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)

## Contents

Page

Foreword ..... iv

Introduction ..... v

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 *I* (*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 *Init* ..... 14
7.1.2 Next-state function *Next* ..... 15
7.1.3 Keystream function *Strm* ..... 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 *Init* ..... 19
7.2.2 Next-state function *Next* ..... 20
7.2.3 Keystream function *Strm* ..... 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

Annex A (informative) Examples ..... 24
A.1 Operations over the finite field GF($2^n$) ..... 24
A.2 Example for MUGI ..... 24
A.2.1 Key, initialization vector, and keystream triplets ..... 24
A.2.2 Sample internal states ..... 24
A.3 Example for SNOW 2.0 ..... 30
A.3.1 128-bit key ..... 30
A.3.2 256-bit key ..... 34

**Annex B** (informative) **Security information** .......... 39
**B.1 Security levels of stream ciphers** .......... 39
**B.1.1 Security-efficiency trade-off in MULTI-S01** .......... 40
**B.2 Implementation examples of dedicated keystream generators** .......... 40

**Annex C** (normative) **Object identifiers** .......... 41

**Bibliography** .......... 43

## 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. |
| $b_i$ | 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. |
| OR | Bitwise or operation. |
| *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. |
| $\alpha_{\mathrm{MUL}}$ | Lookup table used for SNOW 2.0. |
| $\alpha_{\mathrm{inv\_MUL}}$ | 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. |
| $\|\|$ | 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 $A = \{a_{n-1}, a_{n-2}, ., a_0\}$ and $B = \{b_{n-1}, b_{n-2}, ..., b_0\}$ (where the $a_i$ and $b_i$ are bits) are represented as the polynomials, $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_0$ and $B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + ... + b_0$ 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 $C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ... + c_0$ (where the $c_i$ are bits) then let $C$ be the $n$-bit block $\{c_{n-1}, c_{n-2}, ..., c_0\}$. |
| $<<_n t$ | $t$-bit left shift in an $n$-bit register. |
| $>>_n t$ | $t$-bit right shift in an $n$-bit register. |
| $<<<_n t$ | $t$-bit left circular rotation in an $n$-bit register. |
| $>>>_n t$ | $t$-bit right circular rotation in an $n$-bit register. |

## 4.1 Left-truncation of bits

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

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

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 $1 \le k \le n$, the effect of the shift function *Shift* is to produce the $n$-bit variable

$$Shift_k(X \mid F) = (x_k, x_{k+1}, \ldots, x_{n-1}, f_0, f_1, \ldots, f_{k-1}) \qquad (k < n)$$

$$Shift_k(X \mid F) = (f_0, f_1, \ldots, f_{k-1}) \qquad (k = n)$$

The effect is to shift the bits of array $X$ left by $k$ places, discarding $x_0, x_1, \ldots, x_{k-1}$ 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

$$S_0 = Init(IV, K).$$

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

1. Output a keystream block $Z_i = Strm(S_i, K)$.

2. Update the state of the machine $S_{i+1} = Next(S_i, K)$.

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 $C_{-1}$, $C_{-2}$, …, $C_{-r}$.

2. A keystream function, *Strm*, that takes as input $S$ and $r$ ciphertext blocks $C_{i-1}$, $C_{i-2}$, …, $C_{i-r}$, 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:

$$C_i = Out(P_i, Z_i, R),$$

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

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

The output function shall be such that, for any keystream block $Z_i$, plaintext block $P_i$, and other input $R$, we have that

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

### 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

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

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

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

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

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

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 $(P_0, P_1, ..., P_{u-1}) = P$, where $P_i$ is an $n$-bit block.

3. Set $P_u = Z_{t+u+3}$.

4. Set $P_{u+1} = R$.

5. Set $W_{-1} = O^{(n)}$.

6. For $i = 0, 1,..., u+1$ do the following calculations:

   6.1. Let $W_i = P_i \oplus Z_{t+i+1}$.

   6.2. Let $X_i = Z_t \otimes W_i$ (in GF($2^n$)).

   6.3. Let $C_i = X_i \oplus W_{i-1}$.

7. Set $C = C_0 \| C_1 \| ... \| C_{u+1}$.

8 Output $C$.

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

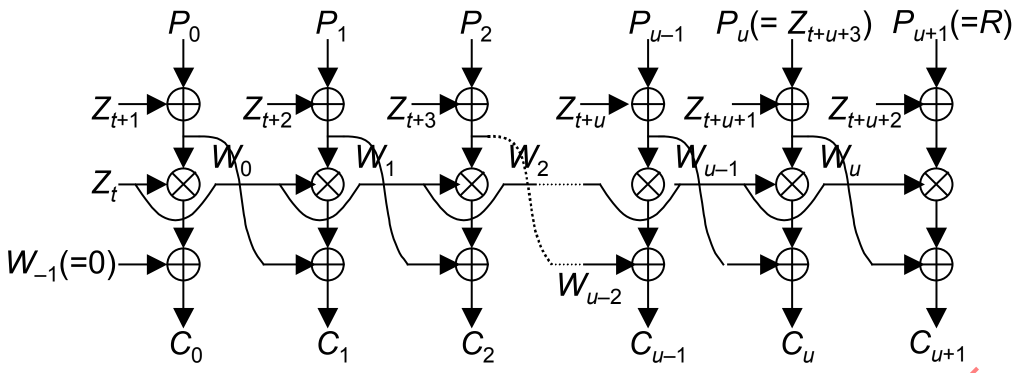

**Figure 1 — *Out* function of MULTI-S01 mode**

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 $\boldsymbol{Out^{-1}(C, Z, R)}$

Although the $Out^{-1}$ function does not output any text before the integrity check, the core computation of the $Out^{-1}$ 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:** $n{\cdot}v$-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 $(C_0, C_1, \ldots , C_{v-1}) = C$, where $C_i$ is an $n$-bit block.

3. For each $C_i$, do the following calculations (for $i = 0, 1, \ldots, v-1$):

   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 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 $P = P_0 \mid\mid P_1 \mid\mid \ldots \mid\mid P_{v-3}$ as plaintext. Otherwise output the special symbol meaning "*reject*" without any text.

Figure 2 shows the operation of the $Out^{-1}$ function.

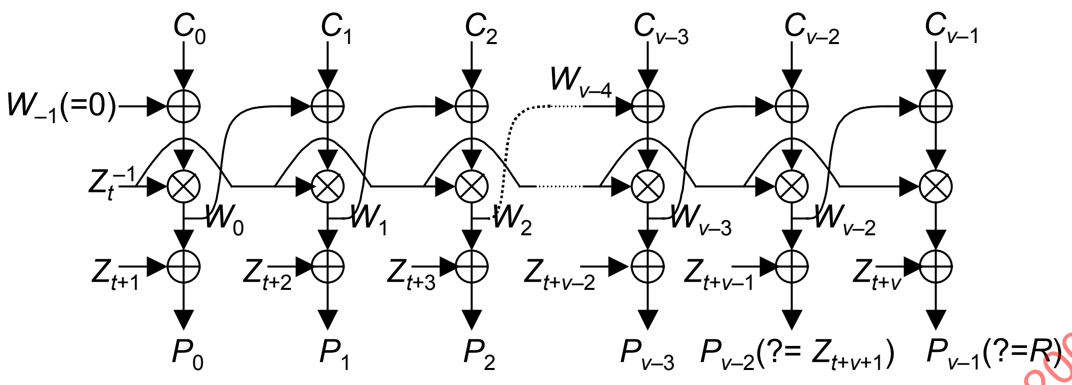

**Figure 2 — $Out^{-1}$ function of MULTI-S01 mode**

### 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:** $(n\,v + c)$-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 $(n\,v + n)$ 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.

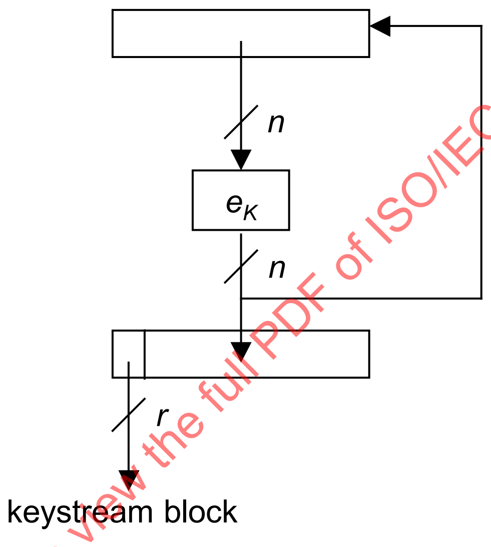

**Figure 3 — Keystream generation based on OFB mode**

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 $S_0 = IV$.

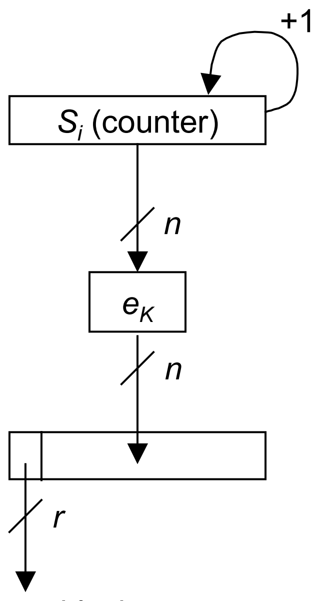

**Figure 4 — Keystream generation based on CTR mode**

## 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. $Next(S_i) = Shift_b(S_i | Shift_r(I(b)|C_i))$.

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 $S_0 = IV$.

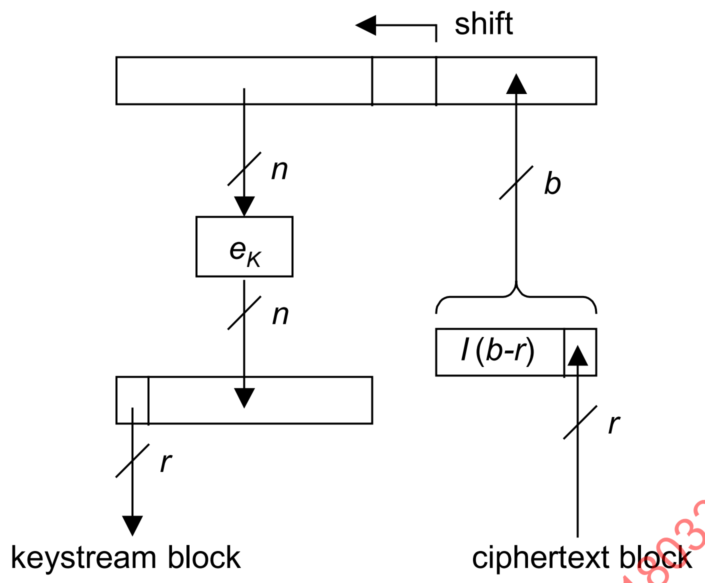

**Figure 5 — Keystream generation based on CFB mode**

## 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:

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

where $b_j{}^{(i)}$ is a block (for $j$ = 0, 1, …, 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:

$D_0$ = 0x6A09E667F3BCC908,

$D_1$ = 0xBB67AE8584CAA73B,

$D_2$ = 0x3C6EF372FE94F82B.

### 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 $a_2^{(-49)} = (K_0 <<<_{64} 7) \oplus (K_1 >>>_{64} 7) \oplus D_0$.

   $D_0$ in the above equation is a constant (see clause 7.1).

2. For $i$ = –49, –48, ..., –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, ..., 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 $IV_0 \,||\, IV_1 = IV$, 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 $a_2^{(-32)} = a_2^{(-33)} \oplus (IV_0 <<<_{64} 7) \oplus (IV_1 >>>_{64} 7) \oplus D_0$.

5. For $i$ = –32, –31, ..., –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 $S_0 = Next^{16}(S_{-16})$,

   where $Next^{16}$ 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 $a_2^{(i+1)} = a_0^{(i)} \oplus F(a_1^{(i)}, (w_2 <<<_{64} 17)) \oplus D_2$.

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.

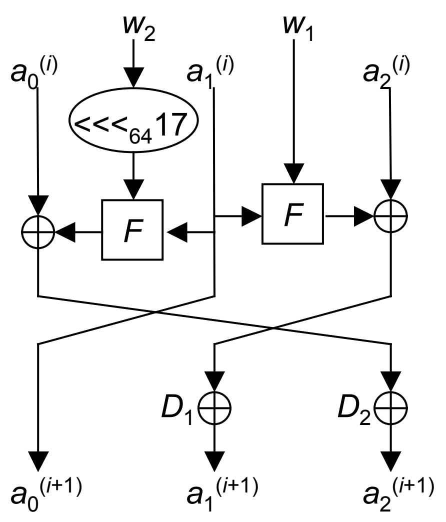

**Figure 6 — $\rho_1$ function of MUGI**

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 $b_{10}^{(i+1)} = b_9^{(i)} \oplus (b_{13}^{(i)} <<<_{64} 32)$.
5. Output $b^{(i+1)}$.

### 7.1.6 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*] / *m*(*x*), where *m*(*x*) is an irreducible polynomial of degree 8 defined over 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 $i = 0, 1, ..., 7$.
4. Set $P_L = P_0 \,||\, P_1 \,||\, P_2 \,||\, P_3$.
5. Set $P_R = P_4 \,||\, P_5 \,||\, P_6 \,||\, P_7$.
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 $Y = Q_4 \,||\, Q_5 \,||\, Q_2 \,||\, Q_3 \,||\, Q_0 \,||\, Q_1 \,||\, Q_6 \,||\, Q_7$.
11. Output $Y$.

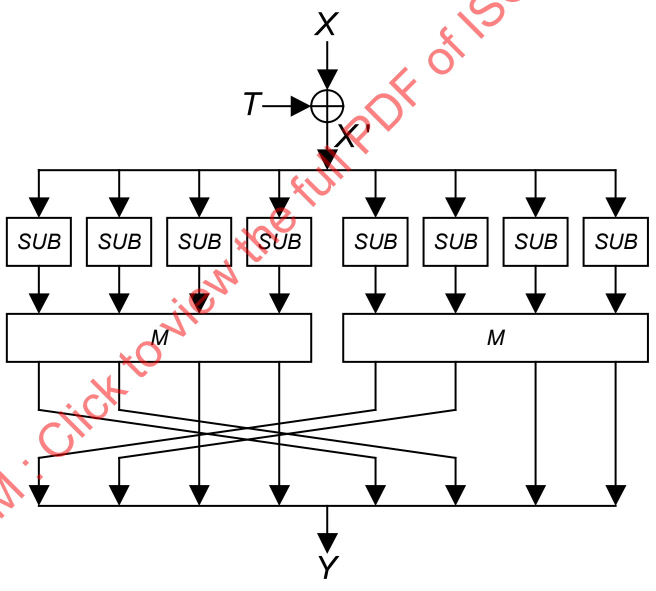

**Figure 7 — *F* function of MUGI**

### 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

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

   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:

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

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.

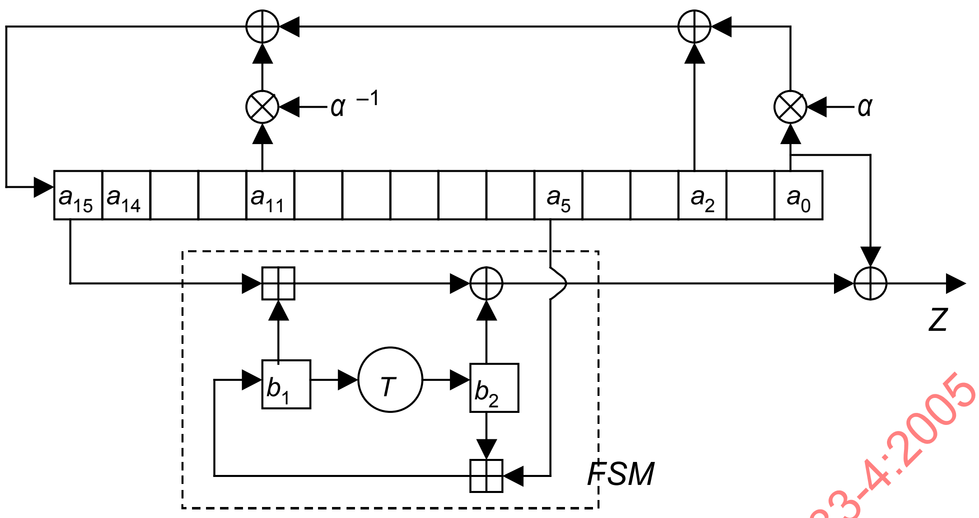

**Figure 8 — Schematic drawing of SNOW**

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

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

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

b. For a 256-bit key

$(K_7, K_6, \ldots, K_0) = K$.

For $j = 0, 1, \ldots, 7$ set $a_{15-j}^{(-34)} = K_{7-j}$, and set $a_{15-j-8}^{(-34)} = \neg(K_{7-j})$.

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

a. $(IV_3, IV_2, IV_1, IV_0) = IV$.

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

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

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

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

4. $S_0 = Next(S_{-1})$.

5. Output $S_0$.

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

1. Set $b_2^{(i+1)} = T(b_1^{(i)})$.
2. Set $b_1^{(i+1)} = b_2^{(i)} +_{32} a_5^{(i)}$.
3. For $j = 0, 1, \ldots, 14$ set $a_j^{(i+1)} = a_{j+1}^{(i)}$.
4. For INIT mode, set $a_{15}^{(i+1)} = (a_0^{(i)} \otimes \alpha) \oplus a_2^{(i)} \oplus (a_{11}^{(i)} \otimes \alpha^{-1}) \oplus FSM(a_{15}^{(i)}, b_1^{(i)}, b_2^{(i)})$,
   otherwise, set $a_{15}^{(i+1)} = (a_0^{(i)} \otimes \alpha) \oplus a_2^{(i)} \oplus (a_{11}^{(i)} \otimes \alpha^{-1})$.
5. $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.
6. Output $S_{i+1}$.

We refer to clauses 7.2.4 and 7.2.5, respectively, for description of the *T* function and the finite field arithmetic involving the fixed element $\alpha$.

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

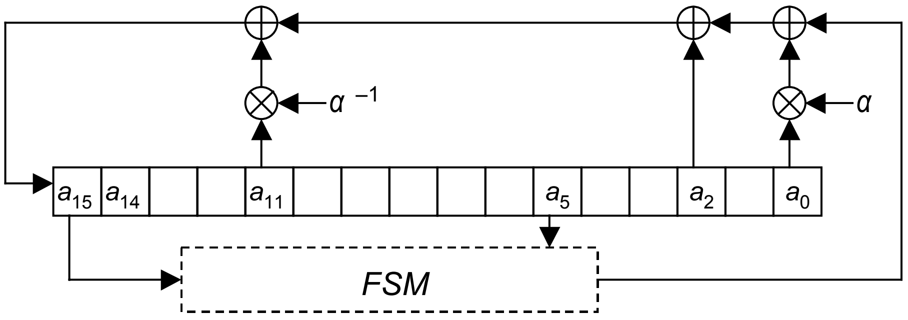

**Figure 9 — INIT mode of *Next* function.**

We refer to clause 7.2.7 for the definition of the *FSM* function.

### 7.2.3 Keystream function *Strm*

The keystream function *Strm* is described as follows:

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

**OUTPUT:** 32-bit keystream $Z_i$.

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

2. Output $Z_i$.

### 7.2.4 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, we make use of the finite field GF($2^8$), 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 $q = T(w)$.

1. Set $(w_3, w_2, w_1, w_0) = w$, where each $w_j$ is 8 bit.

2. For $j = 0, 1, 2, 3$ set $t_j = SUB\,[w_j]$.

3. 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: $t_j = t_{j,7} x^7 + \ldots + t_{j,1} x + t_{j,0}$ , $t_{j,k}$ in GF(2).

4. set $q(y) = c(y) \bullet t(y)$ modulo $(y^4 + 1)$, where $c(y) = (x+1)y^3 + y^2 + y + x$ in GF($2^8$)[*y*].

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

6. Output *q*.

Note that in step 3, 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 clause 7.1.7.

NOTE 2 – We refer to the paper [2] for details of this (and other) optimisation(s).

### 7.2.5 Multiplications of $\alpha$ 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($2^{32}$).

1. Set $w' = (w <<_{32} 8) \oplus \alpha_{MUL}[w >>_{32} 24]$.

2. Output $w'$.

The$\alpha_{MUL}$ is a table as follows:

```
αmul [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,0x9E6FDAAE,0xC2E04CD7,0x237F83C4,0xA9777BF1,0x48E8B4E2,
0x11C0FE03,0xF05F3110,0x7A57C925,0x9BC80636,0xC747904F,0x26D85F5C,0xACD0A769,0x4D4F687A,
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,
0x41F57694,0xA06AB987,0x2A6241B2,0xCBFD8EA1,0x977218D8,0x76EDD7CB,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,0xA13C6F45,0x40A3A056,0xCAAB5863,0x2B349770,
0x6C9CEE93,0x8D032180,0x070BD9B5,0xE69416A6,0xBA1B80DF,0x5B844FCC,0xD18CB7F9,0x301378EA,
0x693B320B,0x88A4FD18,0x02AC052D,0xE333CA3E,0xBFBC5C47,0x5E239354,0xD42B6B61,0x35B4A472,
0x667BFF0A,0x87E43019,0x0DECC82C,0xEC73073F,0xB0FC9146,0x51635E55,0xDB6BA660,0x3AF46973,
0x63DC2392,0x8243EC81,0x084B14B4,0xE9D4DBA7,0xB55B4DDE,0x54C482CD,0xDECC7AF8,0x3F53B5EB};
```

### 7.2.6 Multiplications of $\alpha^{-1}$ in finite field arithmetic

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

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

1. Set $y' = (y >>_{32} 8) \oplus \alpha_{\text{inv\_MUL}}[w \bmod 256]$.

2. Output $y'$.

The $\alpha_{\mathrm{inv\_MUL}}$ is a table as follows:

```
αinv_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,0xF4B0FC91,0xECBFBC5C,
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,0xC6C5EB95,0xDECAAB58,0x96E7C2C0,0x8EE8820D,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,
0xB3C4BD43,0xABCBFD8E,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,0x725DD940,0x3A70B0D8,0x227FF015,0x0A6E30EB,0x12617026,
0x451FD6E5,0x5D109628,0x750156D6,0x6D0E161B,0x25237F83,0x3D2C3F4E,0x153DFFB0,0x0D32BF7D,
0x85672D29,0x9D686DE4,0xB579AD1A,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,0x6E841EE3,0x768B5E2E,
0xFEDECC7A,0xE6D18CB7,0xCEC04C49,0xD6CF0C84,0x9EE2651C,0x86ED25D1,0xAEFCE52F,0xB6F3A5E2};
```

### 7.2.7 Function *FSM* (*x*, *y*, *z*)

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

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

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

2. Output *q*.

# Annex A
(informative)

# Examples

## A.1 Operations over the finite field GF($2^n$)

For any positive integer $n$ there exists a finite field containing exactly $2^n$ elements. This field is unique up to isomorphism and in this document it is referred to as the finite field GF($2^n$).

In the polynomial representation, each element of GF($2^n$) is represented by a binary polynomial of degree less than $n$. More explicitly the bit string $a = a_{n-1}\ldots a_2a_1a_0$ is taken to represent the binary polynomial $a(x) = a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0$. The polynomial basis is the set $B = \{x^{n-1}, \ldots, x^2, x, 1\}$. For two bit strings $a = a_{n-1}\ldots a_2a_1a_0$ and $b = b_{n-1}\ldots b_2b_1b_0$, the sum is $c = a \oplus b = c_{n-1}\ldots c_2c_1c_0$, where $c_i = a_i \oplus b_i$.

Multiplication in the finite field, written $a \otimes b$ corresponds to the multiplication of two polynomials $a(x)b(x)$ modulo a binary irreducible polynomial $p(x)$ of degree $n$. A polynomial is irreducible if it has no non-trivial divisors.

GF($2^n$}\{0} denoted as GF($2^n$)$^*$ is an abelian group with respect to multiplication and the identity is 1. For any non-zero binary polynomial $b(x)$ of degree less than $n$, the multiplicative inverse of $b(x)$, denoted $b^{-1}(x)$, can be computed as follows: the extended Euclidean algorithm is used to compute polynomials $a(x)$ and $c(x)$ such that $b(x)\bullet a(x) + p(x)\bullet c(x) = 1$. Hence, $a(x)\bullet b(x) \bmod p(x) = 1$, which means $b^{-1}(x) = a(x) \bmod p(x)$. The extended Euclidean algorithm is described in [4].

## A.2 Example for MUGI

### A.2.1 Key, initialization vector, and keystream triplets

```
K = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Z = c7 6e 14 e7 08 36 e6 b6 cb 0e 9c 5a 0b f0 3e 1e 0a cf 9a f4 9e be 6d 67 d5 72 6e 37 4b 13 97 ac.

K = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV= 34 61 69 88 51 81 21 39 01 55 00 a5 3b 7e 59 87
Z = 2a a1 c5 c7 20 73 b1 b3 a9 d1 0d c6 85 50 66 10 28 30 56 0d 9a 24 65 c9 9c 29 1c 13 81 4e 08 8d.

K = 51 34 00 b1 04 a0 59 91 30 ad 00 fc 48 d7 59 e0
IV= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Z = bd df ad 5f 04 b8 86 25 c3 ad ac e1 56 d1 c1 99 36 ff a4 e9 a7 fd f7 5a aa b8 29 13 42 85 aa 4b.

K = 69 e7 06 ee 52 95 37 2c 75 13 01 47 30 23 79 93
IV= 2a 00 45 c8 49 27 49 d5 3a 9b 16 4a 25 e4 49 15
Z = e3 cc 67 a0 25 5b 0f 28 2d 9a 5b 1b bd f7 f2 df 84 eb 46 f6 07 d6 e6 dd 32 86 13 43 94 dd 95 fb.
```

### A.2.2 Sample internal states

```
K = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
IV= f0 e0 d0 c0 b0 a0 90 80 70 60 50 40 30 20 10 00
Z = bc 62 43 06 14 b7 9b 71 71 a6 66 81 c3 55 42 de 7a ba 5b 4f b8 0e 82 d7 0b 96 98 28 90 b6 e1 43
```

```
Intermediate values of the internal state

rho function 0
a: 0001020304050607 08090a0b0c0d0e0f 7498f5f1e727d094
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000

rho function 1
a: 08090a0b0c0d0e0f 9724d9144c5d8926 64b47311d52100a5
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 08090a0b0c0d0e0f

rho function 2
a: 9724d9144c5d8926 09671cfbcfaa95fb e2d338166cd8c441
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 3
a: 09671cfbcfaa95fb 9c0c2097edb20067 6ef29c62b7691210
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 4
a: 9c0c2097edb20067 c08ee4dcb2d08591 201239b2b04d5d6a
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 5
a: c08ee4dcb2d08591 738177859f3210f6 48963357b89312eb
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 6
a: 738177859f3210f6 b36b4d944f5d04cb bc7ac7e83f40cca1
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 7
a: b36b4d944f5d04cb 2d13c00221057d8d 65e12d98fb29feca
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 8
a: 2d13c00221057d8d 20ead0479e63cdc3 7169edbc504968d2
```

```
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 0000000000000000
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 9
a: 20ead0479e63cdc3 591a6857e3112cee 8269181ee80366a1
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 0000000000000000 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 10
a: 591a6857e3112cee dfbbb88c02c9c80a fa312d220ef73c78
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 0000000000000000 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 11
a: dfbbb88c02c9c80a 5cc4835080bc5321 78e69bd217041ca7
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   0000000000000000 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 12
a: 5cc4835080bc5321 fd5755df9cc0ceb9 dd032b76f3534504
b: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 13
a: fd5755df9cc0ceb9 c905d08f50fa71db cfcb255e594b38ee
b: 0000000000000000 0000000000000000 0000000000000000 fd5755df9cc0ceb9
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 14
a: c905d08f50fa71db bfe2485ac2696cc7 0a77652c7dbcc580
b: 0000000000000000 0000000000000000 c905d08f50fa71db fd5755df9cc0ceb9
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

rho function 15
a: bfe2485ac2696cc7 7dea261cb61d4fea 3991ce48e105a4a1
b: 0000000000000000 bfe2485ac2696cc7 c905d08f50fa71db fd5755df9cc0ceb9
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

buffer init
a: 7dea261cb61d4fea eafb528479bb687d eb8189612089ff0b
b: 7dea261cb61d4fea bfe2485ac2696cc7 c905d08f50fa71db fd5755df9cc0ceb9
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f
```

```
rho function 0
a: 8d0af6dc06bddf6a 9a9b02c4499b787d f100cffe031d365b

rho function 1
a: 9a9b02c4499b787d 435407f3bbc2c760 b8576326c43c7141

rho function 2
a: 435407f3bbc2c760 b5117172dcf5e507 10d44d672b0cb32b

rho function 3
a: b5117172dcf5e507 9157292760b2892f 45de3e448a22a274

rho function 4
a: 9157292760b2892f aee0542493e7889e d92646e5bf6e90fd

rho function 5
a: aee0542493e7889e a9f2f7fac6cff1ff 668ac5cf634db73d

rho function 6
a: a9f2f7fac6cff1ff 9cb8969f9fc84dc6 d3db5a83153c2d75

rho function 7
a: 9cb8969f9fc84dc6 b1260b2ec980a340 4c06fba0602d20da

rho function 8
a: b1260b2ec980a340 192a6fd877969848 4e9d5f10f22daa44

rho function 9
a: 192a6fd877969848 bbac287d38601209 c31e21b47993441d

rho function 10
a: bbac287d38601209 d58486545129be34 88b995cf25723d71

rho function 11
a: d58486545129be34 c8af8f1422e98119 7cb36f5145a5f171

rho function 12
a: c8af8f1422e98119 00bba312081aa445 2e8517e066c8b410

rho function 13
a: 00bba312081aa445 2f3864a9c279a14c 4e1ba1aafc06cb55

rho function 14
a: 2f3864a9c279a14c 6551f5e9cbc1e0d7 acf8aaa64583d0d7

rho function 15
a: 6551f5e9cbc1e0d7 4e466dffcb92db48 4a8ffe073636f5c3

state init
a: 4e466dffcb92db48 f5eb67b928359d8b 5d3c31a0af9cd78f
b: 7dea261cb61d4fea bfe2485ac2696cc7 c905d08f50fa71db fd5755df9cc0ceb9
   5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee 20ead0479e63cdc3
   2d13c00221057d8d b36b4d944f5d04cb 738177859f3210f6 c08ee4dcb2d08591
   9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926 08090a0b0c0d0e0f

update 1
a: f5eb67b928359d8b ace6a90bde0af786 529108c358fa4ada
b: 464f67f4c79fd547 7dea261cb61d4fea bfe2485ac2696cc7 c905d08f50fa71db
   ddbd859802a3037a 5cc4835080bc5321 dfbbb88c02c9c80a 591a6857e3112cee
   20ead0479e63cdc3 2d13c00221057d8d 7cc1d86f463a1830 738177859f3210f6
   c08ee4dcb2d08591 9c0c2097edb20067 09671cfbcfaa95fb 9724d9144c5d8926
```

```
update 2
a: ace6a90bde0af786 9fa7a15367ae1667 5f241cf311a0bfa7
b: 62cfbead646814ad 464f67f4c79fd547 7dea261cb61d4fea bfe2485ac2696cc7
   901fb8d8b3eb5d35 ddbd859802a3037a 5cc4835080bc5321 dfbbb88c02c9c80a
   591a6857e3112cee 20ead0479e63cdc3 c0a1c065bd095d1a 7cc1d86f463a1830
   738177859f3210f6 c08ee4dcb2d08591 9c0c2097edb20067 09671cfbcfaa95fb

update 3
a: 9fa7a15367ae1667 75195c2e249e4399 8bd43dd671ad8b05
b: a581b5f011a0627d 62cfbead646814ad 464f67f4c79fd547 7dea261cb61d4fea
   6059f0d6c0a0a4cd 901fb8d8b3eb5d35 ddbd859802a3037a 5cc4835080bc5321
   dfbbb88c02c9c80a 591a6857e3112cee 923a55d65eed291f c0a1c065bd095d1a
   7cc1d86f463a1830 738177859f3210f6 c08ee4dcb2d08591 9c0c2097edb20067

update 4
a: 75195c2e249e4399 9b2239a28cc5a4e3 5554ab4e803a0a19
b: 03ab81c48a1c1600 a581b5f011a0627d 62cfbead646814ad 464f67f4c79fd547
   212ea54c36a11ccb 6059f0d6c0a0a4cd 901fb8d8b3eb5d35 ddbd859802a3037a
   5cc4835080bc5321 dfbbb88c02c9c80a c62878a190905b6b 923a55d65eed291f
   c0a1c065bd095d1a 7cc1d86f463a1830 738177859f3210f6 c08ee4dcb2d08591

update 5
a: 9b2239a28cc5a4e3 80c1d0bc18b29e62 4b4f363e4a322d0e
b: b597b8f2964ec608 03ab81c48a1c1600 a581b5f011a0627d 62cfbead646814ad
   9bf2e26cc53cd63d 212ea54c36a11ccb 6059f0d6c0a0a4cd 901fb8d8b3eb5d35
   ddbd859802a3037a 5cc4835080bc5321 9981a0bc7e081065 c62878a190905b6b
   923a55d65eed291f c0a1c065bd095d1a 7cc1d86f463a1830 738177859f3210f6

update 6
a: 80c1d0bc18b29e62 dfe2225186dfbca3 1277ae469887f627
b: e8a34e2713f7b415 b597b8f2964ec608 03ab81c48a1c1600 a581b5f011a0627d
   f2d00675d7834998 9bf2e26cc53cd63d 212ea54c36a11ccb 6059f0d6c0a0a4cd
   901fb8d8b3eb5d35 ddbd859802a3037a e1cdde4a401d9344 9981a0bc7e081065
   c62878a190905b6b 923a55d65eed291f c0a1c065bd095d1a 7cc1d86f463a1830

update 7
a: dfe2225186dfbca3 ca86cdc9d2bf2007 7a5c92de7be1811f
b: fc0008d35e888652 e8a34e2713f7b415 b597b8f2964ec608 03ab81c48a1c1600
   c5d84526d100c6b0 f2d00675d7834998 9bf2e26cc53cd63d 212ea54c36a11ccb
   6059f0d6c0a0a4cd 901fb8d8b3eb5d35 8350ac87909956ac e1cdde4a401d9344
   9981a0bc7e081065 c62878a190905b6b 923a55d65eed291f c0a1c065bd095d1a

update 8
a: ca86cdc9d2bf2007 0870fbf8e065b266 067f3c2be88481d4
b: 1f43e2343bd6e1b9 fc0008d35e888652 e8a34e2713f7b415 b597b8f2964ec608
   22852488bcbd0acb c5d84526d100c6b0 f2d00675d7834998 9bf2e26cc53cd63d
   212ea54c36a11ccb 6059f0d6c0a0a4cd 008fe3b375c32594 8350ac87909956ac
   e1cdde4a401d9344 9981a0bc7e081065 c62878a190905b6b 923a55d65eed291f

update 9
a: 0870fbf8e065b266 73b145404394710d d756724ed3994273
b: 58bc981f8c520918 1f43e2343bd6e1b9 fc0008d35e888652 e8a34e2713f7b415
   2e655a9e53721035 22852488bcbd0acb c5d84526d100c6b0 f2d00675d7834998
   9bf2e26cc53cd63d 212ea54c36a11ccb 1e51e0b359210471 008fe3b375c32594
   8350ac87909956ac e1cdde4a401d9344 9981a0bc7e081065 c62878a190905b6b
```

```
update 10
a: 73b145404394710d 82d164adcac96d62 0607785b7d152b8b
b: ce58835970f5e90d 58bc981f8c520918 1f43e2343bd6e1b9 fc0008d35e888652
   1a734852c474fd8d 2e655a9e53721035 22852488bcbd0acb c5d84526d100c6b0
   f2d00675d7834998 9bf2e26cc53cd63d 61333608d76cc281 1e51e0b359210471
   008fe3b375c32594 8350ac87909956ac e1cdde4a401d9344 9981a0bc7e081065

update 11
a: 82d164adcac96d62 c14072735c68e7e9 f61c61bbde49ed28
b: ea30e5fc3d9c6168 ce58835970f5e90d 58bc981f8c520918 1f43e2343bd6e1b9
   39d84df58f8840e2 1a734852c474fd8d 2e655a9e53721035 22852488bcbd0acb
   c5d84526d100c6b0 f2d00675d7834998 0b6bb4c0466c7aba 61333608d76cc281
   1e51e0b359210471 008fe3b375c32594 8350ac87909956ac e1cdde4a401d9344

update 12
a: c14072735c68e7e9 ce0bee4623950852 af052447a7444e65
b: 631cbae78ad4fe26 ea30e5fc3d9c6168 ce58835970f5e90d 58bc981f8c520918
   3dc6c6bc876beb72 39d84df58f8840e2 1a734852c474fd8d 2e655a9e53721035
   22852488bcbd0acb c5d84526d100c6b0 871323e1d70caa2b 0b6bb4c0466c7aba
   61333608d76cc281 1e51e0b359210471 008fe3b375c32594 8350ac87909956ac

update 13
a: ce0bee4623950852 f6c22506fc93fb5a 9eb296971244bcb3
b: 4210def4ccf1b145 631cbae78ad4fe26 ea30e5fc3d9c6168 ce58835970f5e90d
   76d9c281df20192d 3dc6c6bc876beb72 39d84df58f8840e2 1a734852c474fd8d
   2e655a9e53721035 22852488bcbd0acb 9cf94157cf512603 871323e1d70caa2b
   0b6bb4c0466c7aba 61333608d76cc281 1e51e0b359210471 008fe3b375c32594

update 14
a: f6c22506fc93fb5a b36f504b7eb67fe6 a66ba7dd058722d3
b: ce840df556562dc6 4210def4ccf1b145 631cbae78ad4fe26 ea30e5fc3d9c6168
   d42bcb0bb4811480 76d9c281df20192d 3dc6c6bc876beb72 39d84df58f8840e2
   1a734852c474fd8d 2e655a9e53721035 f5e9e609dd8e3cc3 9cf94157cf512603
   871323e1d70caa2b 0b6bb4c0466c7aba 61333608d76cc281 1e51e0b359210471

update 15
a: b36f504b7eb67fe6 0ce5a4d1a0cbc0f7 bd0c30563f8ee4f7
b: e893c5b5a5b2ff2b ce840df556562dc6 4210def4ccf1b145 631cbae78ad4fe26
   d3e8a809b214218a d42bcb0bb4811480 76d9c281df20192d 3dc6c6bc876beb72
   39d84df58f8840e2 1a734852c474fd8d 680920245819a4f5 f5e9e609dd8e3cc3
   9cf94157cf512603 871323e1d70caa2b 0b6bb4c0466c7aba 61333608d76cc281

update 16
a: 0ce5a4d1a0cbc0f7 316993816117e50f bc62430614b79b71
b: d25c6643a9dabd67 e893c5b5a5b2ff2b ce840df556562dc6 4210def4ccf1b145
   5eda7c5b0dbf1554 d3e8a809b214218a d42bcb0bb4811480 76d9c281df20192d
   3dc6c6bc876beb72 39d84df58f8840e2 cd7fe2794367de6c 680920245819a4f5
   f5e9e609dd8e3cc3 9cf94157cf512603 871323e1d70caa2b 0b6bb4c0466c7aba

update 1
a: 316993816117e50f 4f7c747ce422e686 71a66681c35542de
b: 078e1011e6a7ba4d d25c6643a9dabd67 e893c5b5a5b2ff2b ce840df556562dc6
   34c91c7513d1a868 5eda7c5b0dbf1554 d3e8a809b214218a d42bcb0bb4811480
   76d9c281df20192d 3dc6c6bc876beb72 f6896bf6137101b5 cd7fe2794367de6c
   680920245819a4f5 f5e9e609dd8e3cc3 9cf94157cf512603 871323e1d70caa2b

update 2
a: 4f7c747ce422e686 0aeab5f525c1a62f 7aba5b4fb80e82d7
b: b67ab060b61b4f24 078e1011e6a7ba4d d25c6643a9dabd67 e893c5b5a5b2ff2b
   1aafc6fee2d73946 34c91c7513d1a868 5eda7c5b0dbf1554 d3e8a809b214218a
   d42bcb0bb4811480 76d9c281df20192d e048fa7f72820d7b f6896bf6137101b5
   cd7fe2794367de6c 680920245819a4f5 f5e9e609dd8e3cc3 9cf94157cf512603
```

```
update 3
a: 0aeab5f525c1a62f bd1a2938a57319c8 0b96982890b6e143
b: d385352b2b73c085 b67ab060b61b4f24 078e1011e6a7ba4d d25c6643a9dabd67
   3b7b6dbc17a6dea1 1aafc6fee2d73946 34c91c7513d1a868 5eda7c5b0dbf1554
   d3e8a809b214218a d42bcb0bb4811480 2ec06674b7293909 e048fa7f72820d7b
   f6896bf6137101b5 cd7fe2794367de6c 680920245819a4f5 f5e9e609dd8e3cc3

update 4
a: bd1a2938a57319c8 e4684a2bf28ff50d 4930b5d033157f46
b: ff0353fcf84f9aec d385352b2b73c085 b67ab060b61b4f24 078e1011e6a7ba4d
   8c861a18a465a833 3b7b6dbc17a6dea1 1aafc6fee2d73946 34c91c7513d1a868
   5eda7c5b0dbf1554 d3e8a809b214218a 974c156779fef6f9 2ec06674b7293909
   e048fa7f72820d7b f6896bf6137101b5 cd7fe2794367de6c 680920245819a4f5
```

## A.3 Example for SNOW 2.0

### A.3.1 128-bit key

#### A.3.1.1 Key, initialization vector, and keystream triplets

$(IV_3,IV_2,IV_1,IV_0)$ = (0, 0, 0, 0), key = 80000000000000000000000000000000
Keystream output: 8D590AE9A74A7D056DC9CA74B72D1A4599B0A083FB45D13FCF9411BD9A503783.

$(IV_3,IV_2,IV_1,IV_0)$ = (0, 0, 0, 0), key = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Keystream output: E00982F525F02054214992D8706F2B20DA585E5B85E2746D09F22681B2749407.

$(IV_3,IV_2,IV_1,IV_0)$ = (4,3,2,1), key = 80000000000000000000000000000000
Keystream output: D6403358E0354A6957F43FCE44B4B13FF78E24C246618A0767AC83C10BFC45F0.

$(IV_3,IV_2,IV_1,IV_0)$ = (4,3,2,1), key = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Keystream output: C355385DB31D6CBDF774AF5366C2E8774DEADAC7DC7229DFED171D7B.

#### A.3.1.2 Sample internal states

```
K = 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV= 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 01
Z = d6 40 33 58 e0 35 4a 69 57 f4 3f ce 44 b4 b1 3f f7 8e 24 c2 46 61 8a 07 67 ac 83 c1 0b fc 45 f0

Snow 2.0 Internal state at time -34
15:80000000 14:00000000 13:00000000 12:00000000 11:7fffffff 10:ffffffff 09:ffffffff 08:ffffffff
07:80000000 06:00000000 05:00000000 04:00000000 03:7fffffff 02:ffffffff 01:ffffffff 00:ffffffff

Snow 2.0 Internal state at time -33
15:80000001 14:00000000 13:00000000 12:00000002 11:7fffffff 10:fffffffc 09:fffffffb 08:ffffffff
07:80000000 06:00000000 05:00000000 04:00000000 03:7fffffff 02:ffffffff 01:ffffffff 00:ffffffff
R1:00000000 R2:00000000

Snow 2.0 Internal state at time -32
15:09dfef08 14:80000001 13:00000000 12:00000000 11:00000002 10:7fffffff 09:fffffffc 08:fffffffb
07:ffffffff 06:80000000 05:00000000 04:00000000 03:00000000 02:7fffffff 01:ffffffff 00:ffffffff
R1:00000000 R2:63636363

Snow 2.0 Internal state at time -31
15:e5f1b94c 14:09dfef08 13:80000001 12:00000000 11:00000000 10:00000002 09:7fffffff 08:fffffffc
07:fffffffb 06:ffffffff 05:80000000 04:00000000 03:00000000 02:00000000 01:7fffffff 00:ffffffff
R1:63636363 R2:63636363
```

```
Snow 2.0 Internal state at time -30
15:ea9a3527 14:e5f1b94c 13:09dfef08 12:80000001 11:00000000 10:00000000 09:00000002 08:7fffffff
07:fffffffc 06:fffffffb 05:ffffffff 04:80000000 03:00000000 02:00000000 01:00000000 00:7fffffff
R1:e3636363 R2:fbfbfbfb

Snow 2.0 Internal state at time -29
15:a69fa10d 14:ea9a3527 13:e5f1b94c 12:09dfef08 11:80000001 10:00000000 09:00000000 08:00000002
07:7fffffff 06:fffffffc 05:fffffffb 04:ffffffff 03:80000000 02:00000000 01:00000000 00:00000000
R1:fbfbfbfa R2:34de1111

Snow 2.0 Internal state at time -28
15:8ecaccdb 14:a69fa10d 13:ea9a3527 12:e5f1b94c 11:09dfef08 10:80000001 09:00000000 08:00000000
07:00000002 06:7fffffff 05:fffffffc 04:fffffffb 03:ffffffff 02:80000000 01:00000000 00:00000000
R1:34de110c R2:692d2d4b

Snow 2.0 Internal state at time -27
15:eaf4d48f 14:8ecaccdb 13:a69fa10d 12:ea9a3527 11:e5f1b94c 10:09dfef08 09:80000001 08:00000000
07:00000000 06:00000002 05:7fffffff 04:fffffffc 03:fffffffb 02:ffffffff 01:80000000 00:00000000
R1:692d2d47 R2:b66ede7f

Snow 2.0 Internal state at time -26
15:19805681 14:eaf4d48f 13:8ecaccdb 12:a69fa10d 11:ea9a3527 10:e5f1b94c 09:09dfef08 08:80000001
07:00000000 06:00000000 05:00000002 04:7fffffff 03:fffffffc 02:fffffffb 01:ffffffff 00:80000000
R1:366ede7e R2:12c38109

Snow 2.0 Internal state at time -25
15:e8688f55 14:19805681 13:eaf4d48f 12:8ecaccdb 11:a69fa10d 10:ea9a3527 09:e5f1b94c 08:09dfef08
07:80000001 06:00000000 05:00000000 04:00000002 03:7fffffff 02:fffffffc 01:fffffffb 00:ffffffff
R1:12c3810b R2:86c47640

Snow 2.0 Internal state at time -24
15:fa565ef1 14:e8688f55 13:19805681 12:eaf4d48f 11:8ecaccdb 10:a69fa10d 09:ea9a3527 08:e5f1b94c
07:09dfef08 06:80000001 05:00000000 04:00000000 03:00000002 02:7fffffff 01:fffffffc 00:fffffffb
R1:86c47640 R2:d63b88a5

Snow 2.0 Internal state at time -23
15:6c7a94aa 14:fa565ef1 13:e8688f55 12:19805681 11:eaf4d48f 10:8ecaccdb 09:a69fa10d 08:ea9a3527
07:e5f1b94c 06:09dfef08 05:80000001 04:00000000 03:00000000 02:00000002 01:7fffffff 00:fffffffc
R1:d63b88a5 R2:b7c51902

Snow 2.0 Internal state at time -22
15:5ced2805 14:6c7a94aa 13:fa565ef1 12:e8688f55 11:19805681 10:eaf4d48f 09:8ecaccdb 08:a69fa10d
07:ea9a3527 06:e5f1b94c 05:09dfef08 04:80000001 03:00000000 02:00000000 01:00000002 00:7fffffff
R1:37c51903 R2:db1c5e4f

Snow 2.0 Internal state at time -21
15:26ac1e81 14:5ced2805 13:6c7a94aa 12:fa565ef1 11:e8688f55 10:19805681 09:eaf4d48f 08:8ecaccdb
07:a69fa10d 06:ea9a3527 05:e5f1b94c 04:09dfef08 03:80000001 02:00000000 01:00000000 00:00000002
R1:e4fc4d57 R2:d04da3ad

Snow 2.0 Internal state at time -20
15:2e5cc1a4 14:26ac1e81 13:5ced2805 12:6c7a94aa 11:fa565ef1 10:e8688f55 09:19805681 08:eaf4d48f
07:8ecaccdb 06:a69fa10d 05:ea9a3527 04:e5f1b94c 03:09dfef08 02:80000001 01:00000000 00:00000000
R1:b63f5cf9 R2:6c782451
```

```
Snow 2.0 Internal state at time -19
15:2eb71be8 14:2e5cc1a4 13:26ac1e81 12:5ced2805 11:6c7a94aa 10:fa565ef1 09:e8688f55 08:19805681
07:eaf4d48f 06:8ecaccdb 05:a69fa10d 04:ea9a3527 03:e5f1b94c 02:09dfef08 01:80000001 00:00000000
R1:57125978 R2:13ebdccc

Snow 2.0 Internal state at time -18
15:dc33fa8c 14:2eb71be8 13:2e5cc1a4 12:26ac1e81 11:5ced2805 10:6c7a94aa 09:fa565ef1 08:e8688f55
07:19805681 06:eaf4d48f 05:8ecaccdb 04:a69fa10d 03:ea9a3527 02:e5f1b94c 01:09dfef08 00:80000001
R1:ba8b7dd9 R2:6b132ab7

Snow 2.0 Internal state at time -17
15:3007668a 14:dc33fa8c 13:2eb71be8 12:2e5cc1a4 11:26ac1e81 10:5ced2805 09:6c7a94aa 08:fa565ef1
07:e8688f55 06:19805681 05:eaf4d48f 04:8ecaccdb 03:a69fa10d 02:ea9a3527 01:e5f1b94c 00:09dfef08
R1:f9ddf792 R2:6eb763b9

Snow 2.0 Internal state at time -16
15:6fbbfcfb 14:3007668a 13:dc33fa8c 12:2eb71be8 11:2e5cc1a4 10:26ac1e81 09:5ced2805 08:6c7a94aa
07:fa565ef1 06:e8688f55 05:19805681 04:eaf4d48f 03:8ecaccdb 02:a69fa10d 01:ea9a3527 00:e5f1b94c
R1:59ac3848 R2:510e5e7e

Snow 2.0 Internal state at time -15
15:47128118 14:6fbbfcfb 13:3007668a 12:dc33fa8c 11:2eb71be8 10:2e5cc1a4 09:26ac1e81 08:5ced2805
07:6c7a94aa 06:fa565ef1 05:e8688f55 04:19805681 03:eaf4d48f 02:8ecaccdb 01:a69fa10d 00:ea9a3527
R1:6a8eb4ff R2:ed2a3ff7

Snow 2.0 Internal state at time -14
15:9dd8c346 14:47128118 13:6fbbfcfb 12:3007668a 11:dc33fa8c 10:2eb71be8 09:2e5cc1a4 08:26ac1e81
07:5ced2805 06:6c7a94aa 05:fa565ef1 04:e8688f55 03:19805681 02:eaf4d48f 01:8ecaccdb 00:a69fa10d
R1:d592cf4c R2:aaaf3ebb

Snow 2.0 Internal state at time -13
15:14c6e4b1 14:9dd8c346 13:47128118 12:6fbbfcfb 11:3007668a 10:dc33fa8c 09:2eb71be8 08:2e5cc1a4
07:26ac1e81 06:5ced2805 05:6c7a94aa 04:fa565ef1 03:e8688f55 02:19805681 01:eaf4d48f 00:8ecaccdb
R1:a5059dac R2:b838f49b

Snow 2.0 Internal state at time -12
15:2be1899a 14:14c6e4b1 13:9dd8c346 12:47128118 11:6fbbfcfb 10:3007668a 09:dc33fa8c 08:2eb71be8
07:2e5cc1a4 06:26ac1e81 05:5ced2805 04:6c7a94aa 03:fa565ef1 02:e8688f55 01:19805681 00:eaf4d48f
R1:24b38945 R2:911396b6

Snow 2.0 Internal state at time -11
15:09363669 14:2be1899a 13:14c6e4b1 12:9dd8c346 11:47128118 10:6fbbfcfb 09:3007668a 08:dc33fa8c
07:2eb71be8 06:2e5cc1a4 05:26ac1e81 04:5ced2805 03:6c7a94aa 02:fa565ef1 01:e8688f55 00:19805681
R1:ee00bebb R2:1449ba75

Snow 2.0 Internal state at time -10
15:9e6f24ce 14:09363669 13:2be1899a 12:14c6e4b1 11:9dd8c346 10:47128118 09:6fbbfcfb 08:3007668a
07:dc33fa8c 06:2eb71be8 05:2e5cc1a4 04:26ac1e81 03:5ced2805 02:6c7a94aa 01:fa565ef1 00:e8688f55
R1:3af5d8f6 R2:b8fa206d

Snow 2.0 Internal state at time -9
15:f87d0a5a 14:9e6f24ce 13:09363669 12:2be1899a 11:14c6e4b1 10:9dd8c346 09:47128118 08:6fbbfcfb
07:3007668a 06:dc33fa8c 05:2eb71be8 04:2e5cc1a4 03:26ac1e81 02:5ced2805 01:6c7a94aa 00:fa565ef1
R1:e756e211 R2:5a6f3141

Snow 2.0 Internal state at time -8
15:056b74c0 14:f87d0a5a 13:9e6f24ce 12:09363669 11:2be1899a 10:14c6e4b1 09:9dd8c346 08:47128118
07:6fbbfcfb 06:3007668a 05:dc33fa8c 04:2eb71be8 03:2e5cc1a4 02:26ac1e81 01:5ced2805 00:6c7a94aa
R1:89264d29 R2:87c4f589
```

```
Snow 2.0 Internal state at time -7
15:82d49257 14:056b74c0 13:f87d0a5a 12:9e6f24ce 11:09363669 10:2be1899a 09:14c6e4b1 08:9dd8c346
07:47128118 06:6fbbfcfb 05:3007668a 04:dc33fa8c 03:2eb71be8 02:2e5cc1a4 01:26ac1e81 00:5ced2805
R1:63f8f015 R2:b541dd3f

Snow 2.0 Internal state at time -6
15:4f549ab4 14:82d49257 13:056b74c0 12:f87d0a5a 11:9e6f24ce 10:09363669 09:2be1899a 08:14c6e4b1
07:9dd8c346 06:47128118 05:6fbbfcfb 04:3007668a 03:dc33fa8c 02:2eb71be8 01:2e5cc1a4 00:26ac1e81
R1:e54943c9 R2:cb416287

Snow 2.0 Internal state at time -5
15:01d936bb 14:4f549ab4 13:82d49257 12:056b74c0 11:f87d0a5a 10:9e6f24ce 09:09363669 08:2be1899a
07:14c6e4b1 06:9dd8c346 05:47128118 04:6fbbfcfb 03:3007668a 02:dc33fa8c 01:2eb71be8 00:2e5cc1a4
R1:3afd5f82 R2:f4c17d6d

Snow 2.0 Internal state at time -4
15:99c99eb5 14:01d936bb 13:4f549ab4 12:82d49257 11:056b74c0 10:f87d0a5a 09:9e6f24ce 08:09363669
07:2be1899a 06:14c6e4b1 05:9dd8c346 04:47128118 03:6fbbfcfb 02:3007668a 01:dc33fa8c 00:2eb71be8
R1:3bd3fe85 R2:b5efeab8

Snow 2.0 Internal state at time -3
15:0ea4e3f0 14:99c99eb5 13:01d936bb 12:4f549ab4 11:82d49257 10:056b74c0 09:f87d0a5a 08:9e6f24ce
07:09363669 06:2be1899a 05:14c6e4b1 04:9dd8c346 03:47128118 02:6fbbfcfb 01:3007668a 00:dc33fa8c
R1:53c8adfe R2:a0ddb267

Snow 2.0 Internal state at time -2
15:fa000bc8 14:0ea4e3f0 13:99c99eb5 12:01d936bb 11:4f549ab4 10:82d49257 09:056b74c0 08:f87d0a5a
07:9e6f24ce 06:09363669 05:2be1899a 04:14c6e4b1 03:9dd8c346 02:47128118 01:6fbbfcfb 00:3007668a
R1:b5a49718 R2:6ac944cc

Snow 2.0 Internal state at time -1
15:61dec1b8 14:fa000bc8 13:0ea4e3f0 12:99c99eb5 11:01d936bb 10:4f549ab4 09:82d49257 08:056b74c0
07:f87d0a5a 06:9e6f24ce 05:09363669 04:2be1899a 03:14c6e4b1 02:9dd8c346 01:47128118 00:6fbbfcfb
R1:96aace66 R2:9cd3a85e

Snow 2.0 Internal state at time 0
15:31f914d5 14:61dec1b8 13:fa000bc8 12:0ea4e3f0 11:99c99eb5 10:01d936bb 09:4f549ab4 08:82d49257
07:056b74c0 06:f87d0a5a 05:9e6f24ce 04:09363669 03:2be1899a 02:14c6e4b1 01:9dd8c346 00:47128118
R1:a609dec7 R2:495041dc

Snow 2.0 Internal state at time 1
15:9098ec10 14:31f914d5 13:61dec1b8 12:fa000bc8 11:0ea4e3f0 10:99c99eb5 09:01d936bb 08:4f549ab4
07:82d49257 06:056b74c0 05:f87d0a5a 04:9e6f24ce 03:09363669 02:2be1899a 01:14c6e4b1 00:9dd8c346
R1:e7bf66aa R2:05b5db95

Snow 2.0 Internal state at time 2
15:a5e7b806 14:9098ec10 13:31f914d5 12:61dec1b8 11:fa000bc8 10:0ea4e3f0 09:99c99eb5 08:01d936bb
07:4f549ab4 06:82d49257 05:056b74c0 04:f87d0a5a 03:9e6f24ce 02:09363669 01:2be1899a 00:14c6e4b1
R1:fe32e5ef R2:e728468a

Snow 2.0 Internal state at time 3
15:962fd59e 14:a5e7b806 13:9098ec10 12:31f914d5 11:61dec1b8 10:fa000bc8 09:0ea4e3f0 08:99c99eb5
07:01d936bb 06:4f549ab4 05:82d49257 04:056b74c0 03:f87d0a5a 02:9e6f24ce 01:09363669 00:2be1899a
R1:ec93bb4a R2:ed96a84d

Snow 2.0 Internal state at time 4
15:be037f87 14:962fd59e 13:a5e7b806 12:9098ec10 11:31f914d5 10:61dec1b8 09:fa000bc8 08:0ea4e3f0
07:99c99eb5 06:01d936bb 05:4f549ab4 04:82d49257 03:056b74c0 02:f87d0a5a 01:9e6f24ce 00:09363669
R1:706b3aa4 R2:d0d6a880
```

```
Snow 2.0 Internal state at time 5
15:3e9ee9ba 14:be037f87 13:962fd59e 12:a5e7b806 11:9098ec10 10:31f914d5 09:61dec1b8 08:fa000bc8
07:0ea4e3f0 06:99c99eb5 05:01d936bb 04:4f549ab4 03:82d49257 02:056b74c0 01:f87d0a5a 00:9e6f24ce
R1:202b4334 R2:86c48227

Snow 2.0 Internal state at time 6
15:a14a61e1 14:3e9ee9ba 13:be037f87 12:962fd59e 11:a5e7b806 10:9098ec10 09:31f914d5 08:61dec1b8
07:fa000bc8 06:0ea4e3f0 05:99c99eb5 04:01d936bb 03:4f549ab4 02:82d49257 01:056b74c0 00:f87d0a5a
R1:889db8e2 R2:b6399358

Snow 2.0 Internal state at time 7
15:cc852528 14:a14a61e1 13:3e9ee9ba 12:be037f87 11:962fd59e 10:a5e7b806 09:9098ec10 08:31f914d5
07:61dec1b8 06:fa000bc8 05:0ea4e3f0 04:99c99eb5 03:01d936bb 02:4f549ab4 01:82d49257 00:056b74c0
R1:5003320d R2:121f6605

Snow 2.0 Internal state at time 8
15:4b895ab7 14:cc852528 13:a14a61e1 12:3e9ee9ba 11:be037f87 10:962fd59e 09:a5e7b806 08:9098ec10
07:31f914d5 06:61dec1b8 05:fa000bc8 04:0ea4e3f0 03:99c99eb5 02:01d936bb 01:4f549ab4 00:82d49257
R1:20c449f5 R2:9cf74ff8
```

### A.3.2 256-bit key

#### A.3.2.1 Key, initialization vector, and keystream triplets

$(IV_3,IV_2,IV_1,IV_0) = (0,0,0,0)$,
key = 8000000000000000000000000000000000000000000000000000000000000000
Keystream output: 0B5BCCE20323E28E0FC203809C66AB73CA35A680F2A5DD197E0C5C02287BE822.

$(IV_3,IV_2,IV_1,IV_0) = (0,0,0,0)$,
key = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Keystream output: D9CC22FD861492D0AE6F43FB0F072012078C5AEEE479DE8CF0E555F458EED858.

$(IV_3,IV_2,IV_1,IV_0) = (4,3,2,1)$,
key = 8000000000000000000000000000000000000000000000000000000000000000
Keystream output: 7861080D5755E90B736F10916ED519B12C1A3A4255297FC2246AB7FA6C089526.

$(IV_3,IV_2,IV_1,IV_0) = (4,3,2,1)$,
key = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Keystream output: 29261FCE5ED038201D6AFAF8B87E74FED49ECB10197EAC025D024EB45E0C7655.

#### A.3.2.2 Sample internal state

```
K =  80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV= 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 01
Z0= 78 61 08 0d 57 55 e9 0b 73 6f 10 91 6e d5 19 b1 2c 1a 3a 42 55 29 7f c2 24 6a b7 fa 6c 08 95 26

Snow 2.0 Internal state at time -34
15:80000000 14:00000000 13:00000000 12:00000000 11:00000000 10:00000000 09:00000000 08:00000000
07:7fffffff 06:ffffffff 05:ffffffff 04:ffffffff 03:ffffffff 02:ffffffff 01:ffffffff 00:ffffffff
R1:0804a9cc R2:00000001

Snow 2.0 Internal state at time -33
15:80000001 14:00000000 13:00000000 12:00000002 11:00000000 10:00000003 09:00000004 08:00000000
07:7fffffff 06:ffffffff 05:ffffffff 04:ffffffff 03:ffffffff 02:ffffffff 01:ffffffff 00:ffffffff
R1:00000000 R2:00000000
```