INTERNATIONAL STANDARD

ISO/IEC 18033-4

Second edition
2011-12-15

AMENDMENT 1
2020-08

Information technology — Security techniques — Encryption algorithms —

Part 4: **Stream ciphers**

AMENDMENT 1: ZUC

Technologies de l'information — Techniques de sécurité — Algorithmes de chiffrement —

Partie 4: Chiffrements en flot

AMENDEMENT 1: ZUC

Reference number
ISO/IEC 18033-4:2011/Amd.1:2020(E)

Foreword

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

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

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

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

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

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

A list of all parts in the ISO/IEC 18033 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.

Information technology — Security techniques — Encryption algorithms —

Part 4: Stream ciphers

AMENDMENT 1: ZUC

Introduction

Change the last paragraph as follows:

This document includes six dedicated keystream generators:

— MUGI keystream generator;

— SNOW 2.0 keystream generator;

— Rabbit keystream generator;

— Decim^{v2} keystream generator;

— KCipher-2 (K2) keystream generator; and

— ZUC keystream generator.

4.1

Add the following symbols:

L_1 Linear transform with index 1 used for ZUC.

L_2 Linear transform with index 2 used for ZUC.

SS Subfunction used for ZUC.

$SUB1$ Lookup table with index 1 used for ZUC.

$SUB2$ Lookup table with index 2 used for ZUC.

8.6

Add new subclause 8.6 as follows:

8.6 ZUC keystream generator

8.6.1 Introduction to ZUC

ZUC is a keystream generator which uses as input a 128-bit secret key K and a 128-bit initialization vector IV. These are used to initialize state variables S_i ($i \geq 0$). The bit/byte order is big-endian, i.e., if the key and initialization vector are given as a sequence of bits/bytes, the first/leftmost bit/byte is the most significant bit/byte of the corresponding data. It outputs a 32-bit keystream Z_i at every iteration of the function *Strm*.

The state variable S_i consists of two components. The first consists of sixteen 31-bit variables:

$A^{(i)} = (A_{15}{}^{(i)}, A_{14}{}^{(i)}, ..., A_0{}^{(i)}),$

and maintains the state of a linear feedback shift register. The second consists of two 32-bit variables:

$R^{(i)} = (R_2{}^{(i)}, R_1{}^{(i)}),$

that maintains the state of a finite state machine. ZUC is summarised in Figure 15, which shows a snapshot if its operation, at time *i*, omitting the time-dependent variable (*i*) from the notation.

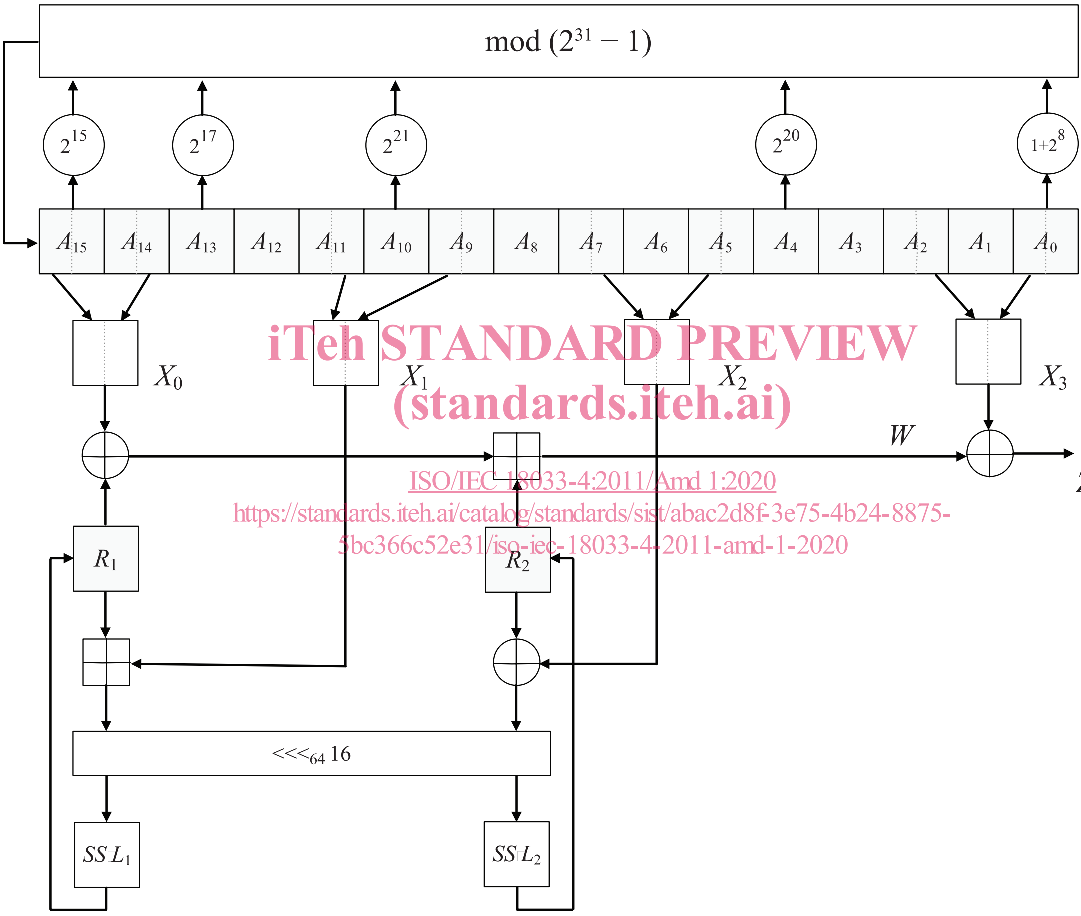

Figure 15 — Schematic drawing of ZUC

The *Init* function, defined in detail in 8.6.2, takes as input the 128-bit key *K* and the 128-bit initialization vector *IV*, and produces the initial value of the state variable $S_0 = (A^{(0)}, R^{(0)})$.

The *Next* function, defined in detail in 8.6.3, takes as input the state variable $S_i = (A^{(i)}, R^{(i)})$ and produces as output the next value of the state variable $S_{i+1} = (A^{(i+1)}, R^{(i+1)})$. The *Next* function runs in two modes, depending on whether the iteration performed is part of the initialization mode or of the normal mode of generating output.

The *Strm* function, defined in detail in 8.6.4, takes as input the state variable $S_i = (A^{(i)}, R^{(i)})$ and produces as output the 32-bit keystream Z_i.

NOTE See document [20] for theoretical background on the design rationale for ZUC.

A 240-bit constant $D = d_0 \,||\, d_1 \,||\, \ldots \,||\, d_{15}$ used in ZUC:

d_0 = 100010011010111, d_1 = 010011010111100, d_2 = 110001001101011, d_3 = 001001101011110,

d_4 = 101011110001001, d_5 = 011010111100010, d_6 = 111000100110101, d_7 = 000100110101111,

d_8 = 100110101111000, d_9 = 010111100010011, d_{10} = 110101111000100, d_{11} = 001101011110001,

d_{12} = 101111000100110, d_{13} = 011110001001101, d_{14} = 111100010011010, d_{15} = 100011110101100,

where for i = 0, 1, ..., 15, d_i is a 15-bit variable in binary notation.

The description uses notations defined in Clause 4 of this part of ISO/IEC 18033. For a string A which has at least 16 bits, the notation A_H represents the leftmost 16 bits of A and the notation A_L represents the rightmost 16 bits of A. For example, if A = 1000100110111110111110101111001 is a 31-bit string, then A_H =1000100110111110 and A_L = 0111110101111001.

8.6.2 Initialization function *Init*

The Initialization function *Init* is as follows.

Input: 128-bit key K, 128-bit initialization vector IV.

Output: Initial value of state variable $S_0 = (A^{(0)}, R^{(0)})$.

a) Initialize the state variable S_{-33} with the key K, the 128-bit initialization vector IV and the constant D.

- Set $(k_0, k_1, \ldots, k_{15}) = K$; $(iv_0, iv_1, \ldots, iv_{15}) = IV$, where k_i and iv_i are bytes for i = 0, 1, ..., 15.

- Set $A_i^{(-33)} = k_i \,||\, d_i \,||\, iv_i$ for i = 0, 1, ..., 15.
- Set $R_1^{(-33)} = R_2^{(-33)} = 0^{(32)}$

b) Set $S_{-1} = Next^{32}(S_{-33}, \text{INIT})$, where $Next^{32}$ denotes 32 iterations of the *Next* function.

c) Set $S_0 = Next(S_{-1}, \text{null})$.

d) Output S_0.

8.6.3 Next-state function *Next*

The *Next* function has two modes, and is defined as follows.

Input: State variable $S_i = (A^{(i)}, R^{(i)})$, mode ∈ {INIT, null}.

Output: Next value of the state variable $S_{i+1} = (A^{(i+1)}, R^{(i+1)})$.

Local variables: 32-bit strings W, W_1, W_2, X_0, X_1, X_2 and 31-bit string V.

a) Set $X_0 = A_{15}{}^{(i)}{}_H \,||\, A_{14}{}^{(i)}{}_L$; $X_1 = A_{11}{}^{(i)}{}_L \,||\, A_9{}^{(i)}{}_H$; $X_2 = A_7{}^{(i)}{}_L \,||\, A_5{}^{(i)}{}_H$.

b) Set $W = (X_0 \oplus R_1^{(i)}) +_{32} R_2^{(i)}$; $W_1 = R_1^{(i)} +_{32} X_1$; $W_2 = R_2^{(i)} \oplus X_2$; $R_1^{(i+1)} = SS(L_1(W_{1L} \,||\, W_{2H}))$; $R_2^{(i+1)} = SS(L_2(W_{2L} \,||\, W_{1H}))$.

c) Set $V = 2^{15}A_{15}^{(i)} + 2^{17}A_{13}^{(i)} + 2^{21}A_{10}^{(i)} + 2^{20}A_4^{(i)} + (1+2^8)A_0^{(i)} \bmod (2^{31}-1)$.

d) If mode = INIT, set $A_{15}^{(i+1)} = V + (31{\sim}W) \bmod (2^{31}-1)$. Otherwise, set $A_{15}^{(i+1)} = V$. If $A_{15}^{(i+1)} = 0$, set $A_{15}^{(i+1)} = 2^{31}-1$.

e) Set $A_j^{(i+1)} = A_{j+1}^{(i)}$ for j = 0, 1, ..., 14.

f) Set $S_{i+1} = (A^{(i+1)}, R^{(i+1)})$.

g) Output S_{i+1}.

NOTE For two 31-bit stings *a* and *b*, if $b = 2^i$, then $ab \bmod (2^{31}\text{-}1) = a <<<_{31} i \bmod (2^{31} - 1)$; if $b = 2^i + 2^j$, then $ab \bmod (2^{31}\text{-}1) = (a <<<_{31} i) + (a <<<_{31} j) \bmod (2^{31} - 1)$. Reference C code for ZUC is given in document [21].

8.6.4 Keystream function *Strm*

The keystream function *Strm* is as follows:

Input: State variable S_i.

Output: 32-bit keystream Z_i.

Local variables: 32-bit strings X_0, X_3.

a) Set $X_0 = A_{15}{}^{(i)}{}_{\mathrm{H}} \,||\, A_{14}{}^{(i)}{}_{\mathrm{L}}$; $X_3 = A_2{}^{(i)}{}_{\mathrm{L}} \,||\, A_0{}^{(i)}{}_{\mathrm{H}}$.

b) Set $Z_i = ((X_0 \oplus R_1{}^{(i)}) +_{32} R_2{}^{(i)}) \oplus X_3$.

c) Output Z_i.

8.6.5 Function *SS*

The function *SS* is as follows:

Input: 32-bit string *X*.

Output: 32-bit string *Y*.

— Define $X = x_3 \,||\, x_2 \,||\, x_1 \,||\, x_0$, where x_i is a byte for $i = 0, 1, 2, 3$.

— Set $Y = SUB1[x_3] \,||\, SUB2[x_2] \,||\, SUB1[x_1] \,||\, SUB2[x_0]$.

— Output *Y*.

The functions *SUB*1 and *SUB*2 are defined by the following substitution tables:

```
SUB1 [256] = {
0x3e,0x72,0x5b,0x47,0xca,0xe0,0x00,0x33,0x04,0xd1,0x54,0x98,0x09,0xb9,0x6d,0xcb,
0x7b,0x1b,0xf9,0x32,0xaf,0x9d,0x6a,0xa5,0xb8,0x2d,0xfc,0x1d,0x08,0x53,0x03,0x90,
0x4d,0x4e,0x84,0x99,0xe4,0xce,0xd9,0x91,0xdd,0xb6,0x85,0x48,0x8b,0x29,0x6e,0xac,
0xcd,0xc1,0xf8,0x1e,0x73,0x43,0x69,0xc6,0xb5,0xbd,0xfd,0x39,0x63,0x20,0xd4,0x38,
0x76,0x7d,0xb2,0xa7,0xcf,0xed,0x57,0xc5,0xf3,0x2c,0xbb,0x14,0x21,0x06,0x55,0x9b,
0xe3,0xef,0x5e,0x31,0x4f,0x7f,0x5a,0xa4,0x0d,0x82,0x51,0x49,0x5f,0xba,0x58,0x1c,
0x4a,0x16,0xd5,0x17,0xa8,0x92,0x24,0x1f,0x8c,0xff,0xd8,0xae,0x2e,0x01,0xd3,0xad,
0x3b,0x4b,0xda,0x46,0xeb,0xc9,0xde,0x9a,0x8f,0x87,0xd7,0x3a,0x80,0x6f,0x2f,0xc8,
0xb1,0xb4,0x37,0xf7,0x0a,0x22,0x13,0x28,0x7c,0xcc,0x3c,0x89,0xc7,0xc3,0x96,0x56,
0x07,0xbf,0x7e,0xf0,0x0b,0x2b,0x97,0x52,0x35,0x41,0x79,0x61,0xa6,0x4c,0x10,0xfe,
0xbc,0x26,0x95,0x88,0x8a,0xb0,0xa3,0xfb,0xc0,0x18,0x94,0xf2,0xe1,0xe5,0xe9,0x5d,
0xd0,0xdc,0x11,0x66,0x64,0x5c,0xec,0x59,0x42,0x75,0x12,0xf5,0x74,0x9c,0xaa,0x23,
0x0e,0x86,0xab,0xbe,0x2a,0x02,0xe7,0x67,0xe6,0x44,0xa2,0x6c,0xc2,0x93,0x9f,0xf1,
0xf6,0xfa,0x36,0xd2,0x50,0x68,0x9e,0x62,0x71,0x15,0x3d,0xd6,0x40,0xc4,0xe2,0x0f,
0x8e,0x83,0x77,0x6b,0x25,0x05,0x3f,0x0c,0x30,0xea,0x70,0xb7,0xa1,0xe8,0xa9,0x65,
0x8d,0x27,0x1a,0xdb,0x81,0xb3,0xa0,0xf4,0x45,0x7a,0x19,0xdf,0xee,0x78,0x34,0x60};

SUB2 [256] = {
0x55,0xc2,0x63,0x71,0x3b,0xc8,0x47,0x86,0x9f,0x3c,0xda,0x5b,0x29,0xaa,0xfd,0x77,
0x8c,0xc5,0x94,0x0c,0xa6,0x1a,0x13,0x00,0xe3,0xa8,0x16,0x72,0x40,0xf9,0xf8,0x42,
0x44,0x26,0x68,0x96,0x81,0xd9,0x45,0x3e,0x10,0x76,0xc6,0xa7,0x8b,0x39,0x43,0xe1,
0x3a,0xb5,0x56,0x2a,0xc0,0x6d,0xb3,0x05,0x22,0x66,0xbf,0xdc,0x0b,0xfa,0x62,0x48,
0xdd,0x20,0x11,0x06,0x36,0xc9,0xc1,0xcf,0xf6,0x27,0x52,0xbb,0x69,0xf5,0xd4,0x87,
0x7f,0x84,0x4c,0xd2,0x9c,0x57,0xa4,0xbc,0x4f,0x9a,0xdf,0xfe,0xd6,0x8d,0x7a,0xeb,
0x2b,0x53,0xd8,0x5c,0xa1,0x14,0x17,0xfb,0x23,0xd5,0x7d,0x30,0x67,0x73,0x08,0x09,
0xee,0xb7,0x70,0x3f,0x61,0xb2,0x19,0x8e,0x4e,0xe5,0x4b,0x93,0x8f,0x5d,0xdb,0xa9,
0xad,0xf1,0xae,0x2e,0xcb,0x0d,0xfc,0xf4,0x2d,0x46,0x6e,0x1d,0x97,0xe8,0xd1,0xe9,
0x4d,0x37,0xa5,0x75,0x5e,0x83,0x9e,0xab,0x82,0x9d,0xb9,0x1c,0xe0,0xcd,0x49,0x89,
0x01,0xb6,0xbd,0x58,0x24,0xa2,0x5f,0x38,0x78,0x99,0x15,0x90,0x50,0xb8,0x95,0xe4,
0xd0,0x91,0xc7,0xce,0xed,0x0f,0xb4,0x6f,0xa0,0xcc,0xf0,0x02,0x4a,0x79,0xc3,0xde,
0xa3,0xef,0xea,0x51,0xe6,0x6b,0x18,0xec,0x1b,0x2c,0x80,0xf7,0x74,0xe7,0xff,0x21,
0x5a,0x6a,0x54,0x1e,0x41,0x31,0x92,0x35,0xc4,0x33,0x07,0x0a,0xba,0x7e,0x0e,0x34,
```

```
0x88,0xb1,0x98,0x7c,0xf3,0x3d,0x60,0x6c,0x7b,0xca,0xd3,0x1f,0x32,0x65,0x04,0x28,
0x64,0xbe,0x85,0x9b,0x2f,0x59,0x8a,0xd7,0xb0,0x25,0xac,0xaf,0x12,0x03,0xe2,0xf2}.
```

8.6.6 Linear transforms L_1 and L_2

Both L_1 and L_2 are linear transforms of 32-bit strings, defined as follows:

$$L_1(X) = X \oplus (X <<<_{32} 2) \oplus (X <<<_{32} 10) \oplus (X <<<_{32} 18) \oplus (X <<<_{32} 24),$$

$$L_2(X) = X \oplus (X <<<_{32} 8) \oplus (X <<<_{32} 14) \oplus (X <<<_{32} 22) \oplus (X <<<_{32} 30).$$

Annex A

Replace the object identifiers in Annex A as follows:

```
EncryptionAlgorithms-4 {
    iso(1) standard(0) encryption-algorithms(18033) part(4)
        asn1-module(0) algorithm-object-identifiers(0) }
    DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

-- IMPORTS None; --

OID ::= OBJECT IDENTIFIER -- Alias

-- Synonyms --

is18033-4 OID ::= { iso(1) standard(0) is18033(18033) part4(4) }

id-kg OID ::= { is18033-4 keystream-generator(1) }
id-scmode OID ::= { is18033-4 stream-cipher-mode(2) }

-- Assignments --

id-kg-mugi OID ::= { id-kg mugi(1) }
id-kg-snow OID ::= { id-kg snow(2) }
id-kg-rabbit OID ::= { id-kg rabbit(3) }
id-kg-decim2 OID ::= { id-kg decim2(4) }
id-kg-k2 OID ::= { id-kg k2(5) }
id-kg-zuc OID ::= { id-kg zuc(6) }

id-scmode-additive OID ::= { id-scmode additive(1) }
id-scmode-multis01 OID ::= { id-scmode multis01(2) }

-- Algorithms and parameters --

StreamCipher ::= AlgorithmIdentifier {{ StreamCipherAlgorithms }}

StreamCipherAlgorithms ALGORITHM ::= {
  additiveStreamCipher |
  multiS01StreamCipher,

... -- Expect additional algorithms --
}

additiveStreamCipher ALGORITHM ::= {
    OID id-scmode-additive PARMS AdditiveStreamCipherParameters
}

AdditiveStreamCipherParameters ::= KeyGenerator

multiS01StreamCipher ALGORITHM ::= {
    OID id-scmode-multis01 PARMS MultiS01StreamCipherParameters
}

MultiS01StreamCipherParameters ::= SEQUENCE {
    keyGenerator KeyGenerator,
```