

# Information technology — Security techniques — Encryption algorithms —

## Part 4: Stream ciphers

ICS 35.040

# National foreword

This British Standard is the UK implementation of ISO/IEC 18033-4:2005+A1:2009. It supersedes BS ISO/IEC 18033-4:2005 which is withdrawn.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 technology — Security techniques.

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.

**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 5 December 2005

© BSI 2010

ISBN 978 0 580 68084 7

## Amendments/corrigenda issued since publication

| Date         | Comments                                   |
|--------------|--------------------------------------------|
| 30 June 2010 | Implementation of ISO/IEC amendment 1:2009 |
|              |                                            |
|              |                                            |
|              |                                            |

# INTERNATIONAL STANDARD

**ISO/IEC  
18033-4**

First edition  
2005-07-15

---

---

## **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 18033-4:2005(E)

**AVA Documents**  
Document Sharing Hub

## Contents

Page

|                                                                               |    |
|-------------------------------------------------------------------------------|----|
| Foreword.....                                                                 | v  |
| Introduction .....                                                            | vi |
| 1 Scope .....                                                                 | 1  |
| 2 Normative references .....                                                  | 1  |
| 3 Terms and definitions.....                                                  | 1  |
| 4 Symbols and abbreviated terms.....                                          | 4  |
| 4.1 Left-truncation of bits.....                                              | 5  |
| 4.2 Shift operation.....                                                      | 6  |
| 4.3 Variable $l(k)$ .....                                                     | 6  |
| 5 General models for stream ciphers .....                                     | 6  |
| 5.1 Keystream generators .....                                                | 6  |
| 5.1.1 Synchronous keystream generators .....                                  | 6  |
| 5.1.2 Self-synchronizing keystream generators .....                           | 6  |
| 5.2 Output functions .....                                                    | 7  |
| 5.2.1 Binary-additive output function .....                                   | 7  |
| 5.2.2 MULTI-S01 output function .....                                         | 8  |
| 6 Constructing keystream generators from block ciphers.....                   | 10 |
| 6.1 Modes of a block cipher for a synchronous keystream generator.....        | 10 |
| 6.1.1 OFB mode .....                                                          | 11 |
| 6.1.2 CTR mode .....                                                          | 11 |
| 6.2 Mode of a block cipher for a self-synchronizing keystream generator ..... | 12 |
| 6.2.1 CFB mode .....                                                          | 12 |
| 7 Dedicated keystream generators .....                                        | 13 |
| 7.1 MUGI keystream generator .....                                            | 13 |
| 7.1.1 Initialization function <i>Init</i> .....                               | 14 |
| 7.1.2 Next-state function <i>Next</i> .....                                   | 15 |
| 7.1.3 Keystream function <i>Strm</i> .....                                    | 15 |
| 7.1.4 Function $\rho_1$ .....                                                 | 15 |
| 7.1.5 Function $\lambda_1$ .....                                              | 16 |
| 7.1.6 Function $F$ .....                                                      | 16 |
| 7.1.7 Function $S_R$ .....                                                    | 17 |
| 7.1.8 Function $M$ .....                                                      | 18 |
| 7.2 SNOW 2.0 keystream generator .....                                        | 18 |
| 7.2.1 Initialization function <i>Init</i> .....                               | 19 |
| 7.2.2 Next-state function <i>Next</i> .....                                   | 20 |
| 7.2.3 Keystream function <i>Strm</i> .....                                    | 21 |
| 7.2.4 Function $T$ .....                                                      | 21 |
| 7.2.5 Multiplications of $\alpha$ in finite field arithmetic.....             | 22 |
| 7.2.6 Multiplications of $\alpha^{-1}$ in finite field arithmetic .....       | 22 |
| 7.2.7 Function $FSM(x, y, z)$ .....                                           | 23 |
| 7.3 Rabbit keystream generator .....                                          | 23 |
| 7.3.1 Additional variables and notation .....                                 | 24 |
| 7.3.2 Initialization function <i>Init</i> .....                               | 24 |
| 7.3.3 Next-state function <i>Next</i> .....                                   | 25 |
| 7.3.4 Keystream function <i>Strm</i> .....                                    | 26 |
| 7.3.5 Function $g$ .....                                                      | 26 |
| 7.4 Decim <sup>v2</sup> keystream generator.....                              | 26 |
| 7.4.1 Additional variables and notation.....                                  | 27 |
| 7.4.2 Initialization function <i>Init</i> .....                               | 28 |
| 7.4.3 Initialization Next-state function <i>InitNext</i> .....                | 29 |
| 7.4.4 Next-state function <i>Next</i> .....                                   | 30 |

|              |                                                                 |    |
|--------------|-----------------------------------------------------------------|----|
| 7.4.5        | Keystream function <i>Strm</i> .....                            | 31 |
| 7.4.6        | Linear feedback function <i>L</i> .....                         | 31 |
| 7.4.7        | Filtering function <i>F</i> .....                               | 31 |
| 7.4.8        | Boolean function <i>Y</i> .....                                 | 31 |
| 7.4.9        | Compression function <i>ABSG</i> .....                          | 31 |
| 7.4.10       | Buffering function <i>B</i> .....                               | 32 |
| Annex A      | (informative) Examples .....                                    | 33 |
| A.1          | Operations over the finite field $GF(2^n)$ .....                | 33 |
| A.2          | Example for MUGI .....                                          | 33 |
| A.2.1        | Key, initialization vector, and keystream triplets .....        | 33 |
| A.2.2        | Sample internal states .....                                    | 33 |
| A.3          | Example for SNOW 2.0 .....                                      | 39 |
| A.3.1        | 128-bit key .....                                               | 39 |
| A.3.2        | 256-bit key .....                                               | 43 |
| A.4          | Examples for Rabbit .....                                       | 47 |
| A.4.1        | Key, initialization vector and keystream triplets .....         | 47 |
| A.4.2        | Sample internal states .....                                    | 47 |
| A.5          | Examples for Decim <sup>v2</sup> .....                          | 49 |
| A.5.1        | Key, initialization vector and keystream triplets .....         | 49 |
| A.5.2        | Sample internal states .....                                    | 50 |
| Annex B      | (informative) Security information .....                        | 70 |
| B.1          | Security levels of stream ciphers .....                         | 70 |
| B.1.1        | Security-efficiency trade-off in MULTI-S01 .....                | 71 |
| B.2          | Implementation examples of dedicated keystream generators ..... | 71 |
| Annex C      | (normative) Object identifiers .....                            | 72 |
| Bibliography | .....                                                           | 75 |

## 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.

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

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**

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this International Standard may involve the use of patents.

The ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with the ISO and IEC. Information may be obtained from:

***ISO/IEC JTC 1/SC 27 Standing Document 8 (SD8) "Patent Information"***

Standing Document 8 (SD8) is publicly available at: <http://www.ni.din.de/sc27>

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

# Information technology — Security techniques — Encryption algorithms —

## Part 4: Stream ciphers

### 1 Scope

This part of ISO/IEC 18033 specifies stream cipher algorithms. A stream cipher is an encryption mechanism that uses a keystream to encrypt a plaintext in bitwise or block-wise manner. There are two types of stream cipher: a synchronous stream cipher, in which the keystream is only generated from 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). Typically the encryption operation is the additive bitwise XOR operation between a keystream and the message. This part of ISO/IEC 18033 describes both pseudorandom number generators for producing keystream, and output functions for stream ciphers.

The algorithms specified in this part of ISO/IEC 18033 have been assigned object identifiers in accordance with ISO/IEC 9834. The list of assigned object identifiers is given in Annex C. Any change to the specification of the algorithms resulting in a change of functional behaviour will result in a change of the object identifier assigned to the algorithm.

NOTE 1 – In applications where a combination of algorithms is being used, or when an algorithm is parameterized by the choice of a combination of other algorithms, the combination may be specified as a sequence of object identifiers. Alternatively, the object identifiers of lower layer algorithms may be included in the parameter field of the higher layer algorithm's identifier structure. For example, the object identifier of a block cipher could be included as a parameter in the algorithm identifier structure for a keystream generator. The algorithm identifier structure is defined in ISO/IEC 9594-8.

NOTE 2 – The encoding of object identifiers is application dependent.

### 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*

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

### 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**

a method of storage of multi-byte numbers with the most significant bytes at the lowest memory addresses.  
[ISO/IEC 10118-1: 2000]

- 3.2**  
**block**  
string of bits of defined length. [ISO/IEC 18033-1: 2005]
- 3.3**  
**block cipher**  
symmetric encipherment system with the property that the encryption algorithm operates on a block of plaintext, i.e. a string of bits of a defined length, to yield a block of ciphertext. [ISO/IEC 18033-1: 2005]
- 3.4**  
**cipher**  
alternative term for encipherment system. [ISO/IEC 18033-1: 2005]
- 3.5**  
**ciphertext**  
data which has been transformed to hide its information content. [ISO/IEC 10116: 1997]
- 3.6**  
**confidentiality**  
the property that information is not made available or disclosed to unauthorized individuals, entities, or processes. [ISO/IEC 13335-1: 2004]
- 3.7**  
**data integrity**  
the property that data has not been altered or destroyed in an unauthorized manner. [ISO/IEC 9797-1: 1999]
- 3.8**  
**decipherment**  
alternative term for decryption. [ISO/IEC 18033-1: 2005]
- 3.9**  
**decryption**  
reversal of a corresponding encipherment. [ISO/IEC 11770-1: 1996]
- 3.10**  
**encipherment**  
alternative term for encryption. [ISO/IEC 18033-1: 2005]
- 3.11**  
**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: 1996]
- 3.12**  
**initialization value**  
value used in defining the starting point of an encipherment process. [ISO 8372: 1987]
- 3.13**  
**key**  
sequence of symbols that controls the operation of a cryptographic transformation (e.g. encipherment, decipherment). [ISO/IEC 11770-1: 1996]
- 3.14**  
**keystream**  
pseudorandom sequence of symbols, intended to be secret, used by the encryption and decryption algorithms of a stream cipher. If a portion of the keystream is known by an attacker, then it shall be computationally infeasible for the attacker to deduce any information about the remainder of the keystream. [ISO/IEC 18033-1:2005]

### 3.15

#### **keystream function**

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

### 3.16

#### **keystream generator**

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

### 3.17

#### **$n$ -bit block cipher**

block cipher with the property that plaintext blocks and ciphertext blocks are  $n$  bits in length.

[ISO/IEC 10116:1997]

### 3.18

#### **next-state function**

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

### 3.19

#### **output function**

output function that combines the keystream and the plaintext to produce the ciphertext. This function is often bitwise XOR.

### 3.20

#### **padding**

appending extra bits to a data string. [ISO/IEC 10118-1: 2000]

### 3.21

#### **plaintext**

unenciphered information. [ISO/IEC 9797-1: 1999]

### 3.22

#### **secret key**

key used with symmetric cryptographic techniques by a specified set of entities. [ISO/IEC 11770-3: 1999]

### 3.23

#### **self-synchronizing stream cipher**

stream cipher with the property that the keystream symbols are generated as a function of a secret key and a fixed number of previous ciphertext bits. [ISO/IEC 18033-1: 2005]

### 3.24

#### **state**

current internal state of a keystream generator.

### 3.25

#### **stream cipher**

symmetric encryption system with the property that the encryption algorithm involves combining a sequence of plaintext symbols with a sequence of keystream symbols one symbol at a time, using an invertible function. Two types of stream cipher can be identified: synchronous stream ciphers and self-synchronizing stream ciphers, distinguished by the method to obtain the keystream. [ISO/IEC 18033-1: 2005]

### 3.26

#### **synchronous stream cipher**

stream cipher with the property that the keystream symbols are generated as a function of a secret key, and are independent of the plaintext and ciphertext. [ISO/IEC 18033-1: 2005]

## 4 Symbols and abbreviated terms

|                 |                                                                                    |
|-----------------|------------------------------------------------------------------------------------|
| $0^{(n)}$       | $n$ -bit variable where 0 is assigned to every bit.                                |
| 0x              | Prefix for hexadecimal values.                                                     |
| $a_i$           | Variables in an internal state of a keystream generator.                           |
| $\boxtimes$ AND | Bitwise logical AND operation. $\boxtimes$                                         |
| $b_i$           | Variables in an internal state of a keystream generator.                           |
| $C_i$           | Ciphertext block.                                                                  |
| CFB             | Cipher FeedBack mode of a block cipher.                                            |
| CTR             | CounTeR mode of a block cipher.                                                    |
| $D_i$           | 64-bit constants used for MUGI.                                                    |
| $e_K$           | Symmetric block cipher encryption function using secret key $K$ .                  |
| $F$             | Subfunction used for MUGI.                                                         |
| $FSM$           | Subfunction used for SNOW 2.0.                                                     |
| $F[x]$          | The polynomial ring over the finite field $F$ .                                    |
| $GF(2^n)$       | Finite field of exactly $2^n$ elements.                                            |
| $Init$          | Function which generates the initial internal state of a keystream generator.      |
| $IV$            | Initialization vector.                                                             |
| $K$             | Key.                                                                               |
| $M$             | Subfunction used for MUGI.                                                         |
| $n$             | Block length.                                                                      |
| $Next$          | Next-state function of a keystream generator.                                      |
| OFB             | Output FeedBack mode of a block cipher.                                            |
| $\boxplus$ OR   | Bitwise logical OR operation. $\boxplus$                                           |
| $Out$           | Output function combining keystream and plaintext in order to generate ciphertext. |
| $P$             | Plaintext.                                                                         |
| $P_i$           | Plaintext block.                                                                   |
| $R$             | Additional input to $Out$ .                                                        |
| $S_i$           | 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  $S_i$  values for  $i < 0$ .

$S_i$  Subfunction used for MUGI.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $Strm$              | Keystream function of a keystream generator.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| $SUB$               | Lookup table used for MUGI and SNOW 2.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| $T$                 | Subfunction used for SNOW 2.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| $Z$                 | Keystream.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| $Z_i$               | Keystream block.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| $\alpha_{MUL}$      | Lookup table used for SNOW 2.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| $\alpha_{inv\_MUL}$ | Lookup table used for SNOW 2.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| $\rho_1$            | Subfunction used for MUGI.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| $\lambda_1$         | Subfunction used for MUGI.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| $\lceil x \rceil$   | 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 $GF(2^n)$ . E.g. $C = A \otimes B$ : 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 $A = \{a_{n-1}, a_{n-2}, \dots, a_0\}$ and $B = \{b_{n-1}, b_{n-2}, \dots, b_0\}$ (where the $a_i$ and $b_i$ are bits) are represented as the polynomials, $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0$ 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 $\{c_{n-1}, c_{n-2}, \dots, c_0\}$ . |
| $\ll_n t$           | $t$ -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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

**[A1] NOTE** Additional variables and notation specific to a given keystream generator are introduced with the algorithm. **[A1]**

#### 4.1 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  $1 \leq j \leq m$ . [ISO/IEC 10116]

## 4.2 Shift operation

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

$$\text{Shift}_k(X | F) = (x_k, x_{k+1}, \dots, x_{n-1}, f_0, f_1, \dots, f_{k-1}) \quad (k < n)$$

$$\text{Shift}_k(X | F) = (f_0, f_1, \dots, f_{k-1}) \quad (k = n)$$

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  $F$  in the rightmost  $k$  places of  $X$ . When  $k = n$  the effect is to totally replace  $X$  by  $F$ . [ISO/IEC 10116]

## 4.3 Variable $I(k)$

The variable  $I(k)$  is a  $k$ -bit variable where 1 is assigned to every bit. [ISO/IEC 10116]

# 5 General models for stream ciphers

In this clause general models for stream ciphers are specified. A stream cipher consists of the combination of a keystream generator and an output function.

## 5.1 Keystream generators

### 5.1.1 Synchronous keystream generators

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

1. 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  $IV$ .
2. 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  $S_{i+1}$ .
3. 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  $i=0,1,\dots$ :

1. Output a keystream block  $Z_i = \text{Strm}(S_i, K)$ .
2. 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*, along with the lengths and alphabets of the key, the initialization vector, the state, and the output block.

### 5.1.2 Self-synchronizing keystream generators

Generation of 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.

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 binary additive mode.

1. 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  $C_{-1}, C_{-2}, \dots, C_{-r}$ .
2. 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*.

## 5.2 Output functions

In this clause two stream cipher output functions are specified, 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 an invertible function *Out* that combines a plaintext block  $P_i$ , a keystream block  $Z_i$  and, optionally, some other input *R* to give a ciphertext block  $C_i$  ( $i \geq 0$ ). A general model for 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 be such that, for any keystream block  $Z_i$ , plaintext block  $P_i$ , and other input *R*, we have that

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

### 5.2.1 Binary-additive output function

A binary-additive stream cipher is a stream cipher in which the keystream, plaintext, and ciphertext blocks are 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 its calculation. Let *n* to be the bit length of  $P_i$ . This function is specified by

$$\text{Set } R = 0^{(n)}.$$

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

The operation  $\text{Out}^{-1}$  is specified by

$$\text{Set } R = 0^{(n)}.$$

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

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 Message Authentication Code (such mechanisms are specified in ISO/IEC 9797).

## 5.2.2 MULTI-S01 output function

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 MULTI-S01 function only accepts messages whose length is a multiple of  $n$ . To encrypt messages whose length is not a multiple of  $n$ , it is necessary to pad the plaintext. A padding technique is described in clause 5.2.2.3. In addition to  $P$  and  $Z$ , the MULTI-S01 takes as another input some redundancy  $R$ . One possibility is to make  $R$  a fixed public  $n$ -bit value. This should be available to both the encryption and decryption processes.

NOTE 1 An encryption mechanism based on MULTI-S01 is secure as long as the underlying keystream generator is secure. The security level with respect to data integrity is generally as high as the security level of the keystream generator. However the security level is always upper-bounded by the length of a forged message. It is believed that a forged message of length  $u \cdot n$  bits will be accepted with probability at most  $(u-2) \cdot 2^{-n}$ , where  $n$  is the security parameter of the mode. For practical purposes, the parameter  $n$  should be at least 64. For higher security,  $n=128$  is recommended.

NOTE 2 – MULTI-S01 was originally proposed in [3].

### 5.2.2.1 *Out*( $P, Z, R$ )

The *Out* function operates in an iterative fashion, processing  $n$  bits of plaintext per iteration.

**INPUT:**  $n \cdot u$ -bit plaintext  $P$ , keystream  $Z$ ,  $n$ -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. Set  $W_{-1} = 0^{(n)}$ .
6. For  $i = 0, 1, \dots, u+1$  do the following calculations:
  - 6.1. Let  $W_i = P_i \oplus Z_{t+i+1}$ .
  - 6.2. Let  $X_i = Z_i \otimes W_i$  (in  $\text{GF}(2^n)$ ).
  - 6.3. Let  $C_i = X_i \oplus W_{i-1}$ .
7. Set  $C = C_0 \parallel C_1 \parallel \dots \parallel C_{u+1}$ .
8. Output  $C$ .

Figure 1 shows the operation of the *Out* function.

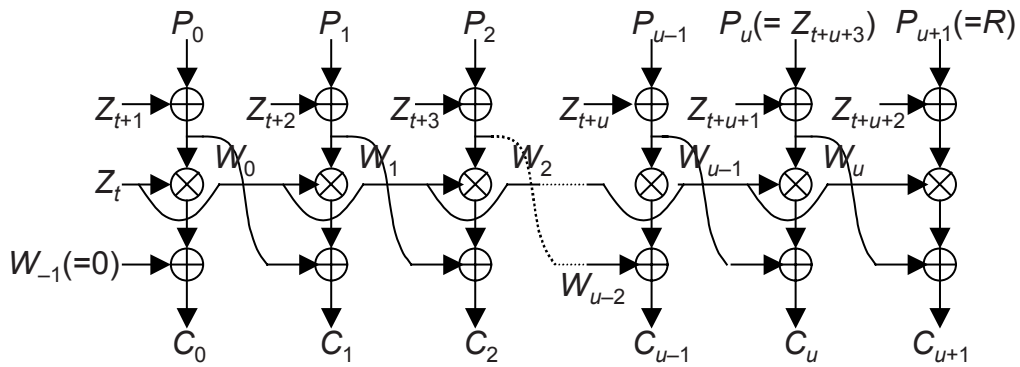


Figure 1 — *Out* function of MULTI-S01 mode

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

#### 5.2.2.2 *Out*<sup>-1</sup>( $C, Z, R$ )

Although the *Out*<sup>-1</sup> function does not output any text before the integrity check, the core computation of the *Out*<sup>-1</sup> function is also incremental and keeps an internal variable  $W$ . To generate a pre-plaintext value  $P_i$ , the function requires  $W_{i-1}$  value in addition to  $C_i$ , generating  $W_i$  for the following process.

**INPUT:**  $n$ - $v$ -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  $i = 0, 1, \dots, v-1$ ):
  - 3.1. Let  $X_i = C_i \oplus W_{i-1}$ , where  $W_{-1} = 0^{(n)}$ .
  - 3.2. Let  $W_i = Z_t^{-1} \otimes X_i$  (in  $\text{GF}(2^n)$ ).
  - 3.3. 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 operation of the  $Out^{-1}$  function.

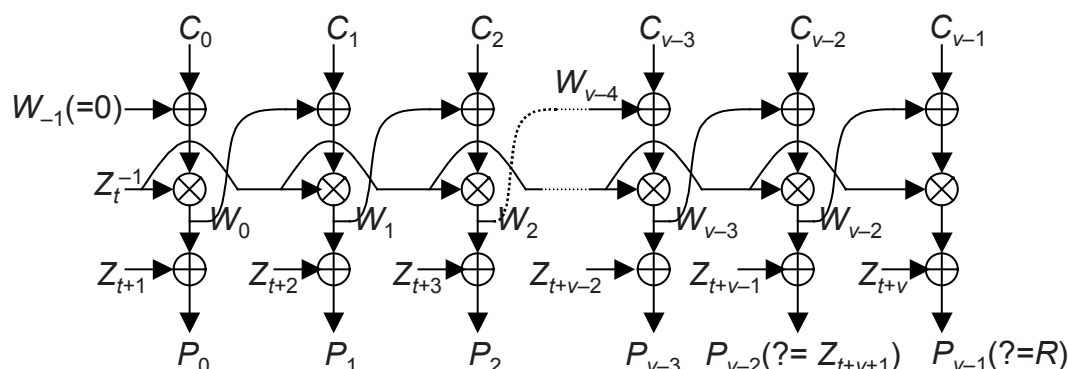


Figure 2 —  $Out^{-1}$  function of MULTI-S01 mode

### 5.2.2.3 Padding mechanism $Pad(M)$

There is a recommended padding method for implementations where lengths of input messages are not guaranteed to be multiples of  $n$ .

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

**OUTPUT:** Padded plaintext  $P$ .

1. Append a single '1' bit to the end of the message.
2. Append the  $(n - c - 1)$ -bit string  $0^{(n-c-1)}$  to the end of the string generated by step 1.
3. Output the whole data string of length  $(nv + n)$  bits.

NOTE 1 – If the length of the message is a multiple of  $n$  in an environment where this situation is not guaranteed, then use of this padding mechanism is recommended.

NOTE 2 – In order to unpad the message, remove consecutive 0 bits at the end of the data, and remove another bit "1".

## 6 Constructing keystream generators from block ciphers

In this clause three modes of operation for block ciphers are specified. These modes are standardised in ISO/IEC 10116, and the meaning of the functions used in the specification is defined in clauses 5.1.1 and 5.1.2 of this document. These three block cipher modes of operation are methods by which a block cipher can be used to construct a keystream generator. The first two of these three block cipher modes of operation, specified in clause 6.1, define synchronous stream ciphers; the third block cipher mode, specified in clause 6.2, defines a self-synchronizing stream cipher.

NOTE – Block ciphers are defined in part 3 of this international standard.

### 6.1 Modes of a block cipher for a synchronous keystream generator

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

### 6.1.1 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 functions *Init*, *Next* and *Strm* are specified as follows:

1.  $Init(IV, K) = IV$ .
2.  $Next(S_i, K) = e_K(S_i)$ .
3.  $Strm(S_i) = (r \sim S_i)$ .

The initialization vector  $IV$  shall be a randomly generated  $n$ -bit string. Figure 3 shows the operation of a keystream generator based on OFB mode.

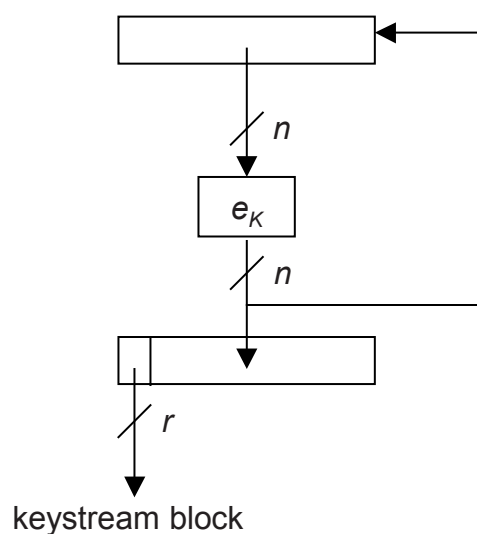


Figure 3 — Keystream generation based on OFB mode

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

### 6.1.2 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 functions *Init*, *Next* and *Strm* are specified as follows:

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

The initialization vector  $IV$  shall be a randomly generated  $n$ -bit string. Figure 4 shows the operation of a keystream generator based on CTR mode.

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

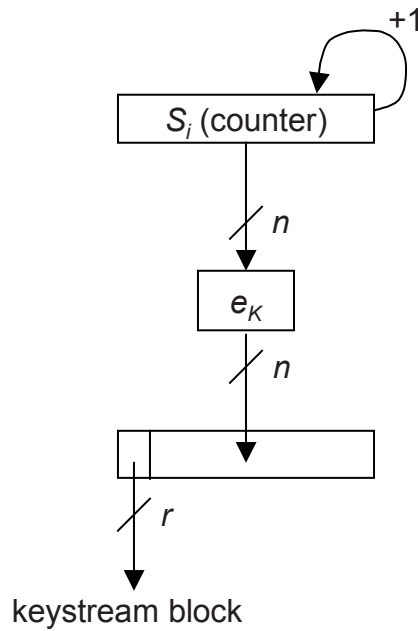


Figure 4 — Keystream generation based on CTR mode

## 6.2 Mode of a block cipher for a self-synchronizing keystream generator

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

### 6.2.1 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  $1 \leq r \leq b$ . The initialization vector  $IV$  shall be a randomly generated  $j$ -bit string.

The functions *Init*, *Next* and *Strm* are specified as follows:

1.  $Init(IV, K) = IV$ .
2.  $Next(S_i) = Shift_b(S \mid Shift_r(I(b) \mid C_i))$ .
3.  $Strm(S, K) = r \sim e_K(n \sim S)$ .

Figure 5 shows the operation of a keystream generator based on CFB mode.

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

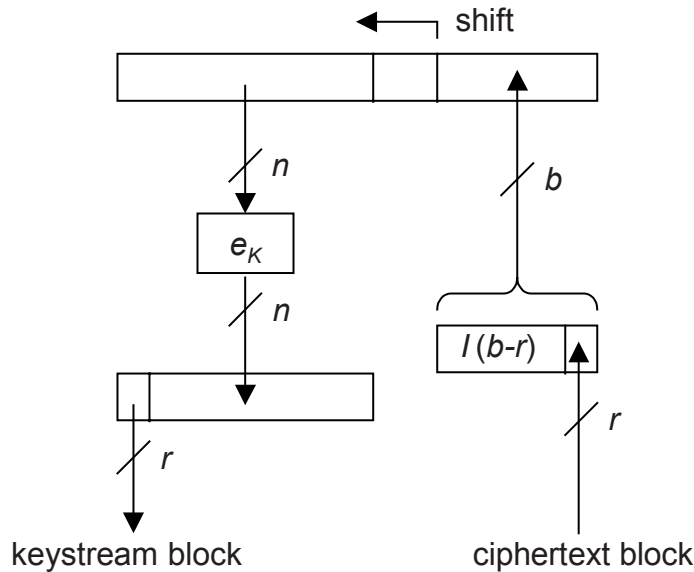


Figure 5 — Keystream generation based on CFB mode

## 7 Dedicated keystream generators

In this clause, two dedicated keystream generators for a synchronous stream cipher are specified, i.e. methods for generating a keystream from a key and an initialization vector.

### 7.1 MUGI keystream generator

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 was originally proposed in [5].

The state variable  $S_i$  is made up of 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 clause 7.1.1, 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 clause 7.1.2, 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 clause 7.1.3, 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  $\rho_1$  and  $\lambda_1$  which are defined in clauses 7.1.4 and 7.1.5, respectively. The function  $\rho_1$  is defined in terms of a function  $F$  which is defined in clause 7.1.6.

There are three constants used in MUGI,  $D_0$  in the initialization function *Init*, and  $D_1$ ,  $D_2$  in  $\rho_1$ . These are given by:

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

$$D_1 = 0 \times BB67AE8584CAA73B,$$

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

### 7.1.1 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 described as follows:

**INPUT:** 128-bit key  $K$ , 128-bit initialization vector  $IV$ .

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

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

1.1. Set  $(K_0, K_1) = K$ , where  $K_i$  is 64 bits for  $i=1,2$ .

1.2. Set  $a_0^{(-49)} = K_0$ .

1.3. Set  $a_1^{(-49)} = K_1$ .

1.4. 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 clause 7.1).

2. For  $i = -49, -48, \dots, -34$ , set  $a^{(i+1)} = \rho_1(a^{(i)}, 0^{(64)}, 0^{(64)})$ . For the description of  $\rho_1$  see clause 7.1.4.

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

4. Add the initialization vector  $IV$  into the state  $a$  as follows:

4.1. Set  $IV_0 \parallel IV_1 = IV$ , where  $IV_i$  is a block.

4.2. Set  $a_0^{(-32)} = a_0^{(-33)} \oplus IV_0$ .

4.3. Set  $a_1^{(-32)} = a_1^{(-33)} \oplus IV_1$ .

4.4. Set  $a_2^{(-32)} = a_2^{(-33)} \oplus (IV_0 \lll_{64} 7) \oplus (IV_1 \ggg_{64} 7) \oplus D_0$ .

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

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

7. Iterate the update function *Next* 16 times:

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

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

8. Output  $S_0$ .

### 7.1.2 Next-state function *Next*

The next-state function of MUGI is described as a combination of  $\rho_1$  and  $\lambda_1$ . The next-state function *Next* of MUGI is described as follows:

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

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

1. Set  $a^{(i+1)} = \rho_1(a^{(i)}, b_4^{(i)}, b_{10}^{(i)})$ . The detailed description of the function  $\rho_1$  is given in clause 7.1.4.
2. Set  $b^{(i+1)} = \lambda_1(b^{(i)}, a_0^{(i)})$ . The detailed description of the function  $\lambda_1$  is given in clause 7.1.5.
3. Set  $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$ .
4. Output  $S_{i+1}$ .

### 7.1.3 Keystream function *Strm*

The keystream function *Strm* is described as follows:

**INPUT:** State variable  $S_i$ .

**OUTPUT:** Keystream block  $Z_i$ .

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

### 7.1.4 Function $\rho_1$

The function  $\rho_1$  is described as follows:

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

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

1. Set  $a_0^{(i+1)} = a_1^{(i)}$ .
2. Set  $a_1^{(i+1)} = a_2^{(i)} \oplus F(a_1^{(i)}, w_1) \oplus D_1$ .
3. Set  $a_2^{(i+1)} = a_0^{(i)} \oplus F(a_1^{(i)}, (w_2 \lll_{64} 17)) \oplus D_2$ .
4. Output  $a^{(i+1)}$ .

$D_1, D_2$  are constants (see clause 7.1 for details).

Figure 6 shows the operation of the  $\rho_1$  function.

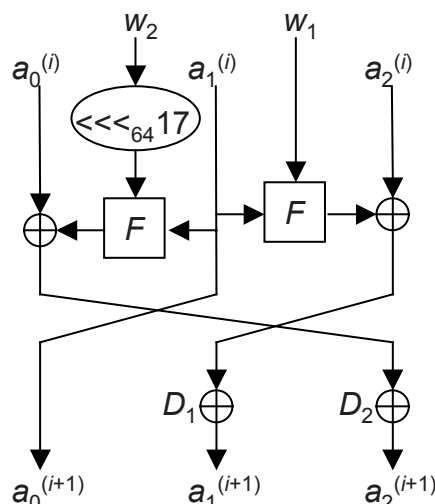


Figure 6 —  $\rho$  function of MUGI  
1

The detailed description of the function  $F$  is given in clause 7.1.6.

### 7.1.5 Function $\lambda_1$

The function  $\lambda_1$  is described as follows:

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

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

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

### 7.1.6 Function $F$

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

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

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 described as follows:

**INPUT:** Two 64-bit strings  $X$  and  $T$ .

**OUTPUT:** 64-bit string  $Y$ .

1.  $X' = X \oplus T$ .
2. 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.
3. Set  $P_i = S_R(X_i)$  for  $i = 0, 1, \dots, 7$ .
4. Set  $P_L = P_0 \parallel P_1 \parallel P_2 \parallel P_3$ .
5. Set  $P_R = P_4 \parallel P_5 \parallel P_6 \parallel P_7$ .
6. Set  $Q_L = M(P_L)$ .
7. Set  $Q_R = M(P_R)$ .
8. Set  $(Q_0, Q_1, Q_2, Q_3) = Q_L$ .
9. Set  $(Q_4, Q_5, Q_6, Q_7) = Q_R$ .
10. Set  $Y = Q_4 \parallel Q_5 \parallel Q_2 \parallel Q_3 \parallel Q_0 \parallel Q_1 \parallel Q_6 \parallel Q_7$ .
11. Output  $Y$ .

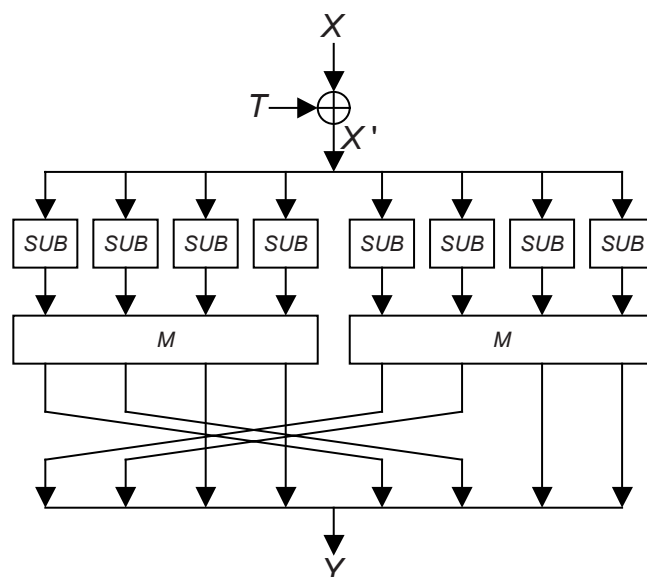


Figure 7 —  $F$  function of MUGI

### 7.1.7 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 described as follows:

**INPUT:** 8-bit string  $x$ .

**OUTPUT:** 8-bit string  $y$ .

1. Set  $y = SUB[x]$ .
2. Output  $y$ .

The *SUB* used in function  $S_R$  is a substitution table 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}

NOTE — The *SUB* is identical to the S box defined in the Advanced Encryption Standard [ISO/IEC 18033-3].

### 7.1.8 Function *M*

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

**INPUT:** 32-bit string *X*.

**OUTPUT:** 32-bit string *Y*.

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

3. Set  $Y = y_0 \parallel y_1 \parallel y_2 \parallel y_3$ .
4. Output *Y*.

## 7.2 SNOW 2.0 keystream generator

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.

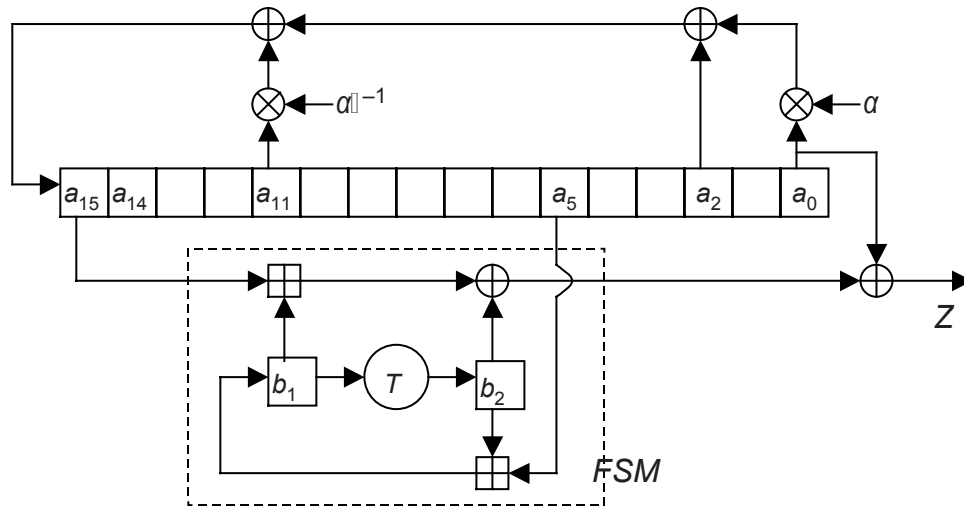


Figure 8 — Schematic drawing of SNOW

SNOW operation is defined by:

The *Init* function, defined in detail in clause 7.2.1, 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 clause 7.2.2, 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 clause 7.2.3, 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 – We refer to the paper [2] for theoretical background on the design rationale for SNOW.

### 7.2.1 Initialization function *Init*

The Initialization function *Init* is described as follows.

**INPUT:** 128- or 256-bit key  $K$ , 128-bit initialization vector  $IV$ .

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

1. Initialize the registers by the key information.

a. For a 128-bit key

$$(K_3, K_2, K_1, K_0) = K.$$

$$\text{For } j = 0, 1, 2, 3 \text{ set } a_{15-j}^{(-34)} = a_{15-j-8}^{(-34)} = K_{3-j}, \text{ and set } a_{15-j-4}^{(-34)} = a_{15-j-12}^{(-34)} = \neg K_{3-j}.$$