



BSI Standards Publication

Information technology — Security techniques — Encryption algorithms

Part 4: Stream ciphers

National foreword

This British Standard is the UK implementation of ISO/IEC 18033-4:2011+A1:2020. It supersedes BS ISO/IEC 18033-4:2011, which is withdrawn.

The start and finish of text introduced or altered by amendment is indicated in the text by tags. Tags indicating changes to ISO/IEC text carry the number of the ISO/IEC amendment. For example, text altered by ISO/IEC amendment 1 is indicated by A1 A1.

The UK participation in its preparation was entrusted to Technical Committee IST/33, Information security, cybersecurity and privacy protection.

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

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 2020
Published by BSI Standards Limited 2020

ISBN 978 0 539 04207 8

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 31 January 2012.

Amendments/corrigenda issued since publication

Date	Text affected
30 September 2020	Implementation of ISO/IEC amendment 1:2020

**Information technology — Security
techniques — Encryption algorithms —**

**Part 4:
Stream ciphers**

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

Partie 4: Chiffrements en flot



Reference number
ISO/IEC 18033-4:2011(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2011



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2011

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

Page

Foreword	iv
Introduction.....	v
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Symbols and abbreviated terms	3
4.1 Symbols.....	3
4.2 Functions	5
5 Framework for stream ciphers	6
6 General models for stream ciphers	6
6.1 Keystream generators.....	6
6.2 Output functions.....	7
7 Constructing keystream generators from block ciphers	10
7.1 Block cipher modes for a synchronous keystream generator	10
7.2 Block cipher mode for a self-synchronizing keystream generator	12
8 Dedicated keystream generators	13
8.1 MUGI keystream generator.....	13
8.2 SNOW 2.0 keystream generator	18
8.3 Rabbit keystream generator	23
8.4 Decim ^{v2} keystream generator	27
8.5 KCipher-2 (K2) keystream generator	33
8.6 ZUC keystream generator.....	42
Annex A (normative) Object Identifiers	46
Annex B (informative) Operations over the finite field $GF(2^n)$	48
Annex C (informative) Examples	49
Annex D (informative) Security information.....	95
Bibliography.....	98

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 18033-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 18033-4:2005), which has been technically revised. It also incorporates the Amendment ISO/IEC 18033-4:2005/Amd.1:2009.

ISO/IEC 18033 consists of the following parts, under the general title *Information technology — Security techniques — Encryption algorithms*:

- *Part 1: General*
- *Part 2: Asymmetric ciphers*
- *Part 3: Block ciphers*
- *Part 4: Stream ciphers*

Introduction

This part of ISO/IEC 18033 includes stream cipher algorithms. A stream cipher is an encryption mechanism that uses a keystream to encrypt a plaintext in a bitwise or a block-wise manner. There are two types of stream ciphers: a synchronous stream cipher, in which the keystream is generated from only the secret key (and an initialization vector) and a self-synchronizing stream cipher, in which the keystream is generated from the secret key and some past ciphertexts (and an initialization vector). This part of ISO/IEC 18033 describes both pseudorandom number generators for producing keystream and output functions to combine a keystream with plaintext.

This part of ISO/IEC 18033 includes two output functions:

- Binary-additive output function; and
- MULTI-S01 output function.

A1 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. **A1**

Information technology — Security techniques — Encryption algorithms —

Part 4: Stream ciphers

1 Scope

This part of ISO/IEC 18033 specifies

- a) output functions to combine a keystream with plaintext,
- b) keystream generators for producing keystream, and
- c) object identifiers assigned to dedicated keystream generators in accordance with ISO/IEC 9834.

NOTE 1 The list of assigned object identifiers is given in Annex A.

NOTE 2 Any change to the specification of these algorithms resulting in a change of functional behaviour will result in a change of the object identifier assigned to the algorithms concerned.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 18033-1, *Information technology — Security techniques — Encryption algorithms — Part 1: General*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 18033-1 and the following apply.

3.1

big-endian

method of storage of multi-byte numbers with the most significant bytes at the lowest memory addresses

[ISO/IEC 10118-1:2000]

3.2

ciphertext

data which has been transformed to hide its information content

[ISO/IEC 10116:2006]

3.3 confidentiality
property that information is not made available or disclosed to unauthorized individuals, entities, or processes

3.4 data integrity
property that data has not been altered or destroyed in an unauthorized manner

[ISO/IEC 9797-1:2011]

3.5 decryption
reversal of a corresponding encryption

[ISO/IEC 10116:2006]

3.6 encryption
reversible transformation of data by a cryptographic algorithm to produce ciphertext, i.e., to hide the information content of the data

[ISO/IEC 9797-1:2011]

3.7 initialization value
value used in defining the starting point of an encryption process

3.8 key
sequence of symbols that controls the operation of a cryptographic transformation (e.g., encryption, decryption, cryptographic check function computation, signature generation, or signature verification)

[ISO/IEC 11770-1:2010]

3.9 keystream function
function that takes as input, the current state of the keystream generator and (optionally) part of the previously generated ciphertext, and gives as output the next part of the keystream

3.10 keystream generator
state-based process (i.e., a finite state machine) that takes as input, a key, an initialization vector, and if necessary the ciphertext, and gives as output a keystream (i.e., a sequence of bits or blocks of bits) of arbitrary length

3.11 n -bit block cipher
block cipher with the property that plaintext blocks and ciphertext blocks are n bits in length

[ISO/IEC 10116:2006]

3.12 next-state function
function that takes as input, the current state of the keystream generator and (optionally) part of the previously generated ciphertext, and gives as output a new state for the keystream generator

3.13 output function
function that combines the keystream and the plaintext to produce the ciphertext

NOTE This function is often bitwise XOR.

3.14**padding**

appending extra bits to a data string

[ISO/IEC 10118-1:2000]

3.15**plaintext**

unencrypted information

[ISO/IEC 9797-1:2011]

3.16**secret key**

key used with symmetric cryptographic techniques by a specified set of entities

[ISO/IEC 11770-3:2008]

3.17**state**

current internal state of a keystream generator

4 Symbols and abbreviated terms**4.1 Symbols**

$0x$	Prefix for hexadecimal values.
$0^{(n)}$	n -bit variable where 0 is assigned to every bit.
AND	Bitwise logical AND operation.
$Am^{(i)}[Y]$	The Y -th bit of the register $Am^{(i)}$ in KCipher-2 (K2).
a_i	Variables in an internal state of a keystream generator.
b_i	Variables in an internal state of a keystream generator.
CFB	Cipher FeedBack mode of a block cipher.
CTR	Counter mode of a block cipher.
C_i	Ciphertext block.
D_i	64-bit constants used for MUGL.
e_K	Symmetric block cipher encryption function using secret key K .
F	Subfunction used for MUGL.
FSM	Subfunction used for SNOW 2.0.
$GF(2^n)$	Finite field of exactly 2^n elements.
$GF(2^n)[x]$	The polynomial ring over the finite field $GF(2^n)$.

<i>Init</i>	Function which generates the initial internal state of a keystream generator.
<i>IV</i>	Initialization vector.
<i>IK</i>	Internal key used for KCipher-2 (K2).
<i>K</i>	Key.
$\langle A_1 \rangle L_1$	Linear transform with index 1 used for ZUC. $\langle A_1 \rangle$
$\langle A_1 \rangle L_2$	Linear transform with index 2 used for ZUC. $\langle A_1 \rangle$
<i>M</i>	Subfunction used for MUGI.
<i>Next</i>	Next-state function of a keystream generator.
<i>NLF</i>	Nonlinear function used for KCipher-2 (K2).
<i>n</i>	Block length.
OFB	Output FeedBack mode of a block cipher.
<i>OR</i>	Bitwise logical OR operation.
<i>Out</i>	Output function combining keystream and plaintext in order to generate ciphertext.
<i>P</i>	Plaintext.
P_i	Plaintext block.
<i>R</i>	Additional input to Out.
S_R	Subfunction used for MUGI.
$\langle A_1 \rangle SS$	Subfunction used for ZUC. $\langle A_1 \rangle$
<i>Strm</i>	Keystream function of a keystream generator.
<i>SUB</i>	Lookup table used for MUGI and SNOW 2.0.
$\langle A_1 \rangle SUB1$	Lookup table with index 1 used for ZUC $\langle A_1 \rangle$
$\langle A_1 \rangle SUB2$	Lookup table with index 2 used for ZUC. $\langle A_1 \rangle$
Sub_{K2}	Subfunction used for KCipher-2 (K2).
S_i	Internal state of a keystream generator.

NOTE During normal operation of the cipher, i will increase monotonically starting from zero. However, during initialization of the ciphers, it is convenient from a notational point of view to let i take negative values and define the starting state S_0 in terms of values of S_i for $i < 0$.

<i>T</i>	Subfunction used for SNOW 2.0.
<i>Z</i>	Keystream.
Z_i	Keystream block.
α_{MUL}	Lookup table used for SNOW 2.0.
α_{MUL0}	Lookup table with index 0 used for KCipher-2 (K2).
α_{MUL1}	Lookup table with index 1 used for KCipher-2 (K2).
α_{MUL2}	Lookup table with index 2 used for KCipher-2 (K2).

α_{MUL3}	Lookup table with index 3 used for KCipher-2 (K2).
$\alpha_{\text{inv_MUL}}$	Inverse lookup table used for SNOW 2.0.
ρ_1	Subfunction used for MUGL.
λ_1	Subfunction used for MUGL.
$\lceil x \rceil$	The smallest integer greater than or equal to the real number x .
$\neg x$	Bitwise complement operation.
$\bullet$	Polynomial multiplication.
$\parallel$	Bit concatenation.
$+_m$	Integer addition modulo 2^m .
$\oplus$	Bitwise XOR (eXclusive OR) operation.
$\otimes$	Operation of multiplication of elements in the finite field $GF(2^n)$.

EXAMPLE $C = A \otimes B$: In this operation, the finite field is represented as a selected irreducible polynomial $F(x)$ of degree n with binary coefficients, the n -bit blocks $A = (a_0, a_1, \dots, a_{n-1})$ and $B = (b_0, b_1, \dots, b_{n-1})$ (where the a_i and b_i are bits) are represented as the polynomials, $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1$ and $B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_0$ respectively, then let $C(x) = A(x) \bullet B(x) \bmod F(x)$, i.e., $C(x)$ is the polynomial of degree at most $n-1$ obtained by multiplying $A(x)$ and $B(x)$, dividing the result by $F(x)$, and then taking the remainder. If $C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_0$ (where the c_i are bits) then let C be the n -bit block $(c_0, c_1, \dots, c_{n-1})$.

$\boxplus$	Modular addition operation
$\ll_n t$	t -bit left shift in an n -bit register.
$\gg_n t$	t -bit right shift in an n -bit register.
$\lll_n t$	t -bit left circular rotation in an n -bit register.
$\ggg_n t$	t -bit right circular rotation in an n -bit register.

4.2 Functions

4.2.1 Left-truncation of bits

The operation of selecting the j leftmost bits of an array $A = (a_0, a_1, \dots, a_{m-1})$ to generate a j -bit array is written

$$(j \sim A) = (a_0, a_1, \dots, a_{j-1})$$

This operation is defined only when $1 \leq j \leq m$.

See ISO/IEC 10116:2006.

4.2.2 Shift operation

The operation *Shift* is defined as follows: Given an n -bit variable X and a k -bit variable V where $1 \leq k \leq n$, the effect of the shift function *Shift* is to produce the n -bit variable

$$\text{Shift}_k(X \mid V) = (x_k, x_{k+1}, \dots, x_{n-1}, v_0, v_1, \dots, v_{k-1}) \quad (k < n)$$

$$\text{Shift}_k(X \mid V) = (v_0, v_1, \dots, v_{k-1}) \quad (k = n)$$

The effect is to shift the bits of array X left by k places, discarding $x_0, x_1, \dots, x_{k-1}$ and to place the array V in the rightmost k places of X . When $k = n$ the effect is to totally replace X by V .

See ISO/IEC 10116:2006.

4.2.3 Variable $I(k)$

The variable $I(k)$ is a k -bit variable where 1 is assigned to every bit.

5 Framework for stream ciphers

This clause contains a high-level description of a framework for the stream ciphers specified in this part of ISO/IEC 18033. A detailed description of the general model for a stream cipher is provided in Clause 6. A stream cipher specified in this part of ISO/IEC 18033 is defined by the specification of the following processes:

- a) The keystream generator, which may be either
 - a Synchronous keystream generator, or
 - a Self-synchronizing keystream generator.

NOTE 1 Block cipher modes of operation are methods by which a block cipher can be used to construct a keystream generator. These modes are standardised in ISO/IEC 10116, and the meaning of the functions used in the specification is defined in 6.2.1 and 6.2.2.

NOTE 2 Block ciphers are defined in this part of ISO/IEC 18033.

- b) The output function, which may be either
 - the Binary-additive output function, or
 - the MULTI-S01 output function.

6 General models for stream ciphers

6.1 Keystream generators

6.1.1 Synchronous keystream generators

A synchronous keystream generator is a finite-state machine. It is defined by:

- a) An initialization function, *Init*, which takes as input a key K and an initialization vector IV , and outputs an initial state S_0 for the keystream generator. The initialization vector should be chosen so that no two messages are ever encrypted using the same key and the same IV .

- b) A next-state function, *Next*, which takes as input the current state of the keystream generator S_i , and outputs the next state of the keystream generator S_{i+1} .
- c) A keystream function, *Strm*, which takes as input a state of the keystream generator S_i , and outputs a keystream block Z_i .

When the synchronous keystream generator is first initialized, it will enter an initial state S_0 defined by:

$$S_0 = \text{Init}(IV, K).$$

On demand the synchronous keystream generator will, for $i=0,1,\dots$:

- a) Output a keystream block $Z_i = \text{Strm}(S_i, K)$.
- b) Update the state of the machine $S_{i+1} = \text{Next}(S_i, K)$.

Therefore to define a synchronous keystream generator it is only necessary to specify the functions *Init*, *Next* and *Strm*, including the lengths and alphabets of the key, the initialization vector, the state, and the output block.

6.1.2 Self-synchronizing keystream generators

Generation of a keystream for a self-synchronizing stream cipher is dependent only on previous ciphertexts, the key, and the initialization vector. A general model for a keystream generator for a self-synchronizing stream cipher is now defined:

- a) An initialization function, *Init*, which takes as input a key K and an initialization vector IV and outputs an internal input for the keystream generator S and r dummy ciphertext blocks $C_{-1}, C_{-2}, \dots, C_{-r}$.
- b) A keystream function, *Strm*, that takes as input S and r ciphertext blocks $C_{i-1}, C_{i-2}, \dots, C_{i-r}$, and outputs a keystream block Z_i .

To define a self-synchronizing keystream generator it is only necessary to specify the number of feedback blocks r and the functions *Init* and *Strm*.

NOTE A self-synchronizing stream cipher differs from a synchronous stream cipher in that the keystream depends only on previous ciphertext, the initialization vector and the key, i.e., the keystream generator operates in a stateless fashion. As a result, a decryptor for such a cipher can recover from loss of synchronization after receiving sufficient ciphertext blocks. This also means that the method of keystream generation is dependent upon the selected output function *Out*, which is typically the bitwise XOR operation.

6.2 Output functions

6.2.1 General model of output function

6.2 specifies two stream cipher output functions, i.e., techniques to be used in a stream cipher to combine a keystream with plaintext to derive ciphertext.

An output function for a synchronous or a self-synchronizing stream cipher is a function *Out* that combines a plaintext block P_i , a keystream block Z_i , and some other input R if necessary to give a ciphertext block C_i ($i \geq 0$). A general model for a stream cipher output function is now defined:

Encryption of a plaintext block P_i by a keystream block Z_i is given by:

$$C_i = \text{Out}(P_i, Z_i, R),$$

and decryption of a ciphertext block C_i by a keystream block Z_i is given by:

$$P_i = \text{Out}^{-1}(C_i, Z_i, R).$$

The output function shall satisfy that for any keystream block Z_i , plaintext block P_i , and other input R ,

$$P_i = \text{Out}^{-1}(\text{Out}(P_i, Z_i, R), Z_i, R).$$

6.2.2 Binary-additive output function

A binary-additive stream cipher is a stream cipher in which the keystream, plaintext, and ciphertext blocks are strings of binary digits, and the operation to combine plaintext with keystream is bitwise XOR. The operation Out takes two inputs and does not use any additional information R for calculation. Let n to be the bit length of P_i . This function is specified by

$$\text{Out}(P_i, Z_i, R) = P_i \oplus Z_i.$$

The operation Out^{-1} is specified by

$$\text{Out}^{-1}(C_i, Z_i, R) = C_i \oplus Z_i.$$

NOTE The binary-additive stream cipher does not provide any integrity protection for encrypted data. If data integrity is required, either the MULTI-S01 output function or a separate integrity mechanism should be used, such as a MAC, i.e., a Message Authentication Code (such mechanisms are specified in ISO/IEC 9797).

6.2.3 MULTI-S01 output function

a) General model of MULTI-S01

MULTI-S01 is an output function for a synchronous stream cipher that supports both data confidentiality and data integrity. The MULTI-S01 encryption operation is suitable for use in an online environment. However, the decryption operation of MULTI-S01 can only be performed in an offline situation, as the integrity check is only performed after receiving all the ciphertext blocks. MULTI-S01 has a security parameter n . The computation of Out depends on the choice of a field $GF(2^n)$, i.e., on the choice of an irreducible polynomial over $GF(2)$ of degree n . The MULTI-S01 function only accepts messages whose length is a multiple of n . To encrypt messages whose length is not a multiple of n , a padding mechanism $\text{Pad}(M)$ is required.

NOTE 1 The redundancy R is generated in such a way that the sender and the receiver share it. R can be a fixed public value like 0x00...0.

b) The encryption function $\text{Out}(P, Z, R)$

Input: $n \cdot u$ -bit plaintext P , keystream $Z = (Z_0, Z_1, \dots)$, where Z_i are n -bit blocks, n -bit redundancy R .

Output: Ciphertext C .

- 1) Let t be the lowest value of i ($i \geq 0$) such that $Z_i \neq 0^{(n)}$.
- 2) Let $(P_0, P_1, \dots, P_{u-1}) = P$, where P_i is an n -bit block.
- 3) Set $P_u = Z_t + u + 3$.
- 4) Set $P_{u+1} = R$.

- 5) For each P_i , do the following calculations (for $i = 0, 1, \dots, u + 1$):
- Let $W_i = P_i \oplus Z_{t+i+1}$.
 - Let $X_i = Z_t \otimes W_i$ (in $GF(2^n)$).
 - Let $C_i = X_i \oplus W_{i-1}$, where W_{i-1} is the W value of the previous block $i - 1$, and $W_{-1} = 0^{(n)}$.
 - Set $C = C_0 \parallel C_1 \parallel \dots \parallel C_{u+1}$.
 - Output C .

Figure 1 shows the block diagram of *Out* function.

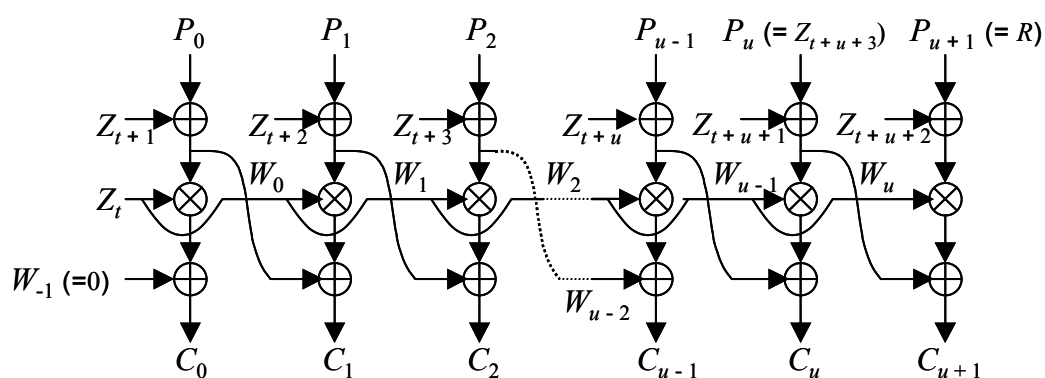


Figure 1 — *Out* function of MULTI-S01 mode

NOTE 2 The irreducible polynomial used to define multiplication in the field depends on n . For instance, in the case of $n = 64$ and 128 , the irreducible polynomial $x^{64} + x^4 + x^3 + x + 1$ and $x^{128} + x^7 + x^2 + x + 1$ can be used.

- c) The decryption function $Out^{-1}(P, Z, R)$

Input: $n \cdot v$ -bit ciphertext C , keystream Z , n -bit redundancy R .

Output: Plaintext P or “*reject*”.

- 1) Let t be the lowest value of i ($i \geq 0$) such that $Z_i \neq 0^{(n)}$.
- 2) Let $(C_0, C_1, \dots, C_{v-1}) = C$, where C_i is an n -bit block.
- 3) For each C_i , do the following calculations (for $i = 0, 1, \dots, v - 1$):
 - Let $X_i = C_i \oplus W_{i-1}$, where $W_{-1} = 0^{(n)}$.
 - Let $W_i = Z_t^{-1} \otimes X_i$ (in $GF(2^n)$).
 - Let $P_i = W_i \oplus Z_{t+i+1}$.
- 4) If $P_{v-2} = Z_{t+v+1}$ and $P_{v-1} = R$, output $P = P_0 \parallel P_1 \parallel \dots \parallel P_{v-3}$ as plaintext. Otherwise, output the special symbol meaning “*reject*” without any text.

Figure 2 shows the block diagram of Out^{-1} function.

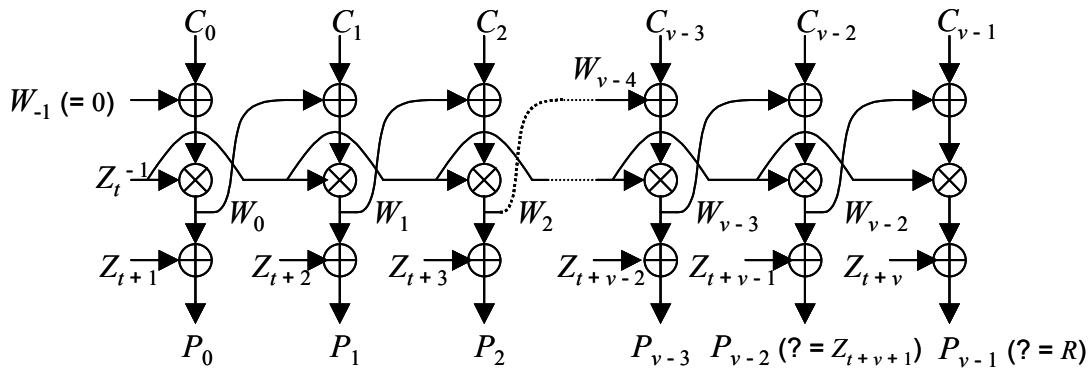


Figure 2 — Out^{-1} function of MULTI-S01 mode

d) Padding mechanism $Pad(M)$

Only when lengths of input messages are not multiples of n , the following padding mechanism $Pad(M)$ is executed:

Input: $(n \cdot v + c)$ -bit string M , where v is a non-negative integer and $0 \leq c < n$.

Output: Padded plaintext P .

- 1) Pad a bit string "1" at the end of the message.
- 2) Pad $(n - c - 1)$ -bit string $0^{(n - c - 1)}$ to the string generated by step a).
- 3) Output the whole data string in length of $(n \cdot v + n)$ bits.

NOTE 3 If the length of the message is a multiple of n in an environment where the length is not certain to be so, this padding mechanism is recommended.

NOTE 4 In order to unpad the message, remove consecutive 0 bits at the end of the data, and remove another bit "1".

7 Constructing keystream generators from block ciphers

7.1 Block cipher modes for a synchronous keystream generator

7.1.1 The OFB (Output FeedBack) mode and the CTR (Counter) mode

Subclause 7.1 specifies two n -bit block cipher modes for a synchronous keystream generator. They are the OFB (Output FeedBack) mode and the CTR (Counter) mode of an n -bit block cipher e_K .

7.1.2 OFB mode

The OFB mode is defined by one parameter r , $1 \leq r \leq n$, which is the size of a plaintext and ciphertext block.

The initialization vector IV is an n -bit string. IV shall be generated differently for two encryptions with the same key K . The functions $Init$, $Next$ and $Strm$ are specified as follows:

— $Init(IV, K) = IV$.

- $Next(S_i, K) = e_K(S_i)$.
- $Strm(S_i) = (r \sim S_i)$.

NOTE $Init(IV, K) = IV$, is equivalent to $S_0 = IV$.

In case of the OFB mode, the binary-additive output function defined in 6.2.2 is used. Figure 3 shows the block diagram of a keystream generator based on CFB mode.

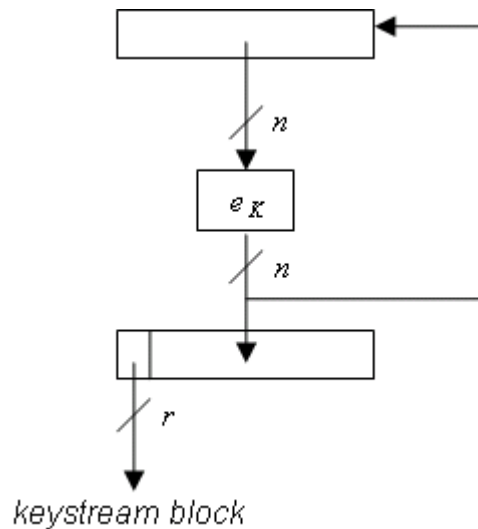


Figure 3 — Keystream generation based on OFB mode

7.1.3 CTR mode

The CTR mode is defined by one parameter r , $1 \leq r \leq n$, which is the size of a plaintext and ciphertext block.

The initialization vector IV is an n -bit string. It shall be assured that $S_i \neq S_j$ for two keystreams $S_0, S_1, S_2, \dots$ and $S_0', S_1', S_2', \dots$ generated with the same key K . The functions $Init$, $Next$ and $Strm$ are specified as follows:

- $Init(IV, K) = IV$.
- $Next(S_i, K) = S_i + 1 \bmod 2^n$.
- $Strm(S_i, K) = (r \sim e_K(S_i))$.

NOTE $Init(IV, K) = IV$, is equivalent to $S_0 = IV$.

In case of the CTR mode, the binary-additive output function defined in 6.2.2 is used. Figure 4 shows the block diagram of a keystream generator based on CFB mode.

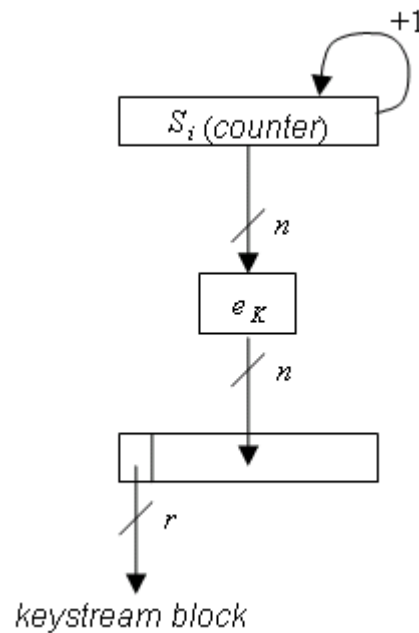


Figure 4 — Keystream generation based on CTR mode

7.2 Block cipher mode for a self-synchronizing keystream generator

7.2.1 Introduction to the CFB mode

The CFB mode of an n -bit block cipher is a self-synchronizing stream cipher.

7.2.2 CFB mode

The CFB (Cipher FeedBack) mode is defined by three parameters, i.e., the size j of feedback buffer S_i , where $n \leq j \leq 1024n$, the size b of feedback variable, where $1 \leq b \leq n$ and the size r of the output block, where $1 \leq r \leq b$.

NOTE 1 The value $b-r$ shall be small compared to b .

The initialization vector IV shall be a randomly generated j -bit string and also shall be generated differently for two encryptions with the same key K . The functions *Init*, *Next* and *Strm* are specified as follows:

- $Init(IV, K) = IV$.
- $Next(S) = Shift_b(S \parallel Shift_r(1(b) \parallel C_i))$.
- $Strm(S, K) = (r \sim e_K((n \sim S)))$.

NOTE 2 $Init(IV, K) = IV$, is equivalent to $S_0 = IV$.

In case of the CFB mode, the binary-additive output function defined in 6.2.2 is used. Figure 5 shows the block diagram of a keystream generator based on CFB mode.

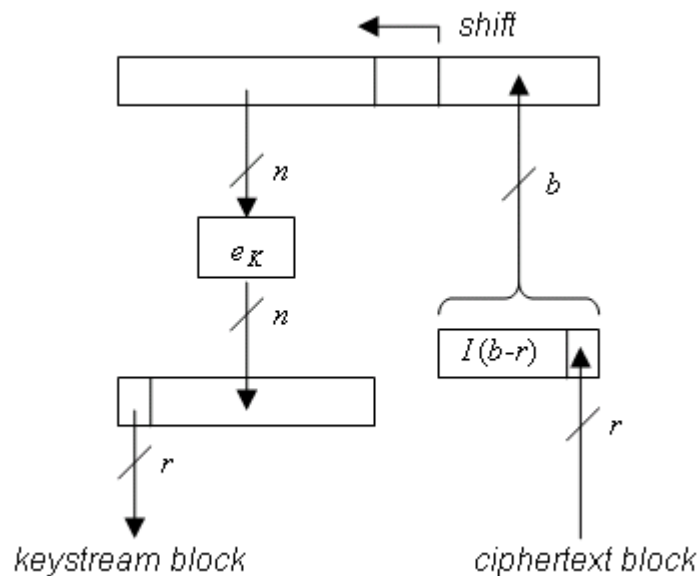


Figure 5 — Keystream generation based on CFB mode

8 Dedicated keystream generators

8.1 MUGI keystream generator

8.1.1 Introduction to MUGI

MUGI is a keystream generator which uses a 128-bit secret key K , a 128-bit initialization vector IV , and a state variable S_i ($i \geq 0$) consisting of 19 64-bit blocks (note that the term block is used through the specification of MUGI for a 64-bit block), and outputs a keystream block Z_i at every iteration of the function $Strm$.

NOTE This keystream generator is originally proposed in [17].

The state variable S_i is sub-divided into a combination of a 3-block variable:

$$a^{(i)} = (a_0^{(i)}, a_1^{(i)}, a_2^{(i)}),$$

where $a_j^{(i)}$ is a block (for $j = 0, 1, 2$), and a 16-block variable:

$$b^{(i)} = (b_0^{(i)}, b_1^{(i)}, \dots, b_{15}^{(i)}),$$

where $b_j^{(i)}$ is a block (for $j = 0, 1, \dots, 15$).

The $Init$ function, defined in detail in 8.1.2, takes as input the 128-bit key K and the 128-bit initializing vector IV , and produces the initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

The $Next$ function, defined in detail in 8.1.3, takes as input the 19-block state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

The $Strm$ function, defined in detail in 8.1.4, takes as input the 19-block state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the keystream block Z_i .

Note that the *Next* function is defined in terms of the functions ρ_1 and λ_1 which are defined in 8.1.5 and 8.1.6, respectively. The function ρ_1 is defined in terms of a function F which is defined in 8.1.7

There are three constants used in MUGI, D_0 in the initialization function *Init*, and D_1, D_2 in ρ_1 . These are given by:

$$D_0 = 0 \times 6A09E667F3BCC908,$$

$$D_1 = 0 \times BB67AE8584CAA73B,$$

$$D_2 = 0 \times 3C6EF372FE94F82B.$$

8.1.2 Initialization function *Init*

The initialization of MUGI is divided into eight steps. The left- and right-half blocks of K are denoted by K_0 and K_1 respectively. IV_0 and IV_1 are defined in the same manner. The initialization function *Init* is as follows:

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

Output: Initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

a) Set the key K into the part of the state variable $a^{(-49)}$ as follows:

- Set $(K_0, K_1) = K$, where K_i is 64 bits for $i = 0, 1$.
- Set $a_0^{(-49)} = K_0$.
- Set $a_1^{(-49)} = K_1$.
- Set $a_2^{(-49)} = (K_0 \lll_{64} 7) \oplus (K_1 \ggg_{64} 7) \oplus D_0$.

D_0 in the above equation is a constant (see 8.1).

b) For $i = -49, -48, \dots, -34$, set $a^{(i+1)} = \rho_1(a^{(i)}, 0^{(64)}, 0^{(64)})$. For the description of ρ_1 see 8.1.5.

c) For $i = 0, 1, \dots, 15$, set $b_{15-i}^{(-16)} = a_0^{(i-48)}$.

d) Add the initialization vector IV into the state a as follows:

- Set $IV_0 \parallel IV_1 = IV$, where IV_i is a block.
- Set $a_0^{(-32)} = a_0^{(-33)} \oplus IV_0$.
- Set $a_1^{(-32)} = a_1^{(-33)} \oplus IV_1$.
- Set $a_2^{(-32)} = a_2^{(-33)} \oplus (IV_0 \lll_{64} 7) \oplus (IV_1 \ggg_{64} 7) \oplus D_0$.

e) For $i = -32, -31, \dots, -17$, set $a^{(i+1)} = \rho_1(a^{(i)}, 0^{(64)}, 0^{(64)})$.

f) Set $S_{-16} = (a^{(-16)}, b^{(-16)})$.

g) Iterate the update function *Next* 16 times:

Set $S_0 = \text{Next}^{16}(S_{-16})$,

where Next^{16} stands for 16 iterations of the next-state function *Next*.

h) Output S_0 .

8.1.3 Next-state function *Next*

The next-state function of MUGI is described as a combination of ρ_1 and λ_1 . The next-state function *Next* of MUGI is as follows:

Input: State variable $S_i = (a^{(i)}, b^{(i)})$.

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

- Set $a^{(i+1)} = \rho_1(a^{(i)}, b_4^{(i)}, b_{10}^{(i)})$. The detailed description of the function ρ_1 is given in 8.1.5.
- Set $b^{(i+1)} = \lambda_1(b^{(i)}, a_0^{(i)})$. The detailed description of the function λ_1 is given in 8.1.6.
- Set $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.
- Output S_{i+1} .

8.1.4 Keystream function *Strm*

The keystream function *Strm* is as follows:

Input: State variable S_i .

Output: Keystream block Z_i .

- Set $Z_i = a_2^{(i)}$.
- Output Z_i .

8.1.5 Function ρ_1

The function ρ_1 is as follows:

Input: State variable $a^{(i)}$, two 64-bit parameters w_1, w_2 .

Output: The next value of the state variable $a^{(i+1)}$.

- Set $a_0^{(i+1)} = a_1^{(i)}$.
- Set $a_1^{(i+1)} = a_2^{(i)} \oplus F(a_1^{(i)}, w_1) \oplus D_1$.
- Set $a_2^{(i+1)} = a_0^{(i)} \oplus F(a_1^{(i)}, (w_2 \lll 17)) \oplus D_2$.
- Output $a^{(i+1)}$.

D_1, D_2 are constants (see 8.1 for details).

Figure 6 shows the block diagram of the function ρ_1 . The detailed description of the function F is given in 8.1.7.

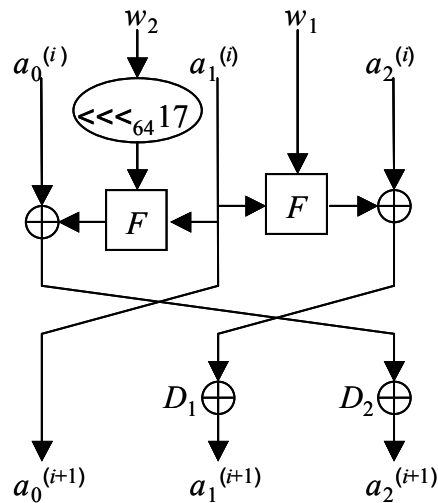


Figure 6 — ρ_1 function of MUGI

8.1.6 Function λ_1

The function λ_1 is as follows:

Input: State variable $b^{(i)}$, 64-bit parameter a' .

Output: The next value of the state variable $b^{(i+1)}$.

- Set $b_j^{(i+1)} = b_{j-1}^{(i)}$ for $j \neq 0, 4, 10$.
- Set $b_0^{(i+1)} = b_{15}^{(i)} \oplus a'$.
- Set $b_4^{(i+1)} = b_3^{(i)} \oplus b_7^{(i)}$.
- Set $b_{10}^{(i+1)} = b_9^{(i)} \oplus (b_{13}^{(i)} \lll_{64} 32)$.

Output $b^{(i+1)}$.

8.1.7 Function F

Function F uses operations over the finite field $GF(2^8)$. In the polynomial representation, $GF(2^8)$ is realized as $GF(2)[x] / f(x)$, where $f(x)$ is an irreducible polynomial of degree 8 defined over $GF(2)$. The MUGI keystream generator uses the following irreducible polynomial:

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

The function F is the composition of a key addition (the data addition from the part of state variable b), a non-linear transformation using the function S_R , a linear transformation using the matrix M and byte shuffling (see Figure 7).

Let us denote the input and the output to the F function as X and Y respectively. Then the function F is as follows:

Input: Two 64-bit strings X and T .

Output: 64-bit string Y .

- $X' = X \oplus T$.
- Set $(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7) = X'$, where X_i is an 8-bit string.
- Set $P_i = S_R(X_i)$ for $i = 0, 1, \dots, 7$.
- Set $P_L = P_0 \parallel P_1 \parallel P_2 \parallel P_3$.
- Set $P_R = P_4 \parallel P_5 \parallel P_6 \parallel P_7$.
- Set $Q_L = M(P_L)$.
- Set $Q_R = M(P_R)$.
- Set $(Q_0, Q_1, Q_2, Q_3) = Q_L$.
- Set $(Q_4, Q_5, Q_6, Q_7) = Q_R$.
- Set $Y = Q_4 \parallel Q_5 \parallel Q_2 \parallel Q_3 \parallel Q_0 \parallel Q_1 \parallel Q_6 \parallel Q_7$.
- Output Y .

Figure 7 shows the block diagram of the function F .

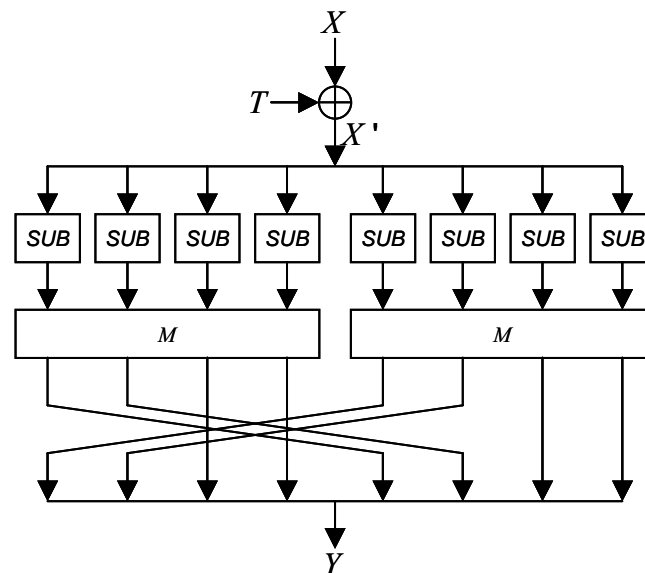


Figure 7 — function F of MUGI

8.1.8 Function S_R

The function S_R is the internal function of F . The function S_R can be described by using a substitution table. In this case, the function S_R is as follows:

Input: 8-bit string x .

Output: 8-bit string y .

— Set $y = SUB[x]$.

— Output y .

The SUB used in function S_R is a substitution as follows:

$SUB[256] = \{$

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

8.1.9 Function M

The function M is the internal function of the F function. The function M is as follows:

Input: 32-bit string X .

Output: 32-bit string Y .

— Set $(x_0, x_1, x_2, x_3) = X$, where x_i is an 8-bit string and an element of $GF(2^8)$.

— Set

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

where 0x01, 0x02, and 0x03 are the hexadecimal expressions of the elements of $GF(2^8)$.

— Set $Y = y_0 \parallel y_1 \parallel y_2 \parallel y_3$.

— Output Y .

8.2 SNOW 2.0 keystream generator

8.2.1 Introduction to SNOW 2.0

SNOW 2.0, in the sequel simply denoted SNOW, is a keystream generator which uses as input a 128 or 256-bit secret key K , and a 128-bit initialization vector IV . These are used to initiate a state variable S_i ($i \geq 0$)

consisting of eighteen $n = 32$ bit blocks. 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 of the corresponding data. For every iteration of the *Strm* function, a 32-bit keystream Z_i is produced as output.

SNOW's state variable S_i consists of two components. First, 16 32-bit variables:

$$a^{(i)} = (a_{15}^{(i)}, a_{14}^{(i)}, \dots, a_0^{(i)}),$$

implements a linear feedback shift register (LFSR). Secondly, 2 32-bit variables:

$$b^{(i)} = (b_2^{(i)}, b_1^{(i)}),$$

maintains the state of a finite state machine (FSM). SNOW is best understood with reference to Figure 8, which shows a snapshot, at time i , omitting the time dependence variable (i) from the notation.

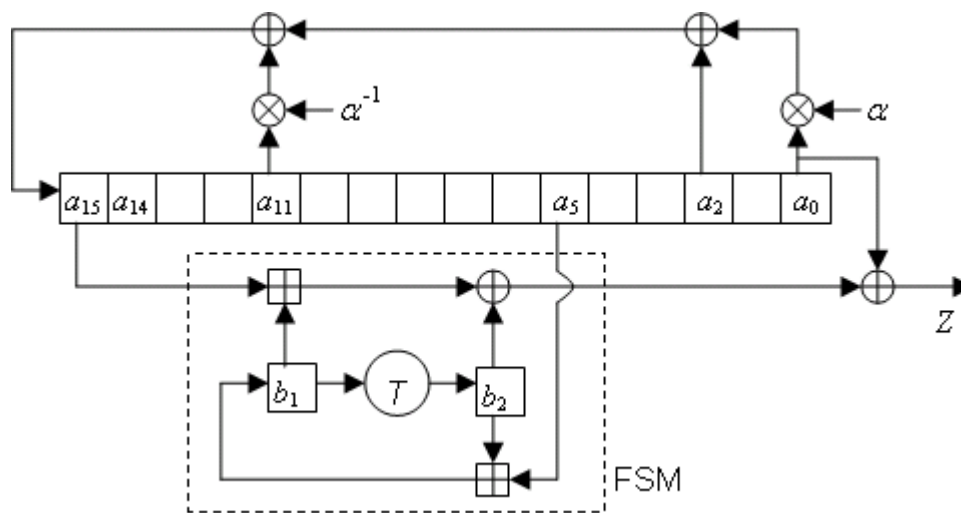


Figure 8 — Schematic drawing of SNOW

SNOW operation is defined by:

The *Init* function, defined in detail in 8.2.2, takes as input the 128 or 256-bit key K and the 128-bit IV , and produces the initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

The *Next* function, defined in detail in 8.2.3, takes as input the 18 32-bit state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$. The *Next* function runs in two modes, depending on whether the iteration performed is part of the initialization, or, of the normal mode of generating output, see below.

The *Strm* function, defined in detail in 8.2.4, takes as input the 18 32-bit state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the 32-bit keystream Z_i .

NOTE 1 For SNOW, the maximum recommended amount of keystream produced from a given (K, IV) is 2^{50} 32 bits. This bound has been selected to provide good security margin against cryptanalysis, and implies no practical limitation in applicability of the algorithm.

NOTE 2 The paper [10] is referred for theoretical background on the design rationale for SNOW.

8.2.2 Initialization function *Init*

The Initialization function *Init* is as follows.

Input: 128- or 256-bit key K , 128-bit initialization vector IV .

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

a) Initialize the registers by the key information

— For a 128-bit key, set $(K_3, K_2, K_1, K_0) = K$, $a_{15-j}^{(-34)} = a_{15-j-8}^{(-34)} = K_{3-j}$, and $a_{15-j-4}^{(-34)} = a_{15-j-12}^{(-34)} = \neg K_{3-j}$ for $j = 0, 1, 2, 3$.

— For a 256-bit key, set $(K_7, K_6, \dots, K_0) = K$, $a_{15-j}^{(-34)} = K_{7-j}$, and $a_{15-j-8}^{(-34)} = \neg(K_{7-j})$ for $j = 0, 1, \dots, 7$.

b) Set $S_{-33} = (a^{(-33)}, b^{(-33)})$ by:

— Set $(IV_3, IV_2, IV_1, IV_0) = IV$.

— Set $a_i^{(-33)} = a_i^{(-34)}$ for $i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 14$.

— Set $a_{15}^{(-33)} = a_{15}^{(-34)} \oplus IV_0$; $a_{12}^{(-33)} = a_{12}^{(-34)} \oplus IV_1$; $a_{10}^{(-33)} = a_{10}^{(-34)} \oplus IV_2$; $a_9^{(-33)} = a_9^{(-34)} \oplus IV_3$.

— Set $b_1^{(-33)} = b_2^{(-33)} = 0^{(32)}$.

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

d) $S_0 = \text{Next}(S_{-1})$.

e) Output S_0 .

8.2.3 Next-state function *Next*

SNOW has two modes for *Next* function.

Input: State variable $S_i = (a^{(i)}, b^{(i)})$, mode = {INIT, null}.

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

a) Set $b_2^{(i+1)} = T(b_1^{(i)})$.

b) Set $b_1^{(i+1)} = b_2^{(i)} +_{32} a_5^{(i)}$.

c) For $j = 0, 1, \dots, 14$ set $a_j^{(i+1)} = a_{j+1}^{(i)}$

d) If INIT mode, set $a_{15}^{(i+1)} = (a_0^{(i)} \otimes \alpha) \oplus a_2^{(i)} \oplus (a_{11}^{(i)} \oplus \alpha^{-1}) \oplus \text{FSM}(a_{15}^{(i)}, b_1^{(i)}, b_2^{(i)})$. Otherwise, set $a_{15}^{(i+1)} = (a_0^{(i)} \otimes \alpha) \oplus a_2^{(i)} \oplus (a_{11}^{(i)} \otimes \alpha^{-1})$.

e) $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

f) Output S_{i+1} .

The description of the T function and the finite field arithmetic involving the fixed element α refers to 8.2.5 and 8.2.6, respectively.

NOTE Figure 9 shows the block diagram of the INIT mode of *Next* function.

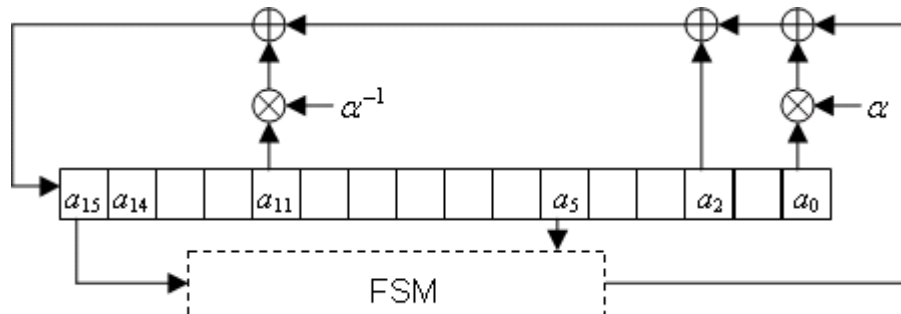


Figure 9 — INIT mode of *Next* function

The definition of the *FSM* function refers to 8.2.8.

8.2.4 Keystream function *Strm*

The keystream function *Strm* is as follows:

Input: State variable S_i .

Output: 32-bit keystream Z_i .

- Set $Z_i = FSM(a_{15}^{(i)}, b_1^{(i)}, b_2^{(i)}) \oplus a_0^{(i)}$.
- Output Z_i .

8.2.5 Function T

The T function is a substitution, specifically a permutation of $GF(2^{32})$, based on components from the Advanced Encryption Standard (AES), ISO/IEC 18033-3. To this end, the finite field $GF(2^8)$ is used, which is viewed as $GF(2)[x]$ modulo the irreducible polynomial

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

and the polynomial ring $GF(2^8)[y]$ modulo $(y^4 + 1)$.

Input: 32-bit string w .

Output: 32-bit string $q = T(w)$.

- Set $(w_3, w_2, w_1, w_0) = w$, where each w_j is 8 bit.
- For $j = 0, 1, 2, 3$ set $t_j = SUB[w_j]$.
- Let $t(y)$ be the polynomial $t(y) = t_3 y^3 + t_2 y^2 + t_1 y + t_0$ in $GF(2^8)[y]$, where t_j is interpreted as an element of $GF(2^8)$ in the natural way: $t_j = t_{j,7} x^7 + \dots + t_{j,1} x + t_{j,0}$ in $GF(2)$.
- set $q(y) = c(y) \cdot t(y)$ modulo $(y^4 + 1)$, where $c(y) = (x+1)y^3 + y^2 + y + x$ in $GF(2^8)[y]$.

e) associate the 32-bit string $q = (q_3, q_2, q_1, q_0)$ with the result of the above, $q(y) = q_3 y^3 + q_2 y^2 + q_1 y + q_0$.

f) Output q .

Note that in step c), the two polynomials are multiplied where coefficient-by-coefficient operations are carried out in $GF(2^8)$ as defined by $f(x)$ above. The result is then reduced modulo $y^4 + 1$.

NOTE 1 The AES S-box can be found in 8.1.8.

NOTE 2 The paper [10] is referred for details of this (and other) optimisation(s).

8.2.6 Multiplication of α in finite field arithmetic

Input: 32-bit string w , representing an element of $GF(2^{32})$.

Output: 32-bit string w' , representing $\alpha \otimes w$ in $GF(2^{32})$.

a) Set $w' = (w \ll_{32} 8) \oplus \alpha_{\text{MUL}}[w \gg_{32} 24]$.

b) Output w' .

The function α_{MUL} is defined in the following:

$\alpha_{\text{MUL}} [256] = \{$
 0x00000000, 0xE19FCF13, 0x6B973726, 0x8A08F835, 0xD6876E4C, 0x3718A15F, 0xBD10596A, 0x5C8F9679,
 0x05A7DC98, 0xE438138B, 0x6E30EBBE, 0x8FAF24AD, 0xD320B2D4, 0x32BF7DC7, 0xB8B785F2, 0x59284AE1,
 0x0AE71199, 0xEB78DE8A, 0x617026BF, 0x80EFE9AC, 0xDC607FD5, 0x3DFFB0C6, 0xB7F748F3, 0x566887E0,
 0x0F40CD01, 0xEEDF0212, 0x64D7FA27, 0x85483534, 0xD9C7A34D, 0x38586C5E, 0xB250946B, 0x53CF5B78,
 0x1467229B, 0xF5F8ED88, 0x7FF015BD, 0x9E6FDAEE, 0xC2E04CD7, 0x237F83C4, 0xA9777BF1, 0x48E8B4E2,
 0x11C0FE03, 0xF05F3110, 0x7A57C925, 0x9BC80636, 0xC747904F, 0x26D85F5C, 0xADC0A769, 0x4D4F687A,
 0x1E803302, 0xFF1FFC11, 0x75170424, 0x9488CB37, 0xC8075D4E, 0x2998925D, 0xA3906A68, 0x420FA57B,
 0x1B27EF9A, 0xFAB82089, 0x70B0D8BC, 0x912F17AF, 0xCDA081D6, 0x2C3F4EC5, 0xA637B6F0, 0x47A879E3,
 0x28CE449F, 0xC9518B8C, 0x435973B9, 0xA2C6BCAA, 0xFE492AD3, 0x1FD6E5C0, 0x95DE1DF5, 0x7441D2E6,
 0x2D699807, 0xCCF65714, 0x46FEAF21, 0xA7616032, 0xFBEEF64B, 0x1A713958, 0x9079C16D, 0x71E60E7E,
 0x22295506, 0xC3B69A15, 0x49BE6220, 0xA821AD33, 0xF4AE3B4A, 0x1531F459, 0x9F390C6C, 0x7EA6C37F,
 0x278E899E, 0xC611468D, 0x4C19BEB8, 0xAD8671AB, 0xF109E7D2, 0x109628C1, 0x9A9ED0F4, 0x7B011FE7,
 0x3CA96604, 0xDD36A917, 0x573E5122, 0xB6A19E31, 0xEA2E0848, 0x0BB1C75B, 0x81B93F6E, 0x6026F07D,
 0x390EBA9C, 0xD891758F, 0x52998DBA, 0xB30642A9, 0xEF89D4D0, 0x0E161BC3, 0x841EE3F6, 0x65812CE5,
 0x364E779D, 0xD7D1B88E, 0x5DD940BB, 0xBC468FA8, 0xE0C919D1, 0x0156D6C2, 0x8B5E2EF7, 0x6AC1E1E4,
 0x33E9AB05, 0xD2766416, 0x587E9C23, 0xB9E15330, 0xE56EC549, 0x04F10A5A, 0x8EF9F26F, 0x6F663D7C,
 0x50358897, 0xB1AA4784, 0x3BA2BFB1, 0xDA3D70A2, 0x86B2E6DB, 0x672D29C8, 0xED25D1FD, 0x0CBA1EEE,
 0x5592540F, 0xB40D9B1C, 0x3E056329, 0xDF9AAC3A, 0x83153A43, 0x628AF550, 0xE8820D65, 0x091DC276,
 0x5AD2990E, 0xBB4D561D, 0x3145AE28, 0xD0DA613B, 0x8C55F742, 0x6DCA3851, 0xE7C2C064, 0x065D0F77,
 0x5F754596, 0xBEEA8A85, 0x34E272B0, 0xD57DBDA3, 0x89F22BDA, 0x686DE4C9, 0xE2651CFC, 0x03FAD3EF,
 0x4452AA0C, 0xA5CD651F, 0x2FC59D2A, 0xCE5A5239, 0x92D5C440, 0x734A0B53, 0xF942F366, 0x18DD3C75,
 0x41F7694, 0xA06AB987, 0x2A6241B2, 0xCBFD8EA1, 0x977218D8, 0x76EDD7CB, 0xFCE52FFE, 0x1D7AE0ED,
 0x4EB5BB95, 0xAF2A7486, 0x25228CB3, 0xC4BD43A0, 0x9832D5D9, 0x79AD1ACA, 0xF3A5E2FF, 0x123A2DEC,
 0x4B12670D, 0xAA8DA81E, 0x2085502B, 0xC11A9F38, 0x9D950941, 0x7C0AC652, 0xF6023E67, 0x179DF174,
 0x78FBCC08, 0x9964031B, 0x136CFB2E, 0xF2F3343D, 0xAE7CA244, 0x4FE36D57, 0xC5EB9562, 0x24745A71,
 0x7D5C1090, 0x9CC3DF83, 0x16CB27B6, 0xF754E8A5, 0xABDB7EDC, 0x4A44B1CF, 0xC04C49FA, 0x21D386E9,
 0x721CDD91, 0x93831282, 0x198BEAB7, 0xF81425A4, 0xA49BB3DD, 0x45047CCE, 0xCF0C84FB, 0x2E934BE8,
 0x77BB0109, 0x9624CE1A, 0x1C2C362F, 0xFDB3F93C, 0xA13C6F45, 0x40A3A056, 0xCAB5863, 0x2B349770,
 0x6C9CEE93, 0x8D032180, 0x070BD9B5, 0xE69416A6, 0xBA1B80DF, 0x5B844FCC, 0xD18CB7F9, 0x301378EA,
 0x693B320B, 0x88A4FD18, 0x02AC052D, 0xE333CA3E, 0xBFBC5C47, 0x5E239354, 0xD42B6B61, 0x35B4A472,
 0x667BFF0A, 0x87E43019, 0x0DECC82C, 0xEC73073F, 0xB0FC9146, 0x51635E55, 0xDB6BA660, 0x3AF46973,
 0x63DC2392, 0x8243EC81, 0x084B14B4, 0xE9D4DBA7, 0xB5B4DDE, 0x54C482CD, 0xDECC7AF8, 0x3F53B5EB};

8.2.7 Multiplication of α^{-1} in finite field arithmetic

Input: 32-bit string y , representing an element of $GF(2^{32})$.

Output: 32-bit string y , representing $\alpha^{-1} \otimes y$ in $GF(2^{32})$.

a) Set $y' = (y \gg_{32} 8) \otimes \alpha_{\text{inv_MUL}}[y \bmod 256]$.

b) Output y' .

The function $\alpha_{\text{inv_MUL}}$ is defined in the following:

```
 $\alpha_{\text{inv\_MUL}}$  [256] = {
0x00000000, 0x180F40CD, 0x301E8033, 0x2811C0FE, 0x603CA966, 0x7833E9AB, 0x50222955, 0x482D6998,
0xC078FBCC, 0xD877BB01, 0xF0667BFF, 0xE8693B32, 0xA04452AA, 0xB84B1267, 0x905AD299, 0x88559254,
0x29F05F31, 0x31FF1FFC, 0x19EEDF02, 0x01E19FCF, 0x49CCF657, 0x51C3B69A, 0x79D27664, 0x61DD36A9,
0xE988A4FD, 0xF187E430, 0xD99624CE, 0xC1996403, 0x89B40D9B, 0x91BB4D56, 0xB9AA8DA8, 0xA1A5CD65,
0x5249BE62, 0x4A46FEAF, 0x62573E51, 0x7A587E9C, 0x32751704, 0x2A7A57C9, 0x026B9737, 0x1A64D7FA,
0x923145AE, 0x8A3E0563, 0xA22FC59D, 0xBA208550, 0xF20DECC8, 0xEA02AC05, 0xC2136CFB, 0xDA1C2C36,
0x7BB9E153, 0x63B6A19E, 0x4BA76160, 0x53A821AD, 0x1B854835, 0x038A08F8, 0x2B9BC806, 0x339488CB,
0xBBC11A9F, 0xA3CE5A52, 0x8BDF9AAC, 0x93D0DA61, 0xDBFDB3F9, 0xC3F2F334, 0xEBE333CA, 0xF3EC7307,
0xA492D5C4, 0xBC9D9509, 0x948C55F7, 0x8C83153A, 0xC4AE7CA2, 0xDCA13C6F, 0xF4B0FC91, 0xECBFC5C,
0x64EA2E08, 0x7CE56EC5, 0x54F4AE3B, 0x4CFBEEF6, 0x04D6876E, 0x1CD9C7A3, 0x34C8075D, 0x2CC74790,
0x8D628AF5, 0x956DCA38, 0xBD7C0AC6, 0xA5734A0B, 0xED5E2393, 0xF551635E, 0xDD40A3A0, 0xC54FE36D,
0x4D1A7139, 0x551531F4, 0x7D04F10A, 0x650BB1C7, 0x2D26D85F, 0x35299892, 0x1D38586C, 0x053718A1,
0xF6DB6BA6, 0xEED42B6B, 0xC6C5EB95, 0xDECAB58, 0x96E7C2C0, 0x8EE8820D, 0xA6F942F3, 0xBEF6023E,
0x36A3906A, 0x2EACD0A7, 0x06BD1059, 0x1EB25094, 0x569F390C, 0x4E9079C1, 0x6681B93F, 0x7E8EF9F2,
0xDF2B3497, 0xC724745A, 0xEF35B4A4, 0xF73AF469, 0xBF179DF1, 0xA718DD3C, 0x8F091DC2, 0x97065D0F,
0x1F53CF5B, 0x075C8F96, 0x2F4D4F68, 0x37420FA5, 0x7F6F663D, 0x676026F0, 0x4F71E60E, 0x577EA6C3,
0xE18D0321, 0xF98243EC, 0xD1938312, 0xC99CC3DF, 0x81B1AA47, 0x99BEEA8A, 0xB1AF2A74, 0xA9A06AB9,
0x21F5F8ED, 0x39FAB820, 0x11EB78DE, 0x09E43813, 0x41C9518B, 0x59C61146, 0x71D7D1B8, 0x69D89175,
0xC87D5C10, 0xD0721CDD, 0xF863DC23, 0xE06C9CEE, 0xA841F576, 0xB04EB5BB, 0x985F7545, 0x80503588,
0x0805A7DC, 0x100AE711, 0x381B27EF, 0x20146722, 0x68390EBA, 0x70364E77, 0x58278E89, 0x4028CE44,
0xB3C4BD43, 0xABCBF8E, 0x83DA3D70, 0x9BD57DBD, 0xD3F81425, 0xCBF754E8, 0xE3E69416, 0xFBE9D4DB,
0x73BC468F, 0x6BB30642, 0x43A2C6BC, 0x5BAD8671, 0x1380EFE9, 0x0B8FAF24, 0x239E6FDA, 0x3B912F17,
0x9A34E272, 0x823BA2BF, 0xAA2A6241, 0xB225228C, 0xFA084B14, 0xE2070BD9, 0xCA16CB27, 0xD2198BEA,
0x5A4C19BE, 0x42435973, 0x6A52998D, 0x725DD940, 0x3A70B0D8, 0x227FF015, 0xA6E30EB, 0x12617026,
0x451FD6E5, 0x5D109628, 0x750156D6, 0x6D0E161B, 0x25237F83, 0x3D2C3F4E, 0x153DFFB0, 0xD32BF7D,
0x85672D29, 0x9D68DE4, 0xB579AD1A, 0xAD76EDD7, 0xE55B844F, 0xFD54C482, 0xD545047C, 0xCD4A44B1,
0x6CEF89D4, 0x74E0C919, 0x5CF109E7, 0x44FE492A, 0x0CD320B2, 0x14DC607F, 0x3CCDA081, 0x24C2E04C,
0xAC977218, 0xB49832D5, 0x9C89F22B, 0x8486B2E6, 0xCCABDB7E, 0xD4A49BB3, 0xFCB55B4D, 0xE4BA1B80,
0x17566887, 0x0F59284A, 0x2748E8B4, 0x3F47A879, 0x776AC1E1, 0x6F65812C, 0x477441D2, 0x5F7B011F,
0xD72E934B, 0xCF21D386, 0xE7301378, 0xFF3F53B5, 0xB7123A2D, 0xAF1D7AE0, 0x870CBA1E, 0x9F03FAD3,
0x3EA637B6, 0x26A9777B, 0x0EB8B785, 0x16B7F748, 0x5E9A9ED0, 0x4695DE1D, 0xE841EE3, 0x768B5E2E,
0xFEDECC7A, 0xE6D18CB7, 0xCEC04C49, 0xD6CF0C84, 0x9EE2651C, 0x86ED25D1, 0xAEFCE52F, 0xB6F3A5E2};
```

8.2.8 Function $FSM(x, y, z)$

Input: Three 32-bit strings, x , y , and z .

Output: 32-bit string q .

a) Set $q = (x +_{32} y) \oplus z$.

b) Output q .

8.3 Rabbit keystream generator

8.3.1 Introduction to Rabbit

Rabbit is a keystream generator which uses a 128-bit secret key K , a 64-bit initialization vector IV , and a 513-bit internal state variable S_i ($i \geq 0$). It outputs a 128-bit keystream block Z_i at every iteration of the function $Strm$.

The 513 bits of the internal state S_i are divided between eight 32-bit state variables $X_0^{(i)}, \dots, X_7^{(i)}$, eight 32-bit counter variables $C_0^{(i)}, \dots, C_7^{(i)}$, and one counter carry bit $b^{(i)}$.

The description uses the notation laid out in Clause 4 of this part of ISO/IEC 18033. In addition, a special notation for bit arrays is used to enhance readability: When labeling the bits of a variable A , the least significant bit is denoted by $A^{(0)}$. The notation $A^{[h..g]}$ represents bits h through g of variable A , where bit position h is more significant than bit position g .

NOTE 1 For Rabbit, the maximum recommended amount of keystream produced from a given key K is 2^{64} keystream blocks. This provides a large security margin against cryptanalysis, while at the same time implying no practical limitations on the applicability of the algorithm.

NOTE 2 The paper [8] is referred for the original proposal of the cipher and the paper [9] is referred for an overview of its cryptographic security.

8.3.2 Additional variables and notation

For the Rabbit keystream generator, the following notation is added:

A	Constant for Rabbit
b	Carry bit for Rabbit
C	Counter variable for Rabbit
g	Subfunction used for Rabbit
X	inner state variable for Rabbit

In addition, a number of other symbols are used for auxiliary local variables in algorithm descriptions. These symbols occur only within a given function specification and do not have a global meaning. They are thus described in the function declaration.

8.3.3 Initialization function *Init*

In the following, the initialization function *Init* of Rabbit is specified.

Input: 128-bit key K , 64-bit initialization vector IV .

Output: Initial value of the state variable $S_0 = (b^{(0)}, X_0^{(0)}, \dots, X_7^{(0)}, C_0^{(0)}, \dots, C_7^{(0)})$.

Local variables: counters i, j

- a) Let $K_0 = K^{[15..0]}$, $K_1 = K^{[31..16]}$, ..., and $K_7 = K^{[127..112]}$.
- b) Set S_{-9} as follows:
 - 1) Set $b^{(-9)} = 0$.
 - 2) For $j = 0, 1, \dots, 7$:
 - If j is even, set $X_j^{(-9)} = K_{(j+1 \bmod 8)} \parallel K_j$ and $C_j^{(-9)} = K_{(j+4 \bmod 8)} \parallel K_{(j+5 \bmod 8)}$.
 - Else, set $X_j^{(-9)} = K_{(j+5 \bmod 8)} \parallel K_{(j+4 \bmod 8)}$ and $C_j^{(-9)} = K_j \parallel K_{(j+1 \bmod 8)}$.
- c) Iterate the next-state function *Next* four times: set $S_i = \text{Next}(S_{i-1})$ for $i = -8, -7, -6, -5$.
- d) Set S_{-4} as follows:
 - 1) Modify the counters as follows:

$$\begin{array}{ll}
 C_0^{(-4)} = C_0^{(-5)} \oplus X_4^{(-5)} \oplus IV^{[31..0]} & C_1^{(-4)} = C_1^{(-5)} \oplus X_5^{(-5)} \oplus (IV^{[63..48]} \parallel IV^{[31..16]}) \\
 C_2^{(-4)} = C_2^{(-5)} \oplus X_6^{(-5)} \oplus IV^{[63..32]} & C_3^{(-4)} = C_3^{(-5)} \oplus X_7^{(-5)} \oplus (IV^{[47..32]} \parallel IV^{[15..0]}) \\
 C_4^{(-4)} = C_4^{(-5)} \oplus X_0^{(-5)} \oplus IV^{[31..0]} & C_5^{(-4)} = C_5^{(-5)} \oplus X_1^{(-5)} \oplus (IV^{[63..48]} \parallel IV^{[31..16]}) \\
 C_6^{(-4)} = C_6^{(-5)} \oplus X_2^{(-5)} \oplus IV^{[63..32]} & C_7^{(-4)} = C_7^{(-5)} \oplus X_3^{(-5)} \oplus (IV^{[47..32]} \parallel IV^{[15..0]})
 \end{array}$$

2) Set $X_0^{(-4)} = X_0^{(-5)}, \dots, X_7^{(-4)} = X_7^{(-5)}, b^{(-4)} = b^{(-5)}$.

e) Iterate the next-state function *Next* four times: set $S_i = \text{Next}(S_{i-1})$ for $i = -3, -2, -1, 0$.

f) Output $S_0 = (b^{(0)}, X_0^{(0)}, \dots, X_7^{(0)}, C_0^{(0)}, \dots, C_7^{(0)})$.

NOTE The *IV* is mixed into the internal state in steps d) and e) of the algorithm. If the application requires frequent re-initialization under the same key, it makes sense to store the internal state after step c) as master state and to perform only steps d) through f) for re-initialization.

8.3.4 Next-state function *Next*

The next-state function *Next* of Rabbit is specified as follows:

Input: State variable $S_i = (b^{(i)}, X_0^{(i)}, \dots, X_7^{(i)}, C_0^{(i)}, \dots, C_7^{(i)})$.

Output: State variable $S_{i+1} = (b^{(i+1)}, X_0^{(i+1)}, \dots, X_7^{(i+1)}, C_0^{(i+1)}, \dots, C_7^{(i+1)})$.

Local variables: counter j , 33-bit positive integer *temp*

a) Set constants $A_0, \dots, A_7$ as follows:

$A_0 = 0 \times 4D34D34D$	$A_1 = 0 \times D34D34D3$	$A_2 = 0 \times 34D34D34$	$A_3 = 0 \times 4D34D34D$
$A_4 = 0 \times D34D34D3$	$A_5 = 0 \times 34D34D34$	$A_6 = 0 \times 4D34D34D$	$A_7 = 0 \times D34D34D3$

b) Let $b_0^{(i+1)} = b^{(i)}$

c) For $j = 0, 1, \dots, 7$:

— Let $\text{temp} = C_j^{(i)} + A_j + b_j^{(i+1)}$; this results in a 33-bit value.

— Let $b_{j+1}^{(i+1)} = \text{temp}^{[32]}$.

— Let $C_j^{(i+1)} = \text{temp}^{[31..0]}$.

d) Let $b^{(i+1)} = b_8^{(i+1)}$

e) For $j = 0, 1, \dots, 7$, let $G_j = g(X_j^{(i)}, C_j^{(i+1)})$, where the function g is given in 8.3.6.

f) Modify internal state as follows:

$$X_0^{(i+1)} = G_0 +_{32} (G_7 \lll_{32} 16) +_{32} (G_6 \lll_{32} 16)$$

$$X_1^{(i+1)} = G_1 +_{32} (G_0 \lll_{32} 8) +_{32} G_7$$

$$X_2^{(i+1)} = G_2 +_{32} (G_1 \lll_{32} 16) +_{32} (G_0 \lll_{32} 16)$$

$$X_3^{(i+1)} = G_3 +_{32} (G_2 \lll_{32} 8) +_{32} G_1$$

$$X_4^{(i+1)} = G_4 +_{32} (G_3 \lll_{32} 16) +_{32} (G_2 \lll_{32} 16)$$

$$X_5^{(i+1)} = G_5 +_{32} (G_4 \lll_{32} 8) +_{32} G_3$$

$$X_6^{(i+1)} = G_6 +_{32} (G_5 \lll_{32} 16) +_{32} (G_4 \lll_{32} 16)$$

$$X_7^{(i+1)} = G_7 +_{32} (G_6 \lll_{32} 8) +_{32} G_5$$

g) Output $S_{i+1} = (b^{(i+1)}, X_0^{(i+1)}, \dots, X_7^{(i+1)}, C_0^{(i+1)}, \dots, C_7^{(i+1)})$.

8.3.5 Keystream function *Strm*

The keystream function *Strm* of Rabbit is specified as follows:

Input: State variable $S_i = (b^{(i)}, X_0^{(i)}, \dots, X_7^{(i)}, C_0^{(i)}, \dots, C_7^{(i)})$.

Output: Keystream block Z_i .

a) Set Z_i as follows:

$$Z_i^{[15..0]} = X_0^{(i)[15..0]} \oplus X_5^{(i)[31..16]}$$

$$Z_i^{[31..16]} = X_0^{(i)[31..16]} \oplus X_3^{(i)[15..0]}$$

$$Z_i^{[47..32]} = X_2^{(i)[15..0]} \oplus X_7^{(i)[31..16]}$$

$$Z_i^{[63..48]} = X_2^{(i)[31..16]} \oplus X_5^{(i)[15..0]}$$

$$Z_i^{[79..64]} = X_4^{(i)[15..0]} \oplus X_1^{(i)[31..16]}$$

$$Z_i^{[95..80]} = X_4^{(i)[31..16]} \oplus X_7^{(i)[15..0]}$$

$$Z_i^{[111..96]} = X_6^{(i)[15..0]} \oplus X_3^{(i)[31..16]}$$

$$Z_i^{[127..112]} = X_6^{(i)[31..16]} \oplus X_1^{(i)[15..0]}$$

b) Output Z_i .

8.3.6 Function *g*

The function *g* is specified as follows:

Input: Two 32-bit parameters u and v .

Output: 32-bit result $g(u, v)$.

Local variables: 64-bit positive integer *temp*

a) Let $temp = (u +_{32} v)^2$; this results in a 64-bit value.

b) Let $g(u, v) = temp^{[31..0]} \oplus temp^{[63..32]}$.

c) Output $g(u, v)$.