parallel \text{Vec}_4(a_{63,0})),$$

where

$$b_i (\text{Vec}_4(a_{63-i,15}) \parallel \dots \parallel \text{Vec}_4(a_{63-i,0})) = \begin{cases} 0^{64}, & \text{if } b_i = 0, \\ (\text{Vec}_4(a_{63-i,15}) \parallel \dots \parallel \text{Vec}_4(a_{63-i,0})), & \text{if } b_i = 1, \end{cases}$$

for all $i = 0, \dots, 63$.

17.2.3.6 Truncation function

Function MSB_n maps the word $z_{k-1} \parallel \dots \parallel z_1 \parallel z_0$, $k \geq n$ to the word $z_{k-1} \parallel \dots \parallel z_{k-n+1} \parallel z_{k-n}$.

17.2.4 Constants

Round constants are expressed in hexadecimal notation. The constant value specified in the form $a_{127}, \dots, a_0$ (where $a_i \in \mathbb{F}_{16}$, $i = 0, \dots, 127$) is $\text{Vec}_4(a_{127}) \parallel \dots \parallel \text{Vec}_4(a_0)$:

C_1 = B1085BDA1ECADAE9EBCB2F81C0657C1F2F6A76432E45D016714EB88D7585C4FC
 4B7CE09192676901A2422A08A460D31505767436CC744D23DD806559F2A64507;

C_2 = 6FA3B58AA99D2F1A4FE39D460F70B5D7F3FEEA720A232B9861D55E0F16B50131
 9AB5176B12D699585CB561C2DB0AA7CA55DDA21BD7CBCD56E679047021B19BB7;

C_3 = F574DCAC2BCE2FC70A39FC286A3D843506F15E5F529C1F8BF2EA7514B1297B7B
 D3E20FE490359EB1C1C93A376062DB09C2B6F443867ADB31991E96F50ABA0AB2;

C_4 = EF1FDFB3E81566D2F948E1A05D71E4DD488E857E335C3C7D9D721CAD685E353F
 A9D72C82ED03D675D8B71333935203BE3453EAA193E837F1220CBEB84E3D12E;

C_5 = 4BEA6BACAD4747999A3F410C6CA923637F151C1F1686104A359E35D7800FFFBD
 BFCD1747253AF5A3DFFF00B723271A167A56A27EA9EA63F5601758FD7C6CFE57;

C_6 = AE4FAEAE1D3AD3D96FA4C33B7A3039C02D66C4F95142A46C187F9AB49AF08EC6
 CFFAA6B71C9AB7B40AF21F66C2BEC6B6BF71C57236904F35FA68407A46647D6E;

C_7 = F4C70E16EEAAC5EC51AC86FEBF240954399EC6C7E6BF87C9D3473E33197A93C9
 0992ABC52D822C3706476983284A05043517454CA23C4AF38886564D3A14D493;

C_8 = 9B1F5B424D93C9A703E7AA020C6E41414EB7F8719C36DE1E89B4443B4DDBC49A
 F4892BCB929B069069D18D2BD1A5C42F36ACC2355951A8D9A47F0DD4BF02E71E;

C_9 = 378F5A541631229B944C9AD8EC165FDE3A7D3A1B258942243CD955B7E00D0984
 800A440BDBB2CEB17B2B8A9AA6079C540E38DC92CB1F2A607261445183235ADB;

C_{10} = ABBEDEA680056F52382AE548B2E4F3F38941E71CFF8A78DB1FFFE18A1B336103
 9FE76702AF69334B7A1E6C303B7652F43698FAD1153BB6C374B4C7FB98459CED;

C_{11} = 7BCD9ED0EFC889FB3002C6CD635AFE94D8FA6BBBEBAB07612001802114846679
 8A1D71EFEA48B9CAEFBACD1D7D476E98DEA2594AC06FD85D6BCAA4CD81F32D1B;

C_{12} = 378EE767F11631BAD21380B00449B17ACDA43C32BCDF1D77F82012D430219F9B
 5D80EF9D1891CC86E71DA4AA88E12852FAF417D5D9B21B9948BC924AF11BD720.

17.2.5 Initializing value

The initializing value, IV , is equal to 0^{512} .

17.3 Padding method

The data string, D , needs to be padded to make it contain a number of bits which is an integer multiple of 512. The padding procedure is as follows:

- D is concatenated with a single “1” bit (bit placed to the left).
- The result of the previous step is concatenated with between zero and 511 “0” bits (placed to the left) such that the length (in bits) of the resultant string is a multiple of 512.
- If the original length of D is L_D , concatenate the string resulting from the previous step with the 512-bit binary representation of L_D .
- If the data after step b) of padding could be expressed in the form $D_0, D_1, \dots, D_k$, where D_i are 512-bit words and k is a positive integer, calculate the value $\Sigma = D_0 \oplus D_1 \oplus \dots \oplus D_k$ and concatenate the string resulting from the previous step with value Σ .

17.4 Description of the round-function

The round-function Φ operates as follows. Note that in [Clause 17](#), the symbols h , m and N are used to denote three distinct 512-bit words which contain values required in the computations. The hash-code value of the data string D is calculated using an iterative procedure. Each iteration is performed using a round-function that transforms two 512-bit words to a 512-bit word and is calculated as:

$$\Phi(m, h) = g_N(h, m) = E(LPS(h \oplus N), m) \oplus h \oplus m,$$

where $E(K, m) = X[K_{13}]LPSX[K_{12}]...LPSX[K_2]LPSX[K_1](m)$ and where N denotes a 512-bit word calculated during the iterative procedure.

The values $K_i \in V_{512}$, $i = 1, \dots, 13$ are calculated as follows:

$$K_1 = K;$$

$$K_i = LPS(K_{i-1} \oplus C_{i-1}), i = 2, \dots, 13.$$

For brevity, instead of $g_{0^{512}}$, the notation g_0 is used.

The hash-function operates as follows. The input for calculating the hash-code is the data string, D (to be hashed), and the initializing value, IV .

The padding method described in [17.3](#) is accomplished within the algorithm that follows, i.e. that the padding does not actually have to be done prior to running the algorithm.

The algorithm for calculating the hash-code consists of the following stages.

a) Stage 1

Assign initial values to the following variables:

- 1) $h := IV$;
- 2) $N := 0^{512}$;
- 3) $\Sigma := 0^{512}$;
- 4) Go to Stage 2.

b) Stage 2

- 1) Check the condition: $L_D < 512$.

If it is true, then go to Stage 3.

Else, perform the following calculations:

- i) Let m be the right-most 512 bits of the message D (so that $D = D' \parallel m$). Then perform the following calculations:
 - $h := g_N(h, m)$;
 - $N := \text{Vec}_{512}(\text{Int}_{512}(N) \cup 512)$;
 - $\Sigma := \text{Vec}_{512}(\text{Int}_{512}(\Sigma) \cup \text{Int}_{512}(m))$;
 - $D := D'$.
- ii) Go to b)1).

c) Stage 3

- 1) $m := 0^{511-L_D} \parallel 1 \parallel D$.

- 2) $h := g_N(h, m)$.
- 3) $N := \text{Vec}_{512}(\text{Int}_{512}(N) \cup L_D)$.
- 4) $\Sigma := \text{Vec}_{512}(\text{Int}_{512}(\Sigma) \cup \text{Int}_{512}(m))$.
- 5) $h := g_0(h, N)$.
- 6) $h := g_0(h, \Sigma)$.
- 7) End of the algorithm.

The value of the variable h [obtained in step 6)] is the hash-code H .

18 Dedicated Hash-Function 12 (STREEBOG-256)

18.1 General

In [Clause 18](#), a padding method, an initializing value and a round-function for use in the general model for hash-functions described in ISO/IEC 10118-1 are specified. The padding method, initializing value and round-function specified here, when used in the above general model, together define the Dedicated Hash-Function 12. This dedicated hash-function can be applied to all data strings, D , containing at most $2^{512}-1$ bits.

The ISO/IEC hash-function identifier for Dedicated Hash-Function 12 is equal to 3C (hexadecimal).

NOTE Dedicated Hash-Function 12 defined in [Clause 18](#) is one of the functions specified in GOST R 34.11-2012, the national standard of the Russian Federation, commonly called STREEBOG[2].

18.2 Parameters, functions and constants

18.2.1 Parameters

For this hash-function, $L_1 = 512$, $L_2 = 512$ and $L_H = 256$.

18.2.2 Byte ordering convention

The byte ordering convention for this hash-function is the same as that for the hash-function of [Clause 17](#).

18.2.3 Functions

The functions for this hash-function are the same as those for the hash-function of [Clause 17](#).

18.2.4 Constants

The constants for this hash-function are the same as those for the hash-function of [Clause 17](#).

18.2.5 Initializing value

The initializing value, IV , equals $(000\ 000\ 01)^{64}$.

18.3 Padding method

The padding method to be used with this hash-function shall be the same as the padding method defined in [Clause 17](#).

18.4 Description of the round-function

The round-function to be used with this hash-function shall be the same as the round-function defined in [Clause 17](#). The final 256-bit hash is obtained by truncating the STREEBOG-512-based hash output to its most significant 256 bits.

19 Dedicated Hash-Function 13 (SHA3-224)

19.1 General

In [Clause 19](#), a permutation-based hash-function with sponge construction SHA-224 is specified. The ISO/IEC hash-function identifier for Dedicated Hash-Function 13 is equal to 3D (hexadecimal).

19.2 Parameters, functions and constants

19.2.1 Parameters

For this hash-function, $L_1 = r = 1\,152$, $L_2 = b = 1\,600$, $c = b - r = 448$, $d = 224$, L_H is up to 224.

19.2.2 Byte ordering convention

Each data input D to the round-function Φ is a block of 1 152 bits that is XORed into the part of the state. The permutation f is then applied to the state. Because the step mappings that comprise the permutation are defined on the array form of the state, it is convenient to regard D as a sequence of 64-bit words that are XORed directly into the state array. For this purpose, when D is represented as a sequence of 144 bytes, $B_0, B_1, \dots, B_{143}$, then D shall be interpreted as a sequence of 18 lane words, $Z_0, Z_1, \dots, Z_{17}$, as follows:

$$Z_i = 2^{56}B_{8i+7} + 2^{48}B_{8i+6} + 2^{40}B_{8i+5} + 2^{32}B_{8i+4} + 2^{24}B_{8i+3} + 2^{16}B_{8i+2} + 2^8B_{8i+1} + B_{8i},$$

for $0 \leq i \leq 17$.

Hence, each group of eight consecutive bytes is a word and the bytes of the word are arranged in increasing order of significance, so that the first byte in the group becomes the least significant byte of the word.

For dedicated hash-function 13, the function is defined on a $5 \times 5 \times w$ state array. For each array six sub-arrays are defined for step-mappings. Lane is one of the subarrays and defined in [19.2.3.3](#). Under this interpretation, D is XORed with the state array as follows:

If j and k are the elements of $\{0, 1, 2, 3, 4\}$ such that (j, k) is the unique pair for which $i = 5k + j$, then for $0 \leq i \leq 17$, $\text{Lane}'(j, k) = Z_i \oplus \text{Lane}(j, k)$, where $\text{Lane}'(j, k)$ is the updated value of the lane.

19.2.3 Functions

19.2.3.1 General

In [19.2.3.8](#), the KECCAK- p permutations are specified. A KECCAK- p permutation is determined by two parameters. First, the fixed length of the strings that are permuted, called the *width* of the permutation, is denoted as b . Second, the number of iterations of an internal transformation, called a *round*, is denoted as n_r . For the Dedicated Hash-Functions 13, 14, 15 and 16, $b = 1\,600$ and $n_r = 24$. However, in some of the examples, in particular when a figure is used to illustrate the operations, smaller values of b are used.

A round of a KECCAK- p permutation, denoted by Rnd , consists of a sequence of five transformations called *step mappings*. The permutation is specified in terms of an array of values for b ($= 1\,600$) bits that is repeatedly updated, called the *state*. The state is initially set to the input values of the permutation.

NOTE The term round is used differently from the term specified in ISO/IEC 10118-1, where a round is a function used to process one single input data block to the hash-function. For the Dedicated Hash-Functions 13, 14, 15 and 16, to process one single data block, the round-function is iterated n_r ($= 24$) times. That is, each execution of round-function Φ , as it is named in ISO/IEC 10118-1, iterates Rnd 24 times.

The notation and terminology for the state are described in [19.2.3.2](#) to [19.2.3.6](#). The step mappings are specified in [19.2.3.7](#). The KECCAK- p permutations, including the round-function Rnd , are specified in [19.2.3.8](#).

19.2.3.2 State

The input and output states of the permutation are comprised of b bit strings. To represent the step mappings, a state is represented as a $5 \times 5 \times w$ array of bits, where $w = b/25$. For $b = 1\,600$, $w = 64$. If S denotes a string that represents the state, then its bits are indexed from 0 to $b-1$, so that

$$S = S[0] \parallel S[1] \parallel \dots \parallel S[b-2] \parallel S[b-1].$$

If $\mathbf{A}$ denotes a $5 \times 5 \times w$ array of bits that represents the state, then its indices are the integer triples (x, y, z) for which $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$. The bit that corresponds to (x, y, z) is denoted by $\mathbf{A}[x, y, z]$. A *state array* is a representation of the state by a three-dimensional array that is indexed in this manner.

19.2.3.3 Parts of the state array

The two-dimensional sub-arrays are called sheets, planes and slices; and the single-dimensional sub-arrays are called rows, columns and lanes.

The algebraic definitions of these sub-arrays are as follows.

Column For a state array, a sub-array of 5 bits with constant x and z coordinates.

Lane For a state array of a KECCAK- p permutation with width b , a sub-array of $b/25$ bits with constant x and y coordinates.

Plane For a state array of a KECCAK- p permutation with width b , a sub-array of $b/5$ bits with constant y coordinate.

Row For a state array, a sub-array of 5 bits with constant y and z coordinates.

Sheet For a state array of a KECCAK- p permutation with width b , a sub-array of $b/5$ bits with a constant x coordinate.

Slice For a state array, a sub-array of 25 bits with a constant z coordinate.

19.2.3.4 Converting strings to state arrays

Let S denote a string of b bits that represents the state for the KECCAK- p permutation. The corresponding state array, denoted by $\mathbf{A}$, is defined as follows.

For all triples (x, y, z) , such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$,

$$\mathbf{A}[x, y, z] = S[w(5y + x) + z].$$

For $b = 1\,600$ and $w = 64$,

$A[0, 0, 0] = S [0]$	$A[1, 0, 0] = S [64]$	$A[4, 0, 0] = S [256]$
$A[0, 0, 1] = S [1]$	$A[1, 0, 1] = S [65]$	$A[4, 0, 1] = S [257]$
$A[0, 0, 2] = S [2]$	$A[1, 0, 2] = S [66]$	$A[4, 0, 2] = S [258]$
$\vdots$	$\vdots$	$\vdots$
$A[0, 0, 61] = S [61]$	$A[1, 0, 61] = S [125]$	$A[4, 0, 61] = S [317]$
$A[0, 0, 62] = S [62]$	$A[1, 0, 62] = S [126]$	$A[4, 0, 62] = S [318]$
$A[0, 0, 63] = S [63]$	$A[1, 0, 63] = S [127]$	$A[4, 0, 63] = S [319]$

and

$A[0, 1, 0] = S [320]$	$A[1, 1, 0] = S [384]$	$A[4, 1, 0] = S [576]$
$A[0, 1, 1] = S [321]$	$A[1, 1, 1] = S [385]$	$A[4, 1, 1] = S [577]$
$A[0, 1, 2] = S [322]$	$A[1, 1, 2] = S [386]$	$A[4, 1, 2] = S [578]$
$\vdots$	$\vdots$	$\vdots$
$A[0, 1, 61] = S [381]$	$A[1, 1, 61] = S [445]$	$A[4, 1, 61] = S [637]$
$A[0, 1, 62] = S [382]$	$A[1, 1, 62] = S [446]$	$A[4, 1, 62] = S [638]$
$A[0, 1, 63] = S [383]$	$A[1, 1, 63] = S [447]$	$A[4, 1, 63] = S [639]$

and

$A[0, 2, 0] = S [640]$	$A[1, 2, 0] = S [704]$	$A[4, 2, 0] = S [896]$
$A[0, 2, 1] = S [641]$	$A[1, 2, 1] = S [705]$	$A[4, 2, 1] = S [897]$
$A[0, 2, 2] = S [642]$	$A[1, 2, 2] = S [706]$	$A[4, 2, 2] = S [898]$
$\vdots$	$\vdots$	$\vdots$
$A[0, 2, 61] = S [701]$	$A[1, 2, 61] = S [765]$	$A[4, 2, 61] = S [957]$
$A[0, 2, 62] = S [702]$	$A[1, 2, 62] = S [766]$	$A[4, 2, 62] = S [958]$
$A[0, 2, 63] = S [703]$	$A[1, 2, 63] = S [767]$	$A[4, 2, 63] = S [959]$

etc.

19.2.3.5 Converting state arrays to strings

Let $\mathbf{A}$ denote a state array. The corresponding string representation, denoted by S , can be constructed from the lanes and planes of $\mathbf{A}$, as follows:

For each pair of integers (i, j) , such that $0 \leq i < 5$ and $0 \leq j < 5$, define the string *Lane* (i, j) by using

$$\text{Lane } (i, j) = A[i, j, 0] \parallel A[i, j, 1] \parallel A[i, j, 2] \parallel \dots \parallel A[i, j, w-2] \parallel A[i, j, w-1].$$

For $b = 1\ 600$ and $w = 64$,

$$\text{Lane } (0, 0) = A[0, 0, 0] \parallel A[0, 0, 1] \parallel A[0, 0, 2] \parallel \dots \parallel A[0, 0, 62] \parallel A[0, 0, 63]$$

$$\text{Lane } (1, 0) = A[1, 0, 0] \parallel A[1, 0, 1] \parallel A[1, 0, 2] \parallel \dots \parallel A[1, 0, 62] \parallel A[1, 0, 63]$$

$$\text{Lane } (2, 0) = A[2, 0, 0] \parallel A[2, 0, 1] \parallel A[2, 0, 2] \parallel \dots \parallel A[2, 0, 62] \parallel A[2, 0, 63]$$

etc.

For each integer, j , such that $0 \leq j < 5$, define the string *Plane* (j) by using

$$\text{Plane}(j) = \text{Lane}(0, j) \parallel \text{Lane}(1, j) \parallel \text{Lane}(2, j) \parallel \text{Lane}(3, j) \parallel \text{Lane}(4, j).$$

Then,

$$S = \text{Plane}(0) \parallel \text{Plane}(1) \parallel \text{Plane}(2) \parallel \text{Plane}(3) \parallel \text{Plane}(4).$$

For $b = 1\,600$ and $w = 64$,

$$\begin{aligned} S = & \mathbf{A}[0, 0, 0] \parallel \mathbf{A}[0, 0, 1] \parallel \mathbf{A}[0, 0, 2] \parallel \dots \parallel \mathbf{A}[0, 0, 62] \parallel \mathbf{A}[0, 0, 63] \\ & \parallel \mathbf{A}[1, 0, 0] \parallel \mathbf{A}[1, 0, 1] \parallel \mathbf{A}[1, 0, 2] \parallel \dots \parallel \mathbf{A}[1, 0, 62] \parallel \mathbf{A}[1, 0, 63] \\ & \parallel \mathbf{A}[2, 0, 0] \parallel \mathbf{A}[2, 0, 1] \parallel \mathbf{A}[2, 0, 2] \parallel \dots \parallel \mathbf{A}[2, 0, 62] \parallel \mathbf{A}[2, 0, 63] \\ & \parallel \mathbf{A}[3, 0, 0] \parallel \mathbf{A}[3, 0, 1] \parallel \mathbf{A}[3, 0, 2] \parallel \dots \parallel \mathbf{A}[3, 0, 62] \parallel \mathbf{A}[3, 0, 63] \\ & \parallel \mathbf{A}[4, 0, 0] \parallel \mathbf{A}[4, 0, 1] \parallel \mathbf{A}[4, 0, 2] \parallel \dots \parallel \mathbf{A}[4, 0, 62] \parallel \mathbf{A}[4, 0, 63] \\ & \parallel \mathbf{A}[0, 1, 0] \parallel \mathbf{A}[0, 1, 1] \parallel \mathbf{A}[0, 1, 2] \parallel \dots \parallel \mathbf{A}[0, 1, 62] \parallel \mathbf{A}[0, 1, 63] \\ & \parallel \mathbf{A}[1, 1, 0] \parallel \mathbf{A}[1, 1, 1] \parallel \mathbf{A}[1, 1, 2] \parallel \dots \parallel \mathbf{A}[1, 1, 62] \parallel \mathbf{A}[1, 1, 63] \\ & \parallel \mathbf{A}[2, 1, 0] \parallel \mathbf{A}[2, 1, 1] \parallel \mathbf{A}[2, 1, 2] \parallel \dots \parallel \mathbf{A}[2, 1, 62] \parallel \mathbf{A}[2, 1, 63] \\ & \parallel \mathbf{A}[3, 1, 0] \parallel \mathbf{A}[3, 1, 1] \parallel \mathbf{A}[3, 1, 2] \parallel \dots \parallel \mathbf{A}[3, 1, 62] \parallel \mathbf{A}[3, 1, 63] \\ & \parallel \mathbf{A}[4, 1, 0] \parallel \mathbf{A}[4, 1, 1] \parallel \mathbf{A}[4, 1, 2] \parallel \dots \parallel \mathbf{A}[4, 1, 62] \parallel \mathbf{A}[4, 1, 63] \\ & \vdots \\ & \parallel \mathbf{A}[0, 4, 0] \parallel \mathbf{A}[0, 4, 1] \parallel \mathbf{A}[0, 4, 2] \parallel \dots \parallel \mathbf{A}[0, 4, 62] \parallel \mathbf{A}[0, 4, 63] \\ & \parallel \mathbf{A}[1, 4, 0] \parallel \mathbf{A}[1, 4, 1] \parallel \mathbf{A}[1, 4, 2] \parallel \dots \parallel \mathbf{A}[1, 4, 62] \parallel \mathbf{A}[1, 4, 63] \\ & \parallel \mathbf{A}[2, 4, 0] \parallel \mathbf{A}[2, 4, 1] \parallel \mathbf{A}[2, 4, 2] \parallel \dots \parallel \mathbf{A}[2, 4, 62] \parallel \mathbf{A}[2, 4, 63] \\ & \parallel \mathbf{A}[3, 4, 0] \parallel \mathbf{A}[3, 4, 1] \parallel \mathbf{A}[3, 4, 2] \parallel \dots \parallel \mathbf{A}[3, 4, 62] \parallel \mathbf{A}[3, 4, 63] \\ & \parallel \mathbf{A}[4, 4, 0] \parallel \mathbf{A}[4, 4, 1] \parallel \mathbf{A}[4, 4, 2] \parallel \dots \parallel \mathbf{A}[4, 4, 62] \parallel \mathbf{A}[4, 4, 63]. \end{aligned}$$

19.2.3.6 Labelling convention for the state array

In the diagrams of the state that accompany the specifications of the step mappings, the lane that corresponds to the coordinates $(x, y) = (0, 0)$ is depicted at the centre of the slices.

19.2.3.7 Step mappings

19.2.3.7.1 General

The five step mappings that comprise a round of KECCAK- p are denoted by θ , ρ , π , χ and ι . Specifications for these functions are given in [19.2.3.7.2](#) to [19.2.3.7.6](#).

The algorithm for each step mapping takes a state array, denoted by $\mathbf{A}$, as an input and returns an updated state array, denoted by $\mathbf{A}'$, as the output.

The ι mapping has a second input: an integer called the *round index*, denoted by i_r , which is defined within Algorithm 5 for KECCAK- p , in [19.2.3.7.6](#). The other step mappings do not depend on the *round index*.

19.2.3.7.2 Specification of θ

Algorithm 1: $\theta(\mathbf{A})$

Input: state array **A**

Output: state array **A'**

Steps:

- a) For all pairs (x, z) , such that $0 \leq x < 5$ and $0 \leq z < w$, let

$$C[x, z] = A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z].$$
- b) For all pairs (x, z) , such that $0 \leq x < 5$ and $0 \leq z < w$ let

$$D[x, z] = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w].$$
- c) For all triples (x, y, z) , such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$, let

$$A'[x, y, z] = A[x, y, z] \oplus D[x, z].$$

The effect of θ is to XOR each bit in the state with the parities of two columns in the array. In particular, for the bit $A[x_0, y_0, z_0]$, the x -coordinate of one of the columns is $(x_0 - 1) \bmod 5$, with the same z -coordinate, z_0 , while the x -coordinate of the other column is $(x_0 + 1) \bmod 5$, with z -coordinate $(z_0 - 1) \bmod w$.

19.2.3.7.3 Specification of ρ

Algorithm 2: $\rho(A)$

Input: state array **A**

Output: state array **A'**

Steps:

- a) For all z such that $0 \leq z < w$, let $A'[0, 0, z] = A[0, 0, z]$.
- b) Let $(x, y) = (1, 0)$.
- c) For t from 0 to 23:
 - 1) for all z such that $0 \leq z < w$, let $A'[x, y, z] = A\{x, y, [z - (t + 1)(t + 2)/2] \bmod w\}$;
 - 2) let $(x, y) = [y, (2x + 3y) \bmod 5]$.
- d) Return **A'**.

19.2.3.7.4 Specification of π

Algorithm 3: $\pi(A)$

Input: state array **A**

Output: state array **A'**

Steps:

- a) For all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$, let

$$A'[x, y, z] = A[(x + 3y) \bmod 5, x, z].$$
- b) Return **A'**.

19.2.3.7.5 Specification of χ

Algorithm 4: $\chi(A)$

Input: state array $\mathbf{A}$

Output: state array $\mathbf{A}'$

Steps:

- a) For all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$, let

$$\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z] \oplus \langle \mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1 \rangle \wedge \mathbf{A}[(x+2) \bmod 5, y, z].$$
- b) Return $\mathbf{A}'$.

The notation $\wedge$ in the right side of the assignment for step a) is Boolean “AND” operation.

19.2.3.7.6 Specification of ι

The ι mapping is parameterized by the round index, i_r , whose values are specified in step b) of Algorithm 7 for computing KECCAK- p , in [19.2.3.8](#). Within the specification of ι in Algorithm 6, this parameter determines $l + 1$ bits of a lane value called the *round constant*, denoted by RC , where $l = \log(w)$. When $w = 64$, $l = 6$. Each of these 7 bits is generated by a function that is based on a linear feedback shift register. This function, denoted by rc , is specified in Algorithm 5.

Algorithm 5: $rc(t)$

Input: integer t

Output: bit $rc(t)$

Steps:

- a) If $t \bmod 255 = 0$, return 1.
- b) Let $R = 10\,000\,000$.
- c) For i from 1 to $t \bmod 255$, let:
 - 1) $R = 0 \parallel R$;
 - 2) $R[0] = R[0] \oplus R[8]$;
 - 3) $R[6] = R[6] \oplus R[8]$;
 - 4) $R[3] = R[3] \oplus R[8]$;
 - 5) $R[2] = R[2] \oplus R[8]$;
 - 6) $R = \text{Trunc}_8[R]$.
- d) Return $R[0]$.

In Algorithm 6, RC is a w -bit binary string and denoted as $RC[0], RC[1], \dots, RC[w-1]$.

Algorithm 6: $\iota(\mathbf{A}, i_r)$

Input: state array $\mathbf{A}$; round index i_r

Output: state array $\mathbf{A}'$

Steps:

- a) For all triples (x, y, z) , such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z]$.
- b) Let $RC = 0^w$.
- c) For j from 0 to 6, let $RC[2j - 1] = rc(j + 7i_r)$.