ETSI/SAGE **Version: 1.1**
Specification **Date: 6th September 2006**

Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification

The SNOW 3G algorithm is the core of the standardised 3GPP Confidentiality and Integrity algorithms UEA2 & UIA2.

Document History		
V1.0	**10th January 2006**	**Publication**
V1.1	**6th September 2006**	**No change to the algorithm specification at all, just removal of an unwanted page header**

PREFACE

This specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the 3GPP Algorithm **SNOW 3G**. **SNOW 3G** is a stream cipher that forms the heart of the 3GPP confidentiality algorithm ***UEA2*** and the 3GPP integrity algorithm ***UIA2***.

This document is the second of four, which between them form the entire specification of 3GPP Confidentiality and Integrity Algorithms:

- Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***.
  Document 1: ***UEA2*** and ***UIA2*** Algorithm Specifications.

- Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***.
  Document 2: **SNOW 3G** Algorithm Specification.

- Specification of the 3GPP Encryption and Confidentiality Algorithms ***UEA2*** & ***UIA2***.
  Document 3: Implementors' Test Data.

- Specification of the 3GPP Encryption and Confidentiality Algorithms ***UEA2*** & ***UIA2***.
  Document 4: Design Conformance Test Data.

The normative part of the specification of **SNOW 3G** is in the main body of this document. The annexes to this document are purely informative. Annex 1 contains remarks about the mathematical background of some functions of **SNOW 3G**. Annex 2 contains implementation options for some functions of **SNOW 3G**. Annex 3 contains illustrations of functional elements of the algorithm, while Annex 4 contains an implementation program listing of the cryptographic algorithm specified in the main body of this document, written in the programming language C.

Similarly the normative part of the specification of the ***UEA2*** (confidentiality) and the ***UIA2*** (integrity) algorithms is in the main body of Document 1. The annexes of those documents and Documents 3 and 4 above, are purely informative.

Blank Page

TABLE OF CONTENTS

REFERENCES

[1] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (3G TS 33.102 version 6.3.0).

[2] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements; (3G TS 33.105 version 6.0.0).

[3] Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***. Document 1: ***UEA2*** and ***UIA2*** specifications.

[4] Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***. Document 2: **SNOW 3G** specification.

[5] Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***. Document 3: Implementors' Test Data.

[6] Specification of the 3GPP Confidentiality and Integrity Algorithms ***UEA2*** & ***UIA2***. Document 4: Design Conformance Test Data.

[7] P. Ekdahl and T. Johansson, "A new version of the stream cipher SNOW", in Selected Areas in Cryptology (SAC 2002), LNCS 2595, pp. 47–61, Springer-Verlag.

[8] J. Daemen, V. Rijmen, "The design of Rijndael", Springer Verlag Series on Information Security and Cryptography, Springer Verlag, 2002, ISBN 3-540-42580-2.

NORMATIVE SECTION

This part of the document contains the normative specification of the **SNOW 3G** algorithm.

1. Outline of the Normative Part of the Document

Section 2 introduces the algorithm and describes the notation used in the subsequent sections.

Section 3 defines the basic components of the algorithm.

Section 4 defines the operation of **SNOW 3G**.

Section 5 contains tables defining functions used in **SNOW 3G**.

2. Introductory Information

2.1. Introduction

Within the security architecture of the 3GPP system there are standardised algorithms: A confidentiality algorithm ***UEA2***, and an integrity algorithm ***UIA2***. These algorithms are fully specified in a companion document. Each of these algorithms is based on the **SNOW 3G** algorithm that is specified here.

SNOW 3G is a word-oriented stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialisation variable. These words can be used to mask the plaintext. First a key initialisation is performed, i.e. the cipher is clocked without producing output, see 4.1. Then with every clock tick it produces a 32-bit word of output, see 4.2.

2.2. Notation

2.2.1. Radix

We use the prefix **0x** to indicate **hexadecimal** numbers.

2.2.2. Bit ordering

All data variables in this specification are presented with the most significant bit on the left hand side and the least significant bit on the right hand side. Where a variable is broken down into a number of sub-strings, the left most (most significant) sub-string is numbered 0, the next most significant is numbered 1 and so on through to the least significant.

For example if a 64-bit value ***X*** is subdivided into four 16-bit substrings ***P, Q, R, S*** we have:

$$X = 0x0123456789ABCDEF$$

with

$$P = 0x0123,\ Q = 0x4567,\ R = 0x89AB,\ S = 0xCDEF.$$

In binary this would be:

$X = 0000000100100011010001010110011110001001101010111100110111101111$

with $P = 0000000100100011$

$\boldsymbol{Q}$ = 0100010101100111

$\boldsymbol{R}$ = 1000100110101011

$\boldsymbol{S}$ = 1100110111101111

2.2.3. Conventions

We use the assignment operator '=', as used in several programming languages. When we write

<variable> = <expression>

we mean that <variable> assumes the value that <expression> had before the assignment took place. For instance,

$$x = x + y + 3$$

means

(new value of x) becomes (old value of x) + (old value of y) + 3.

2.2.4. List of Symbols

= The assignment operator.

$\oplus$ The bitwise exclusive-OR operation.

$\boxplus$ Integer addition modulo 2^{32}.

|| The concatenation of the two operands.

$<<_n t$ t-bit left shift in an n-bit register.

3. Components of SNOW 3G

3.1. Functions used in different Components of SNOW 3G

3.1.1. MULx

MULx maps 16 bits to 8 bits. Let V and c be 8-bit input values. Then MULx is defined:
If the leftmost (i.e. the most significant) bit of V equals 1, then

$\text{MULx}(V, c) = (V <<_8 1) \oplus c$,

else

$\text{MULx}(V, c) = V <<_8 1$.

Example:

MULx(0x69,0x1B) = 0xC2
MULx(0x96,0x1B) = 0x2C ⊕ 0x1B = 0x37.

3.1.2. MULxPOW

MULxPOW maps 16 bits and an positive integer i to 8 bit. Let V and c be 8-bit input values, then MULxPOW(V, i, c) is recursively defined:
If i equals 0, then

$\text{MULxPOW}(V, i, c) = V$,

else

$\text{MULxPOW}(V, i, c) = \text{MULx}(\text{MULxPOW}(V, i - 1, c), c)$.

3.2. Linear Feedback Shift Register (LFSR)

The Linear Feedback Shift Register (LFSR) consists of sixteen stages $s_0, s_1, s_2, \ldots, s_{15}$ each holding 32 bits.

3.3. Finite State Machine (FSM)

The Finite State Machine (FSM) has three 32-bit registers $R1$, $R2$ and $R3$.

The S-boxes S_1 and S_2 are used to update the registers $R2$ and $R3$.

3.3.1. The 32x32-bit S-Box S_1

The S-Box S_1 maps a 32-bit input to a 32-bit output.
Let $w = w_0 \| w_1 \| w_2 \| w_3$ the 32-bit input with w_0 the most and w_3 the least significant byte.
Let $S_1(w) = r_0 \| r_1 \| r_2 \| r_3$ with r_0 the most and r_3 the least significant byte. We use the 8 to 8 bit Rijndael S-Box S_R defined in 5.1.

Then r_0, r_1, r_2, r_3 are defined as

$r_0 = \text{MULx}(S_R(w_0), 0x1B) \oplus S_R(w_1) \oplus S_R(w_2) \oplus \text{MULx}(S_R(w_3), 0x1B) \oplus S_R(w_3)$,

$r_1 = \text{MULx}(S_R(w_0), 0x1B) \oplus S_R(w_0) \oplus \text{MULx}(S_R(w_1), 0x1B) \oplus S_R(w_2) \oplus S_R(w_3)$,

$r_2 = S_R(w_0) \oplus \text{MULx}(S_R(w_1), 0x1B) \oplus S_R(w_1) \oplus \text{MULx}(S_R(w_2), 0x1B) \oplus S_R(w_3)$,

$r_3 = S_R(w_0) \oplus S_R(w_1) \oplus \text{MULx}(S_R(w_2), 0x1B) \oplus S_R(w_2) \oplus \text{MULx}(S_R(w_3), 0x1B)$.

3.3.2. The 32x32-bit S-Box S2

The S-Box S_2 maps a 32-bit input to a 32-bit output.
Let $w = w_0 \| w_1 \| w_2 \| w_3$ the 32-bit input with w_0 the most and w_3 the least significant byte.
Let $S_2(w) = r_0 \| r_1 \| r_2 \| r_3$ with r_0 the most and r_3 the least significant byte. We use the 8 to 8 bit S-Box S_Q defined in 5.2

Then r_0, r_1, r_2, r_3 are defined as

$r_0 = \text{MULx}(S_Q(w_0), 0x69) \oplus S_Q(w_1) \oplus S_Q(w_2) \oplus \text{MULx}(S_Q(w_3), 0x69) \oplus S_Q(w_3),$

$r_1 = \text{MULx}(S_Q(w_0), 0x69) \oplus S_Q(w_0) \oplus \text{MULx}(S_Q(w_1), 0x69) \oplus S_Q(w_2) \oplus S_Q(w_3),$

$r_2 = S_Q(w_0) \oplus \text{MULx}(S_Q(w_1), 0x69) \oplus S_Q(w_1) \oplus \text{MULx}(S_Q(w_2), 0x69) \oplus S_Q(w_3),$

$r_3 = S_Q(w_0) \oplus S_Q(w_1) \oplus \text{MULx}(S_Q(w_2), 0x69) \oplus S_Q(w_2) \oplus \text{MULx}(S_Q(w_3), 0x69).$

3.4. The Clocking Operations

3.4.1. Clocking the LFSR

The clocking of the LFSR has two different modes of operation, the Initialisation Mode 3.4.4 and the Keystream Mode 3.4.5.
In both modes the functions MUL_α and DIV_α are used which are defined in 3.4.2 resp. 3.4.3.

3.4.2. The function MUL_α

The function MUL_α maps 8 bits to 32 bits. Let c be the 8-bit input, then MUL_α is defined as

$\text{MUL}_\alpha(c) =$

$(\text{MULxPOW}(c, 23, 0xA9) \| \text{MULxPOW}(c, 245, 0xA9) \| \text{MULxPOW}(c, 48, 0xA9) \| \text{MULxPOW}(c, 239, 0xA9)).$

3.4.3. The function DIV_α

The function DIV_α maps 8 bits to 32 bits. Let c be the 8-bit input, then DIV_α is defined as

$\text{DIV}_\alpha(c) =$

$(\text{MULxPOW}(c, 16, 0xA9) \| \text{MULxPOW}(c, 39, 0xA9) \| \text{MULxPOW}(c, 6, 0xA9) \| \text{MULxPOW}(c, 64, 0xA9)).$

3.4.4. Initialisation Mode

In the Initialisation Mode the LFSR receives a 32-bit input word F, which is the output of the FSM.

Let $s_0 = s_{0,0} \| s_{0,1} \| s_{0,2} \| s_{0,3}$ with $s_{0,0}$ being the most and $s_{0,3}$ being the least significant byte of s_0.

Let $s_{11} = s_{11,0} \| s_{11,1} \| s_{11,2} \| s_{11,3}$ with $s_{11,0}$ being the most and $s_{11,3}$ being the least significant byte of s_{11}.

Compute the intermediate value v as

$v = (s_{0,1} \| s_{0,2} \| s_{0,3} \| 0x00) \oplus MUL_\alpha(s_{0,0}) \oplus s_2 \oplus (0x00 \| s_{11,0} \| s_{11,1} \| s_{11,2}) \oplus DIV_\alpha(s_{11,3}) \oplus F.$

Set

$s_0 = s_1$, $s_1 = s_2$, $s_2 = s_3$, $s_3 = s_4$, $s_4 = s_5$, $s_5 = s_6$, $s_6 = s_7$, $s_7 = s_8$,
$s_8 = s_9$, $s_9 = s_{10}$, $s_{10} = s_{11}$, $s_{11} = s_{12}$, $s_{12} = s_{13}$, $s_{13} = s_{14}$, $s_{14} = s_{15}$, $s_{15} = v$.

3.4.5. Keystream Mode

In the Keystream Mode the LFSR does not receive any input.

Let $s_0 = s_{0,0} \| s_{0,1} \| s_{0,2} \| s_{0,3}$ with $s_{0,0}$ being the most and $s_{0,3}$ being the least significant byte of s_0.

Let $s_{11} = s_{11,0} \| s_{11,1} \| s_{11,2} \| s_{11,3}$ with $s_{11,0}$ being the most and $s_{11,3}$ being the least significant byte of s_{11}.

Compute the intermediate value v as

$v = (s_{0,1} \| s_{0,2} \| s_{0,3} \| 0x00) \oplus MUL_\alpha(s_{0,0}) \oplus s_2 \oplus (0x00 \| s_{11,0} \| s_{11,1} \| s_{11,2}) \oplus DIV_\alpha(s_{11,3}).$

Set

$s_0 = s_1$, $s_1 = s_2$, $s_2 = s_3$, $s_3 = s_4$, $s_4 = s_5$, $s_5 = s_6$, $s_6 = s_7$, $s_7 = s_8$,
$s_8 = s_9$, $s_9 = s_{10}$, $s_{10} = s_{11}$, $s_{11} = s_{12}$, $s_{12} = s_{13}$, $s_{13} = s_{14}$, $s_{14} = s_{15}$, $s_{15} = v$.

3.4.6. Clocking the FSM

The FSM has two input words s_{15} and s_5 from the LFSR.
It produces a 32-bit output word F:

$F = (\mathbf{s_{15}} \boxplus R1) \oplus R2$

Then the registers are updated.
Compute the intermediate value r as

$r = R2 \boxplus (R3 \oplus \mathbf{s_5})$.

Set

$R3 = S_2(R2)$,

$R2 = S_1(R1)$,

$R1 = r$.

4. Operation of SNOW 3G

4.1. Initialisation

SNOW 3G is initialized with a 128-bit key consisting of four 32-bit words $\mathbf{k_0}$, $\mathbf{k_1}$, $\mathbf{k_2}$, $\mathbf{k_3}$ and an 128-bit initialisation variable consisting of four 32-bit words $\mathbf{IV_0}$, $\mathbf{IV_1}$, $\mathbf{IV_2}$, $\mathbf{IV_3}$ as follows.

Let **1** be the all-ones word (0xffffffff).

$s_{15} = \mathbf{k_3} \oplus \mathbf{IV_0}$	$s_{14} = \mathbf{k_2}$	$s_{13} = \mathbf{k_1}$	$s_{12} = \mathbf{k_0} \oplus \mathbf{IV_1}$
$s_{11} = \mathbf{k_3} \oplus \mathbf{1}$	$s_{10} = \mathbf{k_2} \oplus \mathbf{1} \oplus \mathbf{IV_2}$	$s_9 = \mathbf{k_1} \oplus \mathbf{1} \oplus \mathbf{IV_3}$	$s_8 = \mathbf{k_0} \oplus \mathbf{1}$
$s_7 = \mathbf{k_3}$	$s_6 = \mathbf{k_2}$	$s_5 = \mathbf{k_1}$	$s_4 = \mathbf{k_0}$
$s_3 = \mathbf{k_3} \oplus \mathbf{1}$	$s_2 = \mathbf{k_2} \oplus \mathbf{1}$	$s_1 = \mathbf{k_1} \oplus \mathbf{1}$	$s_0 = \mathbf{k_0} \oplus \mathbf{1}$

The FSM is initialised with $R1 = R2 = R3 = 0$.

Then the cipher runs in a special mode without producing output:

repeat 32-times {

STEP 1: The FSM is clocked (see 3.4.6) producing the 32-bit word F.

STEP 2: Then the LFSR is clocked in Initialisation Mode (see 3.4.4) consuming F.

}

4.2. Generation of Keystream

First the FSM is clocked once, see 3.4.6. The output word of the FSM is discarded.
Then the LFSR is clocked once in Keystream Mode, see 3.4.4.

After that n 32-bit words of keystream are produced:
for $t = 1$ to n {
STEP 1: The FSM is clocked (see 3.4.6) and produces a 32-bit output word F.
STEP 2: The next keystream word is computed as $z_t = F \oplus s_0$.
STEP 3: Then the LFSR is clocked in Keystream Mode, see 3.4.4.
}

5. Definition of Tables used in SNOW 3G

5.1. The Rijndael S-box S_R

The S-box S_R maps 8 bit to 8 bit. Here the input and output is presented in hexadecimal form.

Let x_0, x_1, y_0, y_1 be hexadecimal digits with $S_R(x_0\ 2^4+x_1) = y_0\ 2^4 + y_1$, then the cell at the intersection of the x_0^{th} row and the x_1^{th} column contains the values for $y_0||y_1$ in hexadecimal form.

For example $S_R(42) = S_R(0x2A) = 0xE5 = 229$.

x_0 \ x_1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 1: Rijndael S-Box

5.2. The S-box S_Q

The S-box S_Q maps 8 bit to 8 bit. Here the input is presented in hexadecimal form.

Let x_0, x_1, y_0, y_1 be hexadecimal digits with $S_Q(x_0\ 2^4+x_1) = y_0\ 2^4 + y_1$, then the cell at the intersection of the x_0^{th} row and the x_1^{th} column contains the values for $y_0||y_1$ in hexadecimal form.

For example $S_Q(42) = S_Q(0x2A) = 0xAC = 172$.

x_0 \ x_1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	25	24	73	67	D7	AE	5C	30	A4	EE	6E	CB	7D	B5	82	DB
1	E4	8E	48	49	4F	5D	6A	78	70	88	E8	5F	5E	84	65	E2
2	D8	E9	CC	ED	40	2F	11	28	57	D2	AC	E3	4A	15	1B	B9
3	B2	80	85	A6	2E	02	47	29	07	4B	0E	C1	51	AA	89	D4
4	CA	01	46	B3	EF	DD	44	7B	C2	7F	BE	C3	9F	20	4C	64
5	83	A2	68	42	13	B4	41	CD	BA	C6	BB	6D	4D	71	21	F4
6	8D	B0	E5	93	FE	8F	E6	CF	43	45	31	22	37	36	96	FA
7	BC	0F	08	52	1D	55	1A	C5	4E	23	69	7A	92	FF	5B	5A
8	EB	9A	1C	A9	D1	7E	0D	FC	50	8A	B6	62	F5	0A	F8	DC
9	03	3C	0C	39	F1	B8	F3	3D	F2	D5	97	66	81	32	A0	00
A	06	CE	F6	EA	B7	17	F7	8C	79	D6	A7	BF	8B	3F	1F	53
B	63	75	35	2C	60	FD	27	D3	94	A5	7C	A1	05	58	2D	BD
C	D9	C7	AF	6B	54	0B	E0	38	04	C8	9D	E7	14	B1	87	9C
D	DF	6F	F9	DA	2A	C4	59	16	74	91	AB	26	61	76	34	2B
E	AD	99	FB	72	EC	33	12	DE	98	3B	C0	9B	3E	18	10	3A
F	56	E1	77	C9	1E	9E	95	A3	90	19	A8	6C	09	D0	F0	86

Table 2: S-Box S_Q

INFORMATIVE SECTION

This part of the document is purely informative and does not form part of the normative specification of **SNOW 3G**.

ANNEX 1
Remarks about the mathematical background of some operations of the SNOW 3G Algorithm

1.1 MULx and MULxPOW

The function MULx (see 3.1.1) corresponds to the multiplication with the primitive element of an extension of degree 8 of GF(2). Describe $GF(2^8)$ as GF(2)(β) with β being the root of the irreducible polynomial
$x^8 + c_0x^7 + c_1x^6 + c_2x^5 + c_3x^4 + c_4x^3 + c_5x^2 + c_6x + c_7$.
Define the 8-bit value c as $c = c_0 \| c_1 \| c_2 \| c_3 \| c_4 \| c_5 \| c_6 \| c_7$ with c_0 the most and c_7 the least significant bit. Then for an element V of $GF(2^8)$ the result of MULx(V, c) corresponds to $V \cdot \beta$.

MULxPOW (see 3.1.2) corresponds to the multiplication with the primitive element raised to the power of an positive integer i.
With the definition of above MULxPOW(V, i, c) corresponds to $V \cdot \beta^i$ in GF(2)(β).

1.2 The S-Box S_1 used in the FSM

The S-Box S_1 is based on the round function of Rijndael [8].

Consider an 32-bit input word $w = w_0 \| w_1 \| w_2 \| w_3$ where w_0 is the most and w_3 the least significant byte.

Here the bytes are interpreted as elements of $GF(2^8)$ defined by the polynomial

$x^8 + x^4 + x^3 + x + 1$.

Hence $w_i = w_{i0} \| w_{i1} \| w_{i2} \| w_{i3} \| w_{i4} \| w_{i5} \| w_{i6} \| w_{i7}$ with w_{i0} the most and w_{i7} the least significant bit is interpreted as $w_{i0} x^7 + w_{i1} x^6 + w_{i2} x^5 + w_{i3} x^4 + w_{i4} x^3 + w_{i5} x^2 + w_{i6} x^1 + w_{i7}$.

Then the output $S_1(w) = r = r_0 \| r_1 \| r_2 \| r_3$ with r_0 the most r_1 the least significant byte is defined as

$$
\begin{aligned}
r_0 &= (x+1)\,S_R(w_3) + S_R(w_2) + S_R(w_1) + x\,S_R(w_0),\\
r_1 &= S_R(w_3) + S_R(w_2) + x\,S_R(w_1) + (x+1)\,S_R(w_0),\\
r_2 &= S_R(w_3) + x\,S_R(w_2) + (x+1)\,S_R(w_1) + S_R(w_0),\\
r_3 &= x\,S_R(w_3) + (x+1)\,S_R(w_2) + S_R(w_1) + S_R(w_0).
\end{aligned}
$$

Note: The notation used here differs from the notation in [7].

1.3 The S-Box S_Q used in the S-Box S2

The S-Box S_Q is constructed using the Dickson polynomial
$g_{49}(x) = x \oplus x^9 \oplus x^{13} \oplus x^{15} \oplus x^{33} \oplus x^{41} \oplus x^{45} \oplus x^{47} \oplus x^{49}$.
For an 8-bit input x in $GF(2^8)$ defined by the polynomial $x^8 + x^6 + x^5 + x^3 + 1$ the 8-bit output of S_Q corresponds to $g_{49}(x) \oplus 0x25$.

1.4 The S-Box S_2 used in the FSM

The S-Box S_2 is based on the S-box S_Q.

Consider an 32-bit input word $w = w_0 \| w_1 \| w_2 \| w_3$ where w_0 is the most and w_3 the least significant byte.

Here the bytes are interpreted as elements of $GF(2^8)$ defined by the polynomial

$x^8 + x^6 + x^5 + x^3 + 1$.

Hence $w_i = w_{i0} \| w_{i1} \| w_{i2} \| w_{i3} \| w_{i4} \| w_{i5} \| w_{i6} \| w_{i7}$ with w_{i0} the most and w_{i7} the least significant bit is interpreted as $w_{i0}\, x^7 + w_{i1}\, x^6 + w_{i2}\, x^5 + w_{i3}\, x^4 + w_{i4}\, x^3 + w_{i5}\, x^2 + w_{i6}\, x^1 + w_{i7}$.

Then the output $S_2(w) = r = r_0 \| r_1 \| r_2 \| r_3$ with r_0 the most r_1 the least significant byte is defined as

$$r_0 = (x+1)\, S_Q(w_3) + S_Q(w_2) + S_Q(w_1) + x\, S_Q(w_0),$$

$$r_1 = S_Q(w_3) + S_Q(w_2) + x\, S_Q(w_1) + (x+1)\, S_Q(w_0),$$

$$r_2 = S_Q(w_3) + x\, S_Q(w_2) + (x+1)\, S_Q(w_1) + S_Q(w_0),$$

$$r_3 = x\, S_Q(w_3) + (x+1)\, S_Q(w_2) + S_Q(w_1) + S_Q(w_0).$$

1.5 Interpretation of the 32-bit words contained in the LFSR as elements of $GF(2^{32})$

The 32-bit words can be interpreted as elements of the Galois Field $GF(2^{32})$.
Let β be a root of the irreducible GF(2)[x]-polynomial $x^8+x^7+x^5+x^3+1$.
The elements of $GF(2^8)$ are expressed on the base $\{\beta^7, \beta^6, \beta^5, \beta^4, \beta^3, \beta^2, \beta, 1\}$.
Let α be a root of the irreducible $GF(2^8)[x]$ polynomial $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$.
The elements of $GF(2^{32})$ are expressed on the base $\{\alpha^3, \alpha^2, \alpha, 1\}$.
Hence if a 32-bit word w is stored in 4 bytes b_0, b_1, b_3, b_3 and each byte b_i consists of the bits $b_{i0}, b_{i1}, \ldots, b_{i7}$, then this word can be interpreted as

$w = b_0\alpha^3 + b_1\alpha^2 + b_2\,\alpha + b_3 =$

$(b_{00}\beta^7 + b_{01}\beta^6 + b_{02}\beta^5 + b_{03}\beta^4 + b_{04}\beta^3 + b_{05}\beta^2 + b_{06}\beta + b_{07})\,\alpha^3 +$

$(b_{10}\beta^7 + b_{11}\beta^6 + b_{12}\beta^5 + b_{13}\beta^4 + b_{14}\beta^3 + b_{15}\beta^2 + b_{16}\beta + b_{17})\,\alpha^2 +$

$(b_{20}\beta^7 + b_{21}\beta^6 + b_{22}\beta^5 + b_{23}\beta^4 + b_{24}\beta^3 + b_{25}\beta^2 + b_{26}\beta + b_{27})\,\alpha +$

$b_{30}\beta^7 + b_{31}\beta^6 + b_{32}\beta^5 + b_{33}\beta^4 + b_{34}\beta^3 + b_{35}\beta^2 + b_{36}\beta + b_{37}$

With these definitions multiplication of w with α is equal to a byte shift of w to the left and an XOR with the result of MUL_α.

Similarly a multiplication with α^{-1} can be implemented as a byte shift to the right and an XOR with the result of DIV_α.

ANNEX 2
Implementation options for some operations of the SNOW 3G Algorithm

In this Annex some alternative definitions are given for some operations of SNOW 3G.

2.1. The S-Box S_1 used in the FSM

Consider an 32-bit input word $w = w_0 \| w_1 \| w_2 \| w_3$ where w_0 is the most and w_3 the least significant byte.
In order to compute $S_1(w)$ we use 4 table lookups from the tables S1_T0, S1_T1, S1_T2 and S1_T3 defined in 2.4. Each of these tables maps 8 bits to 32 bits.

Then $S_1(w)$ is computed according to

$S_1(w) = S1\_T0(w_3) \oplus S1\_T1(w_2) \oplus S1\_T2(w_1) \oplus S1\_T3(w_0)$.

Note: The notation used here differs from the notation in [7].

2.2. The S-Box S_2 used in the FSM

Consider an 32-bit input word $w = w_0 \| w_1 \| w_2 \| w_3$ where w_0 is the most and w_3 the least significant byte.
In order to compute $S_2(w)$ we use 4 table lookups from the tables S2_T0, S2_T1, S2_T2 and S2_T3 defined in 2.4. Each of these tables maps 8 bits to 32 bits.

Then $S_2(w)$ is computed according to

$S_2(w) = S2\_T0(w_3) \oplus S2\_T1(w_2) \oplus S2\_T2(w_1) \oplus S2\_T3(w_0)$.

2.3. The functions MUL_α and DIV_α used in the LFSR

The functions MUL_α and DIV_α can be implemented as a table lookup from the tables MUL_{alpha} and DIV_{alpha} defined in 2.5.

2.4. Definitions of tables for the FSM

S1_T0:

0x00	0xA56363C6	0x847C7CF8	0x997777EE	0x8D7B7BF6	0xDF2F2FF	0xBD6B6BD6	0xB16F6FDE	0x54C5C591
0x08	0x50303060	0x3010102	0xA96767CE	0x7D2B2B56	0x19FEFEE7	0x62D7D7B5	0xE6ABAB4D	0x9A7676EC
0x10	0x45CACA8F	0x9D82821F	0x40C9C989	0x877D7DFA	0x15FAFAEF	0xEB5959B2	0xC947478E	0xBF0F0FB
0x18	0xECADAD41	0x67D4D4B3	0xFDA2A25F	0xEAAFAF45	0xBF9C9C23	0xF7A4A453	0x967272E4	0x5BC0C09B
0x20	0xC2B7B775	0x1CFDFDE1	0xAE93933D	0x6A26264C	0x5A36366C	0x413F3F7E	0x2F7F7F5	0x4FCCCC83
0x28	0x5C343468	0xF4A5A551	0x34E5E5D1	0x8F1F1F9	0x937171E2	0x73D8D8AB	0x53313162	0x3F15152A
0x30	0xC040408	0x52C7C795	0x65232346	0x5EC3C39D	0x28181830	0xA1969637	0xF05050A	0xB59A9A2F
0x38	0x907070E	0x36121224	0x9B80801B	0x3DE2E2DF	0x26EBEBCD	0x6927274E	0xCDB2B27F	0x9F7575EA
0x40	0x1B090912	0x9E83831D	0x742C2C58	0x2E1A1A34	0x2D1B1B36	0xB26E6EDC	0xEE5A5AB4	0xFBA0A05B
0x48	0xF65252A4	0x4D3B3B76	0x61D6D6B7	0xCEB3B37D	0x7B292952	0x3EE3E3DD	0x712F2F5E	0x97848413
0x50	0xF55353A6	0x68D1D1B9	0x0	0x2CEDEDC1	0x60202040	0x1FFCFCE3	0xC8B1B179	0xED5B5BB6
0x58	0xBE6A6AD4	0x46CBCB8D	0xD9BEBE67	0x4B393972	0xDE4A4A94	0xD44C4C98	0xE85858B0	0x4ACFCF85
0x60	0x6BD0D0BB	0x2AEFEFC5	0xE5AAAA4F	0x16FBFBED	0xC5434386	0xD74D4D9A	0x55333366	0x94858511
0x68	0xCF45458A	0x10F9F9E9	0x6020204	0x817F7FFE	0xF05050A0	0x443C3C78	0xBA9F9F25	0xE3A8A84B
0x70	0xF35151A2	0xFEA3A35D	0xC0404080	0x8A8F8F05	0xAD92923F	0xBC9D9D21	0x48383870	0x4F5F5F1
0x78	0xDFBCBC63	0xC1B6B677	0x75DADAAF	0x63212142	0x30101020	0x1AFFFFE5	0xEF3F3FD	0x6DD2D2BF
0x80	0x4CCDCD81	0x140C0C18	0x35131326	0x2FECECC3	0xE15F5FBE	0xA2979735	0xCC444488	0x3917172E
0x88	0x57C4C493	0xF2A7A755	0x827E7EFC	0x473D3D7A	0xAC6464C8	0xE75D5DBA	0x2B191932	0x957373E6
0x90	0xA06060C0	0x98818119	0xD14F4F9E	0x7FDCDCA3	0x66222244	0x7E2A2A54	0xAB90903B	0x8388880B
0x98	0xCA46468C	0x29EEEEC7	0xD3B8B86B	0x3C141428	0x79DEDEA7	0xE25E5EBC	0x1D0B0B16	0x76DBDBAD
0xA0	0x3BE0E0DB	0x56323264	0x4E3A3A74	0x1E0A0A14	0xDB494992	0xA06060C	0x6C242448	0xE45C5CB8
0xA8	0x5DC2C29F	0x6ED3D3BD	0xEFACAC43	0xA66262C4	0xA8919139	0xA4959531	0x37E4E4D3	0x8B7979F2
0xB0	0x32E7E7D5	0x43C8C88B	0x5937376E	0xB76D6DDA	0x8C8D8D01	0x64D5D5B1	0xD24E4E9C	0xE0A9A949
0xB8	0xB46C6CD8	0xFA5656AC	0x7F4F4F3	0x25EAEACF	0xAF6565CA	0x8E7A7AF4	0xE9AEAE47	0x18080810
0xC0	0xD5BABA6F	0x887878F0	0x6F25254A	0x722E2E5C	0x241C1C38	0xF1A6A657	0xC7B4B473	0x51C6C697
0xC8	0x23E8E8CB	0x7CDDDDA1	0x9C7474E8	0x211F1F3E	0xDD4B4B96	0xDCBDBD61	0x868B8B0D	0x858A8A0F
0xD0	0x907070E0	0x423E3E7C	0xC4B5B571	0xAA6666CC	0xD8484890	0x5030306	0x1F6F6F7	0x120E0E1C
0xD8	0xA36161C2	0x5F35356A	0xF95757AE	0xD0B9B969	0x91868617	0x58C1C199	0x271D1D3A	0xB99E9E27
0xE0	0x38E1E1D9	0x13F8F8EB	0xB398982B	0x33111122	0xBB6969D2	0x70D9D9A9	0x898E8E07	0xA7949433
0xE8	0xB69B9B2D	0x221E1E3C	0x92878715	0x20E9E9C9	0x49CECE87	0xFF5555AA	0x78282850	0x7ADFDFA5
0xF0	0x8F8C8C03	0xF8A1A159	0x80898909	0x170D0D1A	0xDABFBF65	0x31E6E6D7	0xC6424284	0xB86868D0
0xF8	0xC3414182	0xB0999929	0x772D2D5A	0x110F0F1E	0xCBB0B07B	0xFC5454A8	0xD6BBBB6D	0x3A16162C

Table 3: S1_T0

For example S1_T0(2A) = 0x34E5E5D1.

S1_T1:

0x00	0x6363C6A5	0x7C7CF884	0x7777EE99	0x7B7BF68D	0xF2F2FF0D	0x6B6BD6BD	0x6F6FDEB1	0xC5C59154
0x08	0x30306050	0x1010203	0x6767CEA9	0x2B2B567D	0xFEFEE719	0xD7D7B562	0xABAB4DE6	0x7676EC9A
0x10	0xCACA8F45	0x82821F9D	0xC9C98940	0x7D7DFA87	0xFAFAEF15	0x5959B2EB	0x47478EC9	0xF0F0FB0B
0x18	0xADAD41EC	0xD4D4B367	0xA2A25FFD	0xAFAF45EA	0x9C9C23BF	0xA4A453F7	0x7272E496	0xC0C09B5B
0x20	0xB7B775C2	0xFDFDE11C	0x93933DAE	0x26264C6A	0x36366C5A	0x3F3F7E41	0xF7F7F502	0xCCCC834F
0x28	0x3434685C	0xA5A551F4	0xE5E5D134	0xF1F1F908	0x7171E293	0xD8D8AB73	0x31316253	0x15152A3F
0x30	0x404080C	0xC7C79552	0x23234665	0xC3C39D5E	0x18183028	0x969637A1	0x5050A0F	0x9A9A2FB5
0x38	0x7070E09	0x12122436	0x80801B9B	0xE2E2DF3D	0xEBEBCD26	0x27274E69	0xB2B27FCD	0x7575EA9F
0x40	0x909121B	0x83831D9E	0x2C2C5874	0x1A1A342E	0x1B1B362D	0x6E6EDCB2	0x5A5AB4EE	0xA0A05BFB
0x48	0x5252A4F6	0x3B3B764D	0xD6D6B761	0xB3B37DCE	0x2929527B	0xE3E3DD3E	0x2F2F5E71	0x84841397
0x50	0x5353A6F5	0xD1D1B968	0x0	0xEDEDC12C	0x20204060	0xFCFCE31F	0xB1B179C8	0x5B5BB6ED
0x58	0x6A6AD4BE	0xCBCB8D46	0xBEBE67D9	0x3939724B	0x4A4A94DE	0x4C4C98D4	0x5858B0E8	0xCFCF854A
0x60	0xD0D0BB6B	0xEFEFC52A	0xAAAA4FE5	0xFBFBED16	0x434386C5	0x4D4D9AD7	0x33336655	0x85851194
0x68	0x45458ACF	0xF9F9E910	0x2020406	0x7F7FFE81	0x5050A0F0	0x3C3C7844	0x9F9F25BA	0xA8A84BE3
0x70	0x5151A2F3	0xA3A35DFE	0x404080C0	0x8F8F058A	0x92923FAD	0x9D9D21BC	0x38387048	0xF5F5F104
0x78	0xBCBC63DF	0xB6B677C1	0xDADAAF75	0x21214263	0x10102030	0xFFFFE51A	0xF3F3FD0E	0xD2D2BF6D
0x80	0xCDCD814C	0xC0C1814	0x13132635	0xECECC32F	0x5F5FBEE1	0x979735A2	0x444488CC	0x17172E39
0x88	0xC4C49357	0xA7A755F2	0x7E7EFC82	0x3D3D7A47	0x6464C8AC	0x5D5DBAE7	0x1919322B	0x7373E695
0x90	0x6060C0A0	0x81811998	0x4F4F9ED1	0xDCDCA37F	0x22224466	0x2A2A547E	0x90903BAB	0x88880B83
0x98	0x46468CCA	0xEEEEC729	0xB8B86BD3	0x1414283C	0xDEDEA779	0x5E5EBCE2	0xB0B161D	0xDBDBAD76
0xA0	0xE0E0DB3B	0x32326456	0x3A3A744E	0xA0A141E	0x494992DB	0x6060C0A	0x2424486C	0x5C5CB8E4
0xA8	0xC2C29F5D	0xD3D3BD6E	0xACAC43EF	0x6262C4A6	0x919139A8	0x959531A4	0xE4E4D337	0x7979F28B
0xB0	0xE7E7D532	0xC8C88B43	0x37376E59	0x6D6DDAB7	0x8D8D018C	0xD5D5B164	0x4E4E9CD2	0xA9A949E0
0xB8	0x6C6CD8B4	0x5656ACFA	0xF4F4F307	0xEAEACF25	0x6565CAAF	0x7A7AF48E	0xAEAE47E9	0x8081018
0xC0	0xBABA6FD5	0x7878F088	0x25254A6F	0x2E2E5C72	0x1C1C3824	0xA6A657F1	0xB4B473C7	0xC6C69751
0xC8	0xE8E8CB23	0xDDDDA17C	0x7474E89C	0x1F1F3E21	0x4B4B96DD	0xBDBD61DC	0x8B8B0D86	0x8A8A0F85
0xD0	0x7070E090	0x3E3E7C42	0xB5B571C4	0x6666CCAA	0x484890D8	0x3030605	0xF6F6F701	0xE0E1C12
0xD8	0x6161C2A3	0x35356A5F	0x5757AEF9	0xB9B969D0	0x86861791	0xC1C19958	0x1D1D3A27	0x9E9E27B9
0xE0	0xE1E1D938	0xF8F8EB13	0x98982BB3	0x11112233	0x6969D2BB	0xD9D9A970	0x8E8E0789	0x949433A7
0xE8	0x9B9B2DB6	0x1E1E3C22	0x87871592	0xE9E9C920	0xCECE8749	0x5555AAFF	0x28285078	0xDFDFA57A
0xF0	0x8C8C038F	0xA1A159F8	0x89890980	0xD0D1A17	0xBFBF65DA	0xE6E6D731	0x424284C6	0x6868D0B8
0xF8	0x414182C3	0x999929B0	0x2D2D5A77	0xF0F1E11	0xB0B07BCB	0x5454A8FC	0xBBBB6DD6	0x16162C3A

Table 4: S1_T1

S1_T2:

0x00	0x63C6A563	0x7CF8847C	0x77EE9977	0x7BF68D7B	0xF2FF0DF2	0x6BD6BD6B	0x6FDEB16F	0xC59154C5
0x08	0x30605030	0x1020301	0x67CEA967	0x2B567D2B	0xFEE719FE	0xD7B562D7	0xAB4DE6AB	0x76EC9A76
0x10	0xCA8F45CA	0x821F9D82	0xC98940C9	0x7DFA877D	0xFAEF15FA	0x59B2EB59	0x478EC947	0xF0FB0BF0
0x18	0xAD41ECAD	0xD4B367D4	0xA25FFDA2	0xAF45EAAF	0x9C23BF9C	0xA453F7A4	0x72E49672	0xC09B5BC0
0x20	0xB775C2B7	0xFDE11CFD	0x933DAE93	0x264C6A26	0x366C5A36	0x3F7E413F	0xF7F502F7	0xCC834FCC
0x28	0x34685C34	0xA551F4A5	0xE5D134E5	0xF1F908F1	0x71E29371	0xD8AB73D8	0x31625331	0x152A3F15
0x30	0x4080C04	0xC79552C7	0x23466523	0xC39D5EC3	0x18302818	0x9637A196	0x50A0F05	0x9A2FB59A
0x38	0x70E0907	0x12243612	0x801B9B80	0xE2DF3DE2	0xEBCD26EB	0x274E6927	0xB27FCDB2	0x75EA9F75
0x40	0x9121B09	0x831D9E83	0x2C58742C	0x1A342E1A	0x1B362D1B	0x6EDCB26E	0x5AB4EE5A	0xA05BFBA0
0x48	0x52A4F652	0x3B764D3B	0xD6B761D6	0xB37DCEB3	0x29527B29	0xE3DD3EE3	0x2F5E712F	0x84139784
0x50	0x53A6F553	0xD1B968D1	0x0	0xEDC12CED	0x20406020	0xFCE31FFC	0xB179C8B1	0x5BB6ED5B
0x58	0x6AD4BE6A	0xCB8D46CB	0xBE67D9BE	0x39724B39	0x4A94DE4A	0x4C98D44C	0x58B0E858	0xCF854ACF
0x60	0xD0BB6BD0	0xEFC52AEF	0xAA4FE5AA	0xFBED16FB	0x4386C543	0x4D9AD74D	0x33665533	0x85119485
0x68	0x458ACF45	0xF9E910F9	0x2040602	0x7FFE817F	0x50A0F050	0x3C78443C	0x9F25BA9F	0xA84BE3A8
0x70	0x51A2F351	0xA35DFEA3	0x4080C040	0x8F058A8F	0x923FAD92	0x9D21BC9D	0x38704838	0xF5F104F5
0x78	0xBC63DFBC	0xB677C1B6	0xDAAF75DA	0x21426321	0x10203010	0xFFE51AFF	0xF3FD0EF3	0xD2BF6DD2
0x80	0xCD814CCD	0xC18140C	0x13263513	0xECC32FEC	0x5FBEE15F	0x9735A297	0x4488CC44	0x172E3917
0x88	0xC49357C4	0xA755F2A7	0x7EFC827E	0x3D7A473D	0x64C8AC64	0x5DBAE75D	0x19322B19	0x73E69573
0x90	0x60C0A060	0x81199881	0x4F9ED14F	0xDCA37FDC	0x22446622	0x2A547E2A	0x903BAB90	0x880B8388
0x98	0x468CCA46	0xEEC729EE	0xB86BD3B8	0x14283C14	0xDEA779DE	0x5EBCE25E	0xB161D0B	0xDBAD76DB
0xA0	0xE0DB3BE0	0x32645632	0x3A744E3A	0xA141E0A	0x4992DB49	0x60C0A06	0x24486C24	0x5CB8E45C
0xA8	0xC29F5DC2	0xD3BD6ED3	0xAC43EFAC	0x62C4A662	0x9139A891	0x9531A495	0xE4D337E4	0x79F28B79
0xB0	0xE7D532E7	0xC88B43C8	0x376E5937	0x6DDAB76D	0x8D018C8D	0xD5B164D5	0x4E9CD24E	0xA949E0A9
0xB8	0x6CD8B46C	0x56ACFA56	0xF4F307F4	0xEACF25EA	0x65CAAF65	0x7AF48E7A	0xAE47E9AE	0x8101808
0xC0	0xBA6FD5BA	0x78F08878	0x254A6F25	0x2E5C722E	0x1C38241C	0xA657F1A6	0xB473C7B4	0xC69751C6
0xC8	0xE8CB23E8	0xDDA17CDD	0x74E89C74	0x1F3E211F	0x4B96DD4B	0xBD61DCBD	0x8B0D868B	0x8A0F858A
0xD0	0x70E09070	0x3E7C423E	0xB571C4B5	0x66CCAA66	0x4890D848	0x3060503	0xF6F701F6	0xE1C120E
0xD8	0x61C2A361	0x356A5F35	0x57AEF957	0xB969D0B9	0x86179186	0xC19958C1	0x1D3A271D	0x9E27B99E
0xE0	0xE1D938E1	0xF8EB13F8	0x982BB398	0x11223311	0x69D2BB69	0xD9A970D9	0x8E07898E	0x9433A794
0xE8	0x9B2DB69B	0x1E3C221E	0x87159287	0xE9C920E9	0xCE8749CE	0x55AAFF55	0x28507828	0xDFA57ADF
0xF0	0x8C038F8C	0xA159F8A1	0x89098089	0xD1A170D	0xBF65DABF	0xE6D731E6	0x4284C642	0x68D0B868
0xF8	0x4182C341	0x9929B099	0x2D5A772D	0xF1E110F	0xB07BCBB0	0x54A8FC54	0xBB6DD6BB	0x162C3A16

Table 5: S1_T2

S1_T3:

0x00	0xC6A56363	0xF8847C7C	0xEE997777	0xF68D7B7B	0xFF0DF2F2	0xD6BD6B6B	0xDEB16F6F	0x9154C5C5
0x08	0x60503030	0x2030101	0xCEA96767	0x567D2B2B	0xE719FEFE	0xB562D7D7	0x4DE6ABAB	0xEC9A7676
0x10	0x8F45CACA	0x1F9D8282	0x8940C9C9	0xFA877D7D	0xEF15FAFA	0xB2EB5959	0x8EC94747	0xFB0BF0F0
0x18	0x41ECADAD	0xB367D4D4	0x5FFDA2A2	0x45EAAFAF	0x23BF9C9C	0x53F7A4A4	0xE4967272	0x9B5BC0C0
0x20	0x75C2B7B7	0xE11CFDFD	0x3DAE9393	0x4C6A2626	0x6C5A3636	0x7E413F3F	0xF502F7F7	0x834FCCCC
0x28	0x685C3434	0x51F4A5A5	0xD134E5E5	0xF908F1F1	0xE2937171	0xAB73D8D8	0x62533131	0x2A3F1515
0x30	0x80C0404	0x9552C7C7	0x46652323	0x9D5EC3C3	0x30281818	0x37A19696	0xA0F0505	0x2FB59A9A
0x38	0xE090707	0x24361212	0x1B9B8080	0xDF3DE2E2	0xCD26EBEB	0x4E692727	0x7FCDB2B2	0xEA9F7575
0x40	0x121B0909	0x1D9E8383	0x58742C2C	0x342E1A1A	0x362D1B1B	0xDCB26E6E	0xB4EE5A5A	0x5BFBA0A0
0x48	0xA4F65252	0x764D3B3B	0xB761D6D6	0x7DCEB3B3	0x527B2929	0xDD3EE3E3	0x5E712F2F	0x13978484
0x50	0xA6F55353	0xB968D1D1	0x0	0xC12CEDED	0x40602020	0xE31FFCFC	0x79C8B1B1	0xB6ED5B5B
0x58	0xD4BE6A6A	0x8D46CBCB	0x67D9BEBE	0x724B3939	0x94DE4A4A	0x98D44C4C	0xB0E85858	0x854ACFCF
0x60	0xBB6BD0D0	0xC52AEFEF	0x4FE5AAAA	0xED16FBFB	0x86C54343	0x9AD74D4D	0x66553333	0x11948585
0x68	0x8ACF4545	0xE910F9F9	0x4060202	0xFE817F7F	0xA0F05050	0x78443C3C	0x25BA9F9F	0x4BE3A8A8
0x70	0xA2F35151	0x5DFEA3A3	0x80C04040	0x58A8F8F	0x3FAD9292	0x21BC9D9D	0x70483838	0xF104F5F5
0x78	0x63DFBCBC	0x77C1B6B6	0xAF75DADA	0x42632121	0x20301010	0xE51AFFFF	0xFD0EF3F3	0xBF6DD2D2
0x80	0x814CCDCD	0x18140C0C	0x26351313	0xC32FECEC	0xBEE15F5F	0x35A29797	0x88CC4444	0x2E391717
0x88	0x9357C4C4	0x55F2A7A7	0xFC827E7E	0x7A473D3D	0xC8AC6464	0xBAE75D5D	0x322B1919	0xE6957373
0x90	0xC0A06060	0x19988181	0x9ED14F4F	0xA37FDCDC	0x44662222	0x547E2A2A	0x3BAB9090	0xB838888
0x98	0x8CCA4646	0xC729EEEE	0x6BD3B8B8	0x283C1414	0xA779DEDE	0xBCE25E5E	0x161D0B0B	0xAD76DBDB
0xA0	0xDB3BE0E0	0x64563232	0x744E3A3A	0x141E0A0A	0x92DB4949	0xC0A0606	0x486C2424	0xB8E45C5C
0xA8	0x9F5DC2C2	0xBD6ED3D3	0x43EFACAC	0xC4A66262	0x39A89191	0x31A49595	0xD337E4E4	0xF28B7979
0xB0	0xD532E7E7	0x8B43C8C8	0x6E593737	0xDAB76D6D	0x18C8D8D	0xB164D5D5	0x9CD24E4E	0x49E0A9A9
0xB8	0xD8B46C6C	0xACFA5656	0xF307F4F4	0xCF25EAEA	0xCAAF6565	0xF48E7A7A	0x47E9AEAE	0x10180808
0xC0	0x6FD5BABA	0xF0887878	0x4A6F2525	0x5C722E2E	0x38241C1C	0x57F1A6A6	0x73C7B4B4	0x9751C6C6
0xC8	0xCB23E8E8	0xA17CDDDD	0xE89C7474	0x3E211F1F	0x96DD4B4B	0x61DCBDBD	0xD868B8B	0xF858A8A
0xD0	0xE0907070	0x7C423E3E	0x71C4B5B5	0xCCAA6666	0x90D84848	0x6050303	0xF701F6F6	0x1C120E0E
0xD8	0xC2A36161	0x6A5F3535	0xAEF95757	0x69D0B9B9	0x17918686	0x9958C1C1	0x3A271D1D	0x27B99E9E
0xE0	0xD938E1E1	0xEB13F8F8	0x2BB39898	0x22331111	0xD2BB6969	0xA970D9D9	0x7898E8E	0x33A79494
0xE8	0x2DB69B9B	0x3C221E1E	0x15928787	0xC920E9E9	0x8749CECE	0xAAFF5555	0x50782828	0xA57ADFDF
0xF0	0x38F8C8C	0x59F8A1A1	0x9808989	0x1A170D0D	0x65DABFBF	0xD731E6E6	0x84C64242	0xD0B86868
0xF8	0x82C34141	0x29B09999	0x5A772D2D	0x1E110F0F	0x7BCBB0B0	0xA8FC5454	0x6DD6BBBB	0x2C3A1616

Table 6: S1_T3

S2_T0:

0x00	0x6f25254a	0x6c242448	0x957373e6	0xa96767ce	0x10d7d7c7	0x9baeae35	0xe45c5cb8	0x50303060
0x08	0x85a4a421	0x5beeeeb5	0xb26e6edc	0x34cbcbff	0x877d7dfa	0xb6b5b503	0xef82826d	0x04dbdbdf
0x10	0x45e4e4a1	0xfb8e8e75	0xd8484890	0xdb494992	0xd14f4f9e	0xe75d5dba	0xbe6a6ad4	0x887878f0
0x18	0x907070e0	0xf1888879	0x51e8e8b9	0xe15f5fbe	0xe25e5ebc	0xe5848461	0xaf6565ca	0x4fe2e2ad
0x20	0x01d8d8d9	0x52e9e9bb	0x3dcccf1	0x5eededb3	0xc0404080	0x712f2f5e	0x33111122	0x78282850
0x28	0xf95757ae	0x1fd2d2cd	0x9dacac31	0x4ce3e3af	0xde4a4a94	0x3f15152a	0x2d1b1b36	0xa2b9b91b
0x30	0xbfb2b20d	0xe9808069	0xe6858563	0x83a6a625	0x722e2e5c	0x06020204	0xc947478e	0x7b292952
0x38	0x0907070e	0xdd4b4b96	0x120e0e1c	0x2ac1c1eb	0xf35151a2	0x97aaaa3d	0xf289897b	0x15d4d4c1
0x40	0x37cacafd	0x03010102	0xca46468c	0xbcb3b30f	0x58efefb7	0x0eddddd3	0xcc444488	0x8d7b7bf6
0x48	0x2fc2c2ed	0x817f7ffe	0xabbebe15	0x2cc3c3ef	0xc89f9f57	0x60202040	0xd44c4c98	0xac6464c8
0x50	0xec83836f	0x8fa2a22d	0xb86868d0	0xc6424284	0x35131326	0xb5b4b401	0xc3414182	0x3ecdcdf3
0x58	0xa7baba1d	0x23c6c6e5	0xa4bbbb1f	0xb76d6dda	0xd74d4d9a	0x937171e2	0x63212142	0x75f4f481
0x60	0xfe8d8d73	0xb9b0b009	0x46e5e5a3	0xdc93934f	0x6bfefe95	0xf88f8f77	0x43e6e6a5	0x38cfcff7
0x68	0xc5434386	0xcf45458a	0x53313162	0x66222244	0x5937376e	0x5a36366c	0xd3969645	0x67fafa9d
0x70	0xadbcbc11	0x110f0f1e	0x18080810	0xf65252a4	0x271d1d3a	0xff5555aa	0x2e1a1a34	0x26c5c5e3
0x78	0xd24e4e9c	0x65232346	0xbb6969d2	0x8e7a7af4	0xdf92924d	0x68ffff97	0xed5b5bb6	0xee5a5ab4
0x80	0x54ebebbf	0xc79a9a5d	0x241c1c38	0x92a9a93b	0x1ad1d1cb	0x827e7efc	0x170d0d1a	0x6dfcfc91
0x88	0xf05050a0	0xf78a8a7d	0xb3b6b605	0xa66262c4	0x76f5f583	0x1e0a0a14	0x61f8f899	0x0ddcdcd1
0x90	0x05030306	0x443c3c78	0x140c0c18	0x4b393972	0x7af1f18b	0xa1b8b819	0x7cf3f38f	0x473d3d7a
0x98	0x7ff2f28d	0x16d5d5c3	0xd0979747	0xaa6666cc	0xea81816b	0x56323264	0x89a0a029	0x00000000
0xA0	0x0a06060c	0x3bcecef5	0x73f6f685	0x57eaeabd	0xb0b7b707	0x3917172e	0x70f7f787	0xfd8c8c71
0xA8	0x8b7979f2	0x13d6d6c5	0x80a7a727	0xa8bfbf17	0xf48b8b7f	0x413f3f7e	0x211f1f3e	0xf55353a6
0xB0	0xa56363c6	0x9f7575ea	0x5f35356a	0x742c2c58	0xa06060c0	0x6efdfd93	0x6927274e	0x1cd3d3cf
0xB8	0xd5949441	0x86a5a523	0x847c7cf8	0x8aa1a12b	0x0f05050a	0xe85858b0	0x772d2d5a	0xaebdbd13
0xC0	0x02d9d9db	0x20c7c7e7	0x98afaf37	0xbd6b6bd6	0xfc5454a8	0x1d0b0b16	0x49e0e0a9	0x48383870
0xC8	0x0c040408	0x31c8c8f9	0xce9d9d53	0x40e7e7a7	0x3c141428	0xbab1b10b	0xe0878767	0xcd9c9c51
0xD0	0x08dfdfd7	0xb16f6fde	0x62f9f99b	0x07dadadd	0x7e2a2a54	0x25c4c4e1	0xeb5959b2	0x3a16162c
0xD8	0x9c7474e8	0xda91914b	0x94abab3f	0x6a26264c	0xa36161c2	0x9a7676ec	0x5c343468	0x7d2b2b56
0xE0	0x9eadad33	0xc299995b	0x64fbfb9f	0x967272e4	0x5dececb1	0x55333366	0x36121224	0x0bdeded5
0xE8	0xc1989859	0x4d3b3b76	0x29c0c0e9	0xc49b9b5f	0x423e3e7c	0x28181830	0x30101020	0x4e3a3a74
0xF0	0xfa5656ac	0x4ae1e1ab	0x997777ee	0x32c9c9fb	0x221e1e3c	0xcb9e9e55	0xd6959543	0x8ca3a32f
0xF8	0xd9909049	0x2b191932	0x91a8a839	0xb46c6cd8	0x1b090912	0x19d0d0c9	0x79f0f089	0xe3868665

Table 7: S2_T0

S2_T1:

0x00	0x25254a6f	0x2424486c	0x7373e695	0x6767cea9	0xd7d7c710	0xaeae359b	0x5c5cb8e4	0x30306050
0x08	0xa4a42185	0xeeeeb55b	0x6e6edcb2	0xcbcbff34	0x7d7dfa87	0xb5b503b6	0x82826def	0xdbdbdf04
0x10	0xe4e4a145	0x8e8e75fb	0x484890d8	0x494992db	0x4f4f9ed1	0x5d5dbae7	0x6a6ad4be	0x7878f088
0x18	0x7070e090	0x888879f1	0xe8e8b951	0x5f5fbee1	0x5e5ebce2	0x848461e5	0x6565caaf	0xe2e2ad4f
0x20	0xd8d8d901	0xe9e9bb52	0xccccf13d	0xededb35e	0x404080c0	0x2f2f5e71	0x11112233	0x28285078
0x28	0x5757aef9	0xd2d2cd1f	0xacac319d	0xe3e3af4c	0x4a4a94de	0x15152a3f	0x1b1b362d	0xb9b91ba2
0x30	0xb2b20dbf	0x808069e9	0x858563e6	0xa6a62583	0x2e2e5c72	0x02020406	0x47478ec9	0x2929527b
0x38	0x07070e09	0x4b4b96dd	0x0e0e1c12	0xc1c1eb2a	0x5151a2f3	0xaaaa3d97	0x89897bf2	0xd4d4c115
0x40	0xcacafd37	0x01010203	0x46468cca	0xb3b30fbc	0xefefb758	0xddddd30e	0x444488cc	0x7b7bf68d
0x48	0xc2c2ed2f	0x7f7ffe81	0xbebe15ab	0xc3c3ef2c	0x9f9f57c8	0x20204060	0x4c4c98d4	0x6464c8ac
0x50	0x83836fec	0xa2a22d8f	0x6868d0b8	0x424284c6	0x13132635	0xb4b401b5	0x414182c3	0xcdcdf33e
0x58	0xbaba1da7	0xc6c6e523	0xbbbb1fa4	0x6d6ddab7	0x4d4d9ad7	0x7171e293	0x21214263	0xf4f48175
0x60	0x8d8d73fe	0xb0b009b9	0xe5e5a346	0x93934fdc	0xfefe956b	0x8f8f77f8	0xe6e6a543	0xcfcff738
0x68	0x434386c5	0x45458acf	0x31316253	0x22224466	0x37376e59	0x36366c5a	0x969645d3	0xfafa9d67
0x70	0xbcbc11ad	0x0f0f1e11	0x08081018	0x5252a4f6	0x1d1d3a27	0x5555aaff	0x1a1a342e	0xc5c5e326
0x78	0x4e4e9cd2	0x23234665	0x6969d2bb	0x7a7af48e	0x92924ddf	0xffff9768	0x5b5bb6ed	0x5a5ab4ee
0x80	0xebebbf54	0x9a9a5dc7	0x1c1c3824	0xa9a93b92	0xd1d1cb1a	0x7e7efc82	0x0d0d1a17	0xfcfc916d
0x88	0x5050a0f0	0x8a8a7df7	0xb6b605b3	0x6262c4a6	0xf5f58376	0x0a0a141e	0xf8f89961	0xdcdcd10d
0x90	0x03030605	0x3c3c7844	0x0c0c1814	0x3939724b	0xf1f18b7a	0xb8b819a1	0xf3f38f7c	0x3d3d7a47
0x98	0xf2f28d7f	0xd5d5c316	0x979747d0	0x6666ccaa	0x81816bea	0x32326456	0xa0a02989	0x00000000
0xA0	0x06060c0a	0xcecef53b	0xf6f68573	0xeaeabd57	0xb7b707b0	0x17172e39	0xf7f78770	0x8c8c71fd
0xA8	0x7979f28b	0xd6d6c513	0xa7a72780	0xbfbf17a8	0x8b8b7ff4	0x3f3f7e41	0x1f1f3e21	0x5353a6f5
0xB0	0x6363c6a5	0x7575ea9f	0x35356a5f	0x2c2c5874	0x6060c0a0	0xfdfd936e	0x27274e69	0xd3d3cf1c
0xB8	0x949441d5	0xa5a52386	0x7c7cf884	0xa1a12b8a	0x05050a0f	0x5858b0e8	0x2d2d5a77	0xbdbd13ae
0xC0	0xd9d9db02	0xc7c7e720	0xafaf3798	0x6b6bd6bd	0x5454a8fc	0x0b0b161d	0xe0e0a949	0x38387048
0xC8	0x0404080c	0xc8c8f931	0x9d9d53ce	0xe7e7a740	0x1414283c	0xb1b10bba	0x878767e0	0x9c9c51cd
0xD0	0xdfdfd708	0x6f6fdeb1	0xf9f99b62	0xdadadd07	0x2a2a547e	0xc4c4e125	0x5959b2eb	0x16162c3a
0xD8	0x7474e89c	0x91914bda	0xabab3f94	0x26264c6a	0x6161c2a3	0x7676ec9a	0x3434685c	0x2b2b567d
0xE0	0xadad339e	0x99995bc2	0xfbfb9f64	0x7272e496	0xececb15d	0x33336655	0x12122436	0xdeded50b
0xE8	0x989859c1	0x3b3b764d	0xc0c0e929	0x9b9b5fc4	0x3e3e7c42	0x18183028	0x10102030	0x3a3a744e
0xF0	0x5656acfa	0xe1e1ab4a	0x7777ee99	0xc9c9fb32	0x1e1e3c22	0x9e9e55cb	0x959543d6	0xa3a32f8c
0xF8	0x909049d9	0x1919322b	0xa8a83991	0x6c6cd8b4	0x0909121b	0xd0d0c919	0xf0f08979	0x868665e3

Table 8: S2_T1

S2_T2:

0x00	0x254a6f25	0x24486c24	0x73e69573	0x67cea967	0xd7c710d7	0xae359bae	0x5cb8e45c	0x30605030
0x08	0xa42185a4	0xeeb55bee	0x6edcb26e	0xcbff34cb	0x7dfa877d	0xb503b6b5	0x826def82	0xdbdf04db
0x10	0xe4a145e4	0x8e75fb8e	0x4890d848	0x4992db49	0x4f9ed14f	0x5dbae75d	0x6ad4be6a	0x78f08878
0x18	0x70e09070	0x8879f188	0xe8b951e8	0x5fbee15f	0x5ebce25e	0x8461e584	0x65caaf65	0xe2ad4fe2
0x20	0xd8d901d8	0xe9bb52e9	0xccf13dcc	0xedb35eed	0x4080c040	0x2f5e712f	0x11223311	0x28507828
0x28	0x57aef957	0xd2cd1fd2	0xac319dac	0xe3af4ce3	0x4a94de4a	0x152a3f15	0x1b362d1b	0xb91ba2b9
0x30	0xb20dbfb2	0x8069e980	0x8563e685	0xa62583a6	0x2e5c722e	0x02040602	0x478ec947	0x29527b29
0x38	0x070e0907	0x4b96dd4b	0x0e1c120e	0xc1eb2ac1	0x51a2f351	0xaa3d97aa	0x897bf289	0xd4c115d4
0x40	0xcafd37ca	0x01020301	0x468cca46	0xb30fbcb3	0xefb758ef	0xddd30edd	0x4488cc44	0x7bf68d7b
0x48	0xc2ed2fc2	0x7ffe817f	0xbe15abbe	0xc3ef2cc3	0x9f57c89f	0x20406020	0x4c98d44c	0x64c8ac64
0x50	0x836fec83	0xa22d8fa2	0x68d0b868	0x4284c642	0x13263513	0xb401b5b4	0x4182c341	0xcdf33ecd
0x58	0xba1da7ba	0xc6e523c6	0xbb1fa4bb	0x6ddab76d	0x4d9ad74d	0x71e29371	0x21426321	0xf48175f4
0x60	0x8d73fe8d	0xb009b9b0	0xe5a346e5	0x934fdc93	0xfe956bfe	0x8f77f88f	0xe6a543e6	0xcff738cf
0x68	0x4386c543	0x458acf45	0x31625331	0x22446622	0x376e5937	0x366c5a36	0x9645d396	0xfa9d67fa
0x70	0xbc11adbc	0x0f1e110f	0x08101808	0x52a4f652	0x1d3a271d	0x55aaff55	0x1a342e1a	0xc5e326c5
0x78	0x4e9cd24e	0x23466523	0x69d2bb69	0x7af48e7a	0x924ddf92	0xff9768ff	0x5bb6ed5b	0x5ab4ee5a
0x80	0xebbf54eb	0x9a5dc79a	0x1c38241c	0xa93b92a9	0xd1cb1ad1	0x7efc827e	0x0d1a170d	0xfc916dfc
0x88	0x50a0f050	0x8a7df78a	0xb605b3b6	0x62c4a662	0xf58376f5	0x0a141e0a	0xf89961f8	0xdcd10ddc
0x90	0x03060503	0x3c78443c	0x0c18140c	0x39724b39	0xf18b7af1	0xb819a1b8	0xf38f7cf3	0x3d7a473d
0x98	0xf28d7ff2	0xd5c316d5	0x9747d097	0x66ccaa66	0x816bea81	0x32645632	0xa02989a0	0x00000000
0xA0	0x060c0a06	0xcef53bce	0xf68573f6	0xeabd57ea	0xb707b0b7	0x172e3917	0xf78770f7	0x8c71fd8c
0xA8	0x79f28b79	0xd6c513d6	0xa72780a7	0xbf17a8bf	0x8b7ff48b	0x3f7e413f	0x1f3e211f	0x53a6f553
0xB0	0x63c6a563	0x75ea9f75	0x356a5f35	0x2c58742c	0x60c0a060	0xfd936efd	0x274e6927	0xd3cf1cd3
0xB8	0x9441d594	0xa52386a5	0x7cf8847c	0xa12b8aa1	0x050a0f05	0x58b0e858	0x2d5a772d	0xbd13aebd
0xC0	0xd9db02d9	0xc7e720c7	0xaf3798af	0x6bd6bd6b	0x54a8fc54	0x0b161d0b	0xe0a949e0	0x38704838
0xC8	0x04080c04	0xc8f931c8	0x9d53ce9d	0xe7a740e7	0x14283c14	0xb10bbab1	0x8767e087	0x9c51cd9c
0xD0	0xdfd708df	0x6fdeb16f	0xf99b62f9	0xdadd07da	0x2a547e2a	0xc4e125c4	0x59b2eb59	0x162c3a16
0xD8	0x74e89c74	0x914bda91	0xab3f94ab	0x264c6a26	0x61c2a361	0x76ec9a76	0x34685c34	0x2b567d2b
0xE0	0xad339ead	0x995bc299	0xfb9f64fb	0x72e49672	0xecb15dec	0x33665533	0x12243612	0xded50bde
0xE8	0x9859c198	0x3b764d3b	0xc0e929c0	0x9b5fc49b	0x3e7c423e	0x18302818	0x10203010	0x3a744e3a
0xF0	0x56acfa56	0xe1ab4ae1	0x77ee9977	0xc9fb32c9	0x1e3c221e	0x9e55cb9e	0x9543d695	0xa32f8ca3
0xF8	0x9049d990	0x19322b19	0xa83991a8	0x6cd8b46c	0x09121b09	0xd0c919d0	0xf08979f0	0x8665e386

Table 9: S2_T2

S2_T3:

0x00	0x4a6f2525	0x486c2424	0xe6957373	0xcea96767	0xc710d7d7	0x359baeae	0xb8e45c5c	0x60503030
0x08	0x2185a4a4	0xb55beeee	0xdcb26e6e	0xff34cbcb	0xfa877d7d	0x03b6b5b5	0x6def8282	0xdf04dbdb
0x10	0xa145e4e4	0x75fb8e8e	0x90d84848	0x92db4949	0x9ed14f4f	0xbae75d5d	0xd4be6a6a	0xf0887878
0x18	0xe0907070	0x79f18888	0xb951e8e8	0xbee15f5f	0xbce25e5e	0x61e58484	0xcaaf6565	0xad4fe2e2
0x20	0xd901d8d8	0xbb52e9e9	0xf13dcccc	0xb35eeded	0x80c04040	0x5e712f2f	0x22331111	0x50782828
0x28	0xaef95757	0xcd1fd2d2	0x319dacac	0xaf4ce3e3	0x94de4a4a	0x2a3f1515	0x362d1b1b	0x1ba2b9b9
0x30	0x0dbfb2b2	0x69e98080	0x63e68585	0x2583a6a6	0x5c722e2e	0x04060202	0x8ec94747	0x527b2929
0x38	0x0e090707	0x96dd4b4b	0x1c120e0e	0xeb2ac1c1	0xa2f35151	0x3d97aaaa	0x7bf28989	0xc115d4d4
0x40	0xfd37caca	0x02030101	0x8cca4646	0x0fbcb3b3	0xb758efef	0xd30edddd	0x88cc4444	0xf68d7b7b
0x48	0xed2fc2c2	0xfe817f7f	0x15abbebe	0xef2cc3c3	0x57c89f9f	0x40602020	0x98d44c4c	0xc8ac6464
0x50	0x6fec8383	0x2d8fa2a2	0xd0b86868	0x84c64242	0x26351313	0x01b5b4b4	0x82c34141	0xf33ecdcd
0x58	0x1da7baba	0xe523c6c6	0x1fa4bbbb	0xdab76d6d	0x9ad74d4d	0xe2937171	0x42632121	0x8175f4f4
0x60	0x73fe8d8d	0x09b9b0b0	0xa346e5e5	0x4fdc9393	0x956bfefe	0x77f88f8f	0xa543e6e6	0xf738cfcf
0x68	0x86c54343	0x8acf4545	0x62533131	0x44662222	0x6e593737	0x6c5a3636	0x45d39696	0x9d67fafa
0x70	0x11adbcbc	0x1e110f0f	0x10180808	0xa4f65252	0x3a271d1d	0xaaff5555	0x342e1a1a	0xe326c5c5
0x78	0x9cd24e4e	0x46652323	0xd2bb6969	0xf48e7a7a	0x4ddf9292	0x9768ffff	0xb6ed5b5b	0xb4ee5a5a
0x80	0xbf54ebeb	0x5dc79a9a	0x38241c1c	0x3b92a9a9	0xcb1ad1d1	0xfc827e7e	0x1a170d0d	0x916dfcfc
0x88	0xa0f05050	0x7df78a8a	0x05b3b6b6	0xc4a66262	0x8376f5f5	0x141e0a0a	0x9961f8f8	0xd10ddcdc
0x90	0x06050303	0x78443c3c	0x18140c0c	0x724b3939	0x8b7af1f1	0x19a1b8b8	0x8f7cf3f3	0x7a473d3d
0x98	0x8d7ff2f2	0xc316d5d5	0x47d09797	0xccaa6666	0x6bea8181	0x64563232	0x2989a0a0	0x00000000
0xA0	0x0c0a0606	0xf53bcece	0x8573f6f6	0xbd57eaea	0x07b0b7b7	0x2e391717	0x8770f7f7	0x71fd8c8c
0xA8	0xf28b7979	0xc513d6d6	0x2780a7a7	0x17a8bfbf	0x7ff48b8b	0x7e413f3f	0x3e211f1f	0xa6f55353
0xB0	0xc6a56363	0xea9f7575	0x6a5f3535	0x58742c2c	0xc0a06060	0x936efdfd	0x4e692727	0xcf1cd3d3
0xB8	0x41d59494	0x2386a5a5	0xf8847c7c	0x2b8aa1a1	0x0a0f0505	0xb0e85858	0x5a772d2d	0x13aebdbd
0xC0	0xdb02d9d9	0xe720c7c7	0x3798afaf	0xd6bd6b6b	0xa8fc5454	0x161d0b0b	0xa949e0e0	0x70483838
0xC8	0x080c0404	0xf931c8c8	0x53ce9d9d	0xa740e7e7	0x283c1414	0x0bbab1b1	0x67e08787	0x51cd9c9c
0xD0	0xd708dfdf	0xdeb16f6f	0x9b62f9f9	0xdd07dada	0x547e2a2a	0xe125c4c4	0xb2eb5959	0x2c3a1616
0xD8	0xe89c7474	0x4bda9191	0x3f94abab	0x4c6a2626	0xc2a36161	0xec9a7676	0x685c3434	0x567d2b2b
0xE0	0x339eadad	0x5bc29999	0x9f64fbfb	0xe4967272	0xb15decec	0x66553333	0x24361212	0xd50bdede
0xE8	0x59c19898	0x764d3b3b	0xe929c0c0	0x5fc49b9b	0x7c423e3e	0x30281818	0x20301010	0x744e3a3a
0xF0	0xacfa5656	0xab4ae1e1	0xee997777	0xfb32c9c9	0x3c221e1e	0x55cb9e9e	0x43d69595	0x2f8ca3a3
0xF8	0x49d99090	0x322b1919	0x3991a8a8	0xd8b46c6c	0x121b0909	0xc919d0d0	0x8979f0f0	0x65e38686

Table 10: S2_T3

2.5. Definitions of tables for the LFSR

MUL_{alpha} is defined by the following table:

0x00	0x00000000	0xE19FCF13	0x6B973726	0x8A08F835	0xD6876E4C	0x3718A15F	0xBD10596A	0x5C8F9679
0x08	0x05A7DC98	0xE438138B	0x6E30EBBE	0x8FAF24AD	0xD320B2D4	0x32BF7DC7	0xB8B785F2	0x59284AE1
0x10	0x0AE71199	0xEB78DE8A	0x617026BF	0x80EFE9AC	0xDC607FD5	0x3DFFB0C6	0xB7F748F3	0x566887E0
0x18	0x0F40CD01	0xEEDF0212	0x64D7FA27	0x85483534	0xD9C7A34D	0x38586C5E	0xB250946B	0x53CF5B78
0x20	0x1467229B	0xF5F8ED88	0x7FF015BD	0x9E6FDAAE	0xC2E04CD7	0x237F83C4	0xA9777BF1	0x48E8B4E2
0x28	0x11C0FE03	0xF05F3110	0x7A57C925	0x9BC80636	0xC747904F	0x26D85F5C	0xACD0A769	0x4D4F687A
0x30	0x1E803302	0xFF1FFC11	0x75170424	0x9488CB37	0xC8075D4E	0x2998925D	0xA3906A68	0x420FA57B
0x38	0x1B27EF9A	0xFAB82089	0x70B0D8BC	0x912F17AF	0xCDA081D6	0x2C3F4EC5	0xA637B6F0	0x47A879E3
0x40	0x28CE449F	0xC9518B8C	0x435973B9	0xA2C6BCAA	0xFE492AD3	0x1FD6E5C0	0x95DE1DF5	0x7441D2E6
0x48	0x2D699807	0xCCF65714	0x46FEAF21	0xA7616032	0xFBEEF64B	0x1A713958	0x9079C16D	0x71E60E7E
0x50	0x22295506	0xC3B69A15	0x49BE6220	0xA821AD33	0xF4AE3B4A	0x1531F459	0x9F390C6C	0x7EA6C37F
0x58	0x278E899E	0xC611468D	0x4C19BEB8	0xAD8671AB	0xF109E7D2	0x109628C1	0x9A9ED0F4	0x7B011FE7
0x60	0x3CA96604	0xDD36A917	0x573E5122	0xB6A19E31	0xEA2E0848	0x0BB1C75B	0x81B93F6E	0x6026F07D
0x68	0x390EBA9C	0xD891758F	0x52998DBA	0xB30642A9	0xEF89D4D0	0x0E161BC3	0x841EE3F6	0x65812CE5
0x70	0x364E779D	0xD7D1B88E	0x5DD940BB	0xBC468FA8	0xE0C919D1	0x0156D6C2	0x8B5E2EF7	0x6AC1E1E4
0x78	0x33E9AB05	0xD2766416	0x587E9C23	0xB9E15330	0xE56EC549	0x04F10A5A	0x8EF9F26F	0x6F663D7C
0x80	0x50358897	0xB1AA4784	0x3BA2BFB1	0xDA3D70A2	0x86B2E6DB	0x672D29C8	0xED25D1FD	0x0CBA1EEE
0x88	0x5592540F	0xB40D9B1C	0x3E056329	0xDF9AAC3A	0x83153A43	0x628AF550	0xE8820D65	0x091DC276
0x90	0x5AD2990E	0xBB4D561D	0x3145AE28	0xD0DA613B	0x8C55F742	0x6DCA3851	0xE7C2C064	0x065D0F77
0x98	0x5F754596	0xBEEA8A85	0x34E272B0	0xD57DBDA3	0x89F22BDA	0x686DE4C9	0xE2651CFC	0x03FAD3EF
0xA0	0x4452AA0C	0xA5CD651F	0x2FC59D2A	0xCE5A5239	0x92D5C440	0x734A0B53	0xF942F366	0x18DD3C75
0xA8	0x41F57694	0xA06AB987	0x2A6241B2	0xCBFD8EA1	0x977218D8	0x76EDD7CB	0xFCE52FFE	0x1D7AE0ED
0xB0	0x4EB5BB95	0xAF2A7486	0x25228CB3	0xC4BD43A0	0x9832D5D9	0x79AD1ACA	0xF3A5E2FF	0x123A2DEC
0xB8	0x4B12670D	0xAA8DA81E	0x2085502B	0xC11A9F38	0x9D950941	0x7C0AC652	0xF6023E67	0x179DF174
0xC0	0x78FBCC08	0x9964031B	0x136CFB2E	0xF2F3343D	0xAE7CA244	0x4FE36D57	0xC5EB9562	0x24745A71
0xC8	0x7D5C1090	0x9CC3DF83	0x16CB27B6	0xF754E8A5	0xABDB7EDC	0x4A44B1CF	0xC04C49FA	0x21D386E9
0xD0	0x721CDD91	0x93831282	0x198BEAB7	0xF81425A4	0xA49BB3DD	0x45047CCE	0xCF0C84FB	0x2E934BE8
0xD8	0x77BB0109	0x9624CE1A	0x1C2C362F	0xFDB3F93C	0xA13C6F45	0x40A3A056	0xCAAB5863	0x2B349770
0xE0	0x6C9CEE93	0x8D032180	0x070BD9B5	0xE69416A6	0xBA1B80DF	0x5B844FCC	0xD18CB7F9	0x301378EA
0xE8	0x693B320B	0x88A4FD18	0x02AC052D	0xE333CA3E	0xBFBC5C47	0x5E239354	0xD42B6B61	0x35B4A472
0xF0	0x667BFF0A	0x87E43019	0x0DECC82C	0xEC73073F	0xB0FC9146	0x51635E55	0xDB6BA660	0x3AF46973
0xF8	0x63DC2392	0x8243EC81	0x084B14B4	0xE9D4DBA7	0xB55B4DDE	0x54C482CD	0xDECC7AF8	0x3F53B5EB

Table 11: MUL_{alpha}

Example: $MUL_{alpha}(0x2A) = 0x7A57C925$.

DIV_{alpha} is defined by the following table:

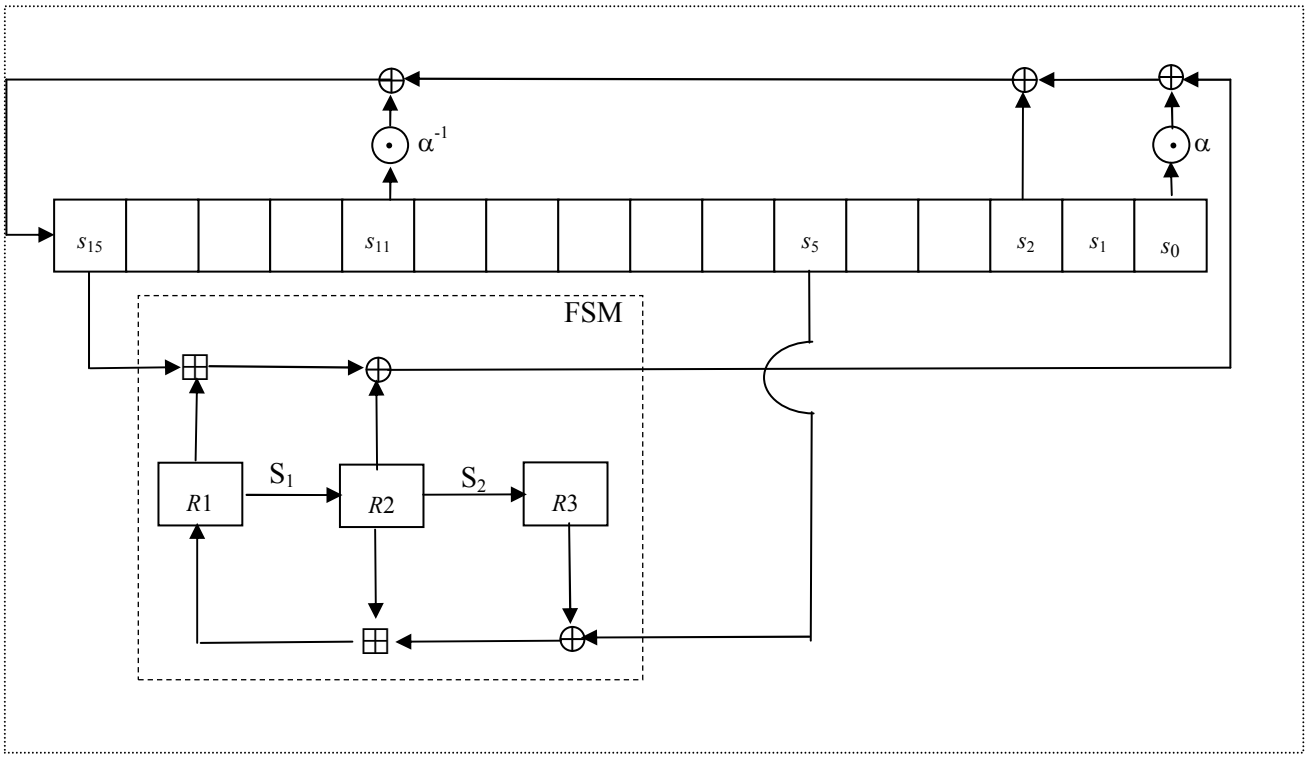
0x00	0x00000000	0x180F40CD	0x301E8033	0x2811C0FE	0x603CA966	0x7833E9AB	0x50222955	0x482D6998
0x08	0xC078FBCC	0xD877BB01	0xF0667BFF	0xE8693B32	0xA04452AA	0xB84B1267	0x905AD299	0x88559254
0x10	0x29F05F31	0x31FF1FFC	0x19EEDF02	0x01E19FCF	0x49CCF657	0x51C3B69A	0x79D27664	0x61DD36A9
0x18	0xE988A4FD	0xF187E430	0xD99624CE	0xC1996403	0x89B40D9B	0x91BB4D56	0xB9AA8DA8	0xA1A5CD65
0x20	0x5249BE62	0x4A46FEAF	0x62573E51	0x7A587E9C	0x32751704	0x2A7A57C9	0x026B9737	0x1A64D7FA
0x28	0x923145AE	0x8A3E0563	0xA22FC59D	0xBA208550	0xF20DECC8	0xEA02AC05	0xC2136CFB	0xDA1C2C36
0x30	0x7BB9E153	0x63B6A19E	0x4BA76160	0x53A821AD	0x1B854835	0x038A08F8	0x2B9BC806	0x339488CB
0x38	0xBBC11A9F	0xA3CE5A52	0x8BDF9AAC	0x93D0DA61	0xDBFDB3F9	0xC3F2F334	0xEBE333CA	0xF3EC7307
0x40	0xA492D5C4	0xBC9D9509	0x948C55F7	0x8C83153A	0xC4AE7CA2	0xDCA13C6F	0xF4B0FC91	0xECBFBC5C
0x48	0x64EA2E08	0x7CE56EC5	0x54F4AE3B	0x4CFBEEF6	0x04D6876E	0x1CD9C7A3	0x34C8075D	0x2CC74790
0x50	0x8D628AF5	0x956DCA38	0xBD7C0AC6	0xA5734A0B	0xED5E2393	0xF551635E	0xDD40A3A0	0xC54FE36D
0x58	0x4D1A7139	0x551531F4	0x7D04F10A	0x650BB1C7	0x2D26D85F	0x35299892	0x1D38586C	0x053718A1
0x60	0xF6DB6BA6	0xEED42B6B	0xC6C5EB95	0xDECAAB58	0x96E7C2C0	0x8EE8820D	0xA6F942F3	0xBEF6023E
0x68	0x36A3906A	0x2EACD0A7	0x06BD1059	0x1EB25094	0x569F390C	0x4E9079C1	0x6681B93F	0x7E8EF9F2
0x70	0xDF2B3497	0xC724745A	0xEF35B4A4	0xF73AF469	0xBF179DF1	0xA718DD3C	0x8F091DC2	0x97065D0F
0x78	0x1F53CF5B	0x075C8F96	0x2F4D4F68	0x37420FA5	0x7F6F663D	0x676026F0	0x4F71E60E	0x577EA6C3
0x80	0xE18D0321	0xF98243EC	0xD1938312	0xC99CC3DF	0x81B1AA47	0x99BEEA8A	0xB1AF2A74	0xA9A06AB9
0x88	0x21F5F8ED	0x39FAB820	0x11EB78DE	0x09E43813	0x41C9518B	0x59C61146	0x71D7D1B8	0x69D89175
0x90	0xC87D5C10	0xD0721CDD	0xF863DC23	0xE06C9CEE	0xA841F576	0xB04EB5BB	0x985F7545	0x80503588
0x98	0x0805A7DC	0x100AE711	0x381B27EF	0x20146722	0x68390EBA	0x70364E77	0x58278E89	0x4028CE44
0xA0	0xB3C4BD43	0xABCBFD8E	0x83DA3D70	0x9BD57DBD	0xD3F81425	0xCBF754E8	0xE3E69416	0xFBE9D4DB
0xA8	0x73BC468F	0x6BB30642	0x43A2C6BC	0x5BAD8671	0x1380EFE9	0x0B8FAF24	0x239E6FDA	0x3B912F17
0xB0	0x9A34E272	0x823BA2BF	0xAA2A6241	0xB225228C	0xFA084B14	0xE2070BD9	0xCA16CB27	0xD2198BEA
0xB8	0x5A4C19BE	0x42435973	0x6A52998D	0x725DD940	0x3A70B0D8	0x227FF015	0x0A6E30EB	0x12617026
0xC0	0x451FD6E5	0x5D109628	0x750156D6	0x6D0E161B	0x25237F83	0x3D2C3F4E	0x153DFFB0	0x0D32BF7D
0xC8	0x85672D29	0x9D686DE4	0xB579AD1A	0xAD76EDD7	0xE55B844F	0xFD54C482	0xD545047C	0xCD4A44B1
0xD0	0x6CEF89D4	0x74E0C919	0x5CF109E7	0x44FE492A	0x0CD320B2	0x14DC607F	0x3CCDA081	0x24C2E04C
0xD8	0xAC977218	0xB49832D5	0x9C89F22B	0x8486B2E6	0xCCABDB7E	0xD4A49BB3	0xFCB55B4D	0xE4BA1B80
0xE0	0x17566887	0x0F59284A	0x2748E8B4	0x3F47A879	0x776AC1E1	0x6F65812C	0x477441D2	0x5F7B011F
0xE8	0xD72E934B	0xCF21D386	0xE7301378	0xFF3F53B5	0xB7123A2D	0xAF1D7AE0	0x870CBA1E	0x9F03FAD3
0xF0	0x3EA637B6	0x26A9777B	0x0EB8B785	0x16B7F748	0x5E9A9ED0	0x4695DE1D	0x6E841EE3	0x768B5E2E
0xF8	0xFEDECC7A	0xE6D18CB7	0xCEC04C49	0xD6CF0C84	0x9EE2651C	0x86ED25D1	0xAEFCE52F	0xB6F3A5E2

Table 12: DIV_{alpha}

ANNEX 3
Figures of the SNOW 3G Algorithm

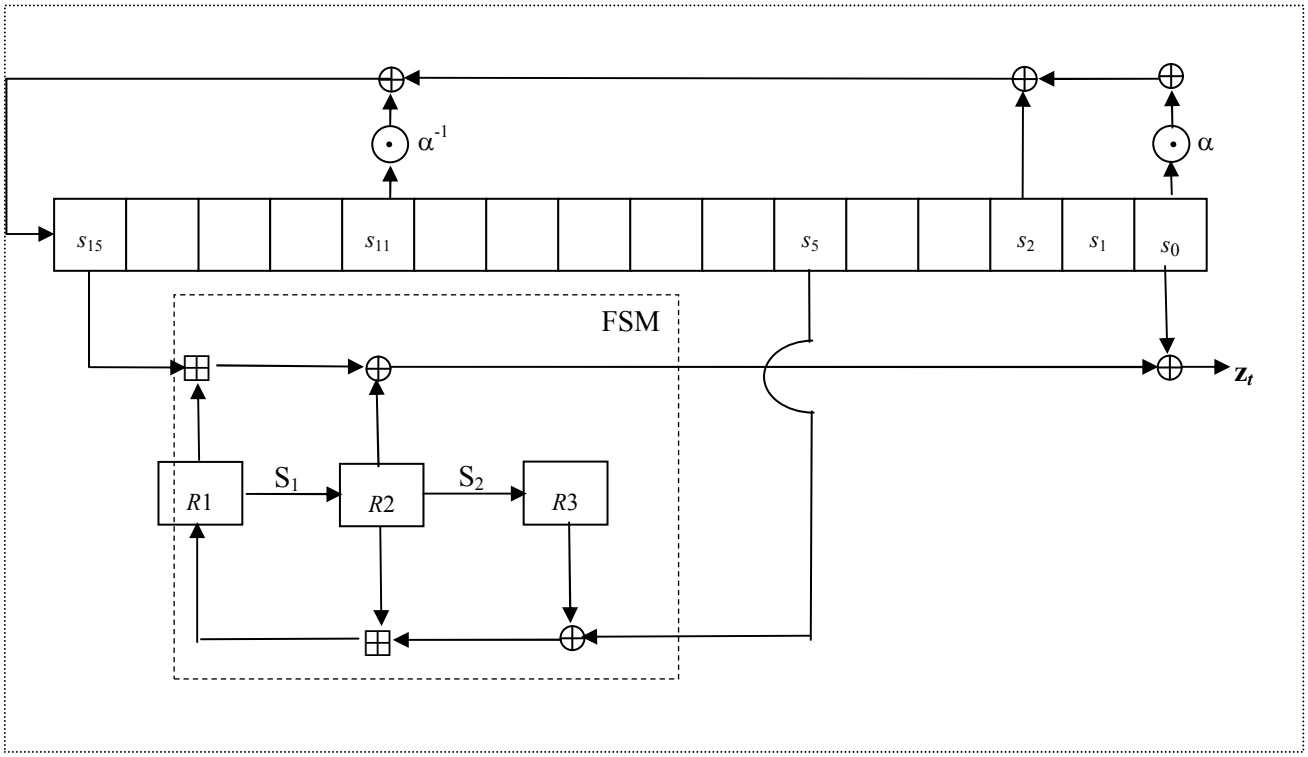
SNOW 3G Algorithm during key initialisation

This illustrates the operation of **SNOW 3G** described in chapter 3.4.4 of the normative part.



SNOW 3G Algorithm during keystream-generation

This illustrates the operation of **SNOW 3G** described in chapter 3.4.5 of the normative part.



ANNEX 4
Simulation Program Listing

4.1. Header file

```
/*-------------------------------------------------------
 *                   SNOW_3G.h
 *-------------------------------------------------------*/
typedef unsigned char u8;
typedef unsigned int u32;
typedef unsigned long long u64;


/* Initialization.
 * Input k[4]: Four 32-bit words making up 128-bit key.
 * Input IV[4]: Four 32-bit words making 128-bit initialization variable.
 * Output: All the LFSRs and FSM are initialized for key generation.
 * See Section 4.1.
 */

void Initialize(u32 k[4], u32 IV[4]);


/* Generation of Keystream.
 * input n: number of 32-bit words of keystream.
 * input z: space for the generated keystream, assumes
 *          memory is allocated already.
 * output: generated keystream which is filled in z
 * See section 4.2.
 */

void GenerateKeystream(u32 n, u32 *z);
```

4.2. Code

```
/*-----------------------------------------------------------------------
 *               SNOW_3G.c
 *-----------------------------------------------------------------------*/

#include "SNOW_3G.h"

/* LFSR */
u32 LFSR_S0 = 0x00;
u32 LFSR_S1 = 0x00;
u32 LFSR_S2 = 0x00;
u32 LFSR_S3 = 0x00;
u32 LFSR_S4 = 0x00;
u32 LFSR_S5 = 0x00;
u32 LFSR_S6 = 0x00;
u32 LFSR_S7 = 0x00;
u32 LFSR_S8 = 0x00;
u32 LFSR_S9 = 0x00;
u32 LFSR_S10 = 0x00;
u32 LFSR_S11 = 0x00;
u32 LFSR_S12 = 0x00;
u32 LFSR_S13 = 0x00;
u32 LFSR_S14 = 0x00;
u32 LFSR_S15 = 0x00;

/* FSM */

u32 FSM_R1 = 0x00;
u32 FSM_R2 = 0x00;
u32 FSM_R3 = 0x00;
```

```
/* Rijndael S-box SR */

u8 SR[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
};

/* S-box SQ */

u8 SQ[256] = {
0x25,0x24,0x73,0x67,0xD7,0xAE,0x5C,0x30,0xA4,0xEE,0x6E,0xCB,0x7D,0xB5,0x82,0xDB,
0xE4,0x8E,0x48,0x49,0x4F,0x5D,0x6A,0x78,0x70,0x88,0xE8,0x5F,0x5E,0x84,0x65,0xE2,
0xD8,0xE9,0xCC,0xED,0x40,0x2F,0x11,0x28,0x57,0xD2,0xAC,0xE3,0x4A,0x15,0x1B,0xB9,
0xB2,0x80,0x85,0xA6,0x2E,0x02,0x47,0x29,0x07,0x4B,0x0E,0xC1,0x51,0xAA,0x89,0xD4,
0xCA,0x01,0x46,0xB3,0xEF,0xDD,0x44,0x7B,0xC2,0x7F,0xBE,0xC3,0x9F,0x20,0x4C,0x64,
0x83,0xA2,0x68,0x42,0x13,0xB4,0x41,0xCD,0xBA,0xC6,0xBB,0x6D,0x4D,0x71,0x21,0xF4,
0x8D,0xB0,0xE5,0x93,0xFE,0x8F,0xE6,0xCF,0x43,0x45,0x31,0x22,0x37,0x36,0x96,0xFA,
0xBC,0x0F,0x08,0x52,0x1D,0x55,0x1A,0xC5,0x4E,0x23,0x69,0x7A,0x92,0xFF,0x5B,0x5A,
0xEB,0x9A,0x1C,0xA9,0xD1,0x7E,0x0D,0xFC,0x50,0x8A,0xB6,0x62,0xF5,0x0A,0xF8,0xDC,
0x03,0x3C,0x0C,0x39,0xF1,0xB8,0xF3,0x3D,0xF2,0xD5,0x97,0x66,0x81,0x32,0xA0,0x00,
0x06,0xCE,0xF6,0xEA,0xB7,0x17,0xF7,0x8C,0x79,0xD6,0xA7,0xBF,0x8B,0x3F,0x1F,0x53,
0x63,0x75,0x35,0x2C,0x60,0xFD,0x27,0xD3,0x94,0xA5,0x7C,0xA1,0x05,0x58,0x2D,0xBD,
0xD9,0xC7,0xAF,0x6B,0x54,0x0B,0xE0,0x38,0x04,0xC8,0x9D,0xE7,0x14,0xB1,0x87,0x9C,
0xDF,0x6F,0xF9,0xDA,0x2A,0xC4,0x59,0x16,0x74,0x91,0xAB,0x26,0x61,0x76,0x34,0x2B,
0xAD,0x99,0xFB,0x72,0xEC,0x33,0x12,0xDE,0x98,0x3B,0xC0,0x9B,0x3E,0x18,0x10,0x3A,
0x56,0xE1,0x77,0xC9,0x1E,0x9E,0x95,0xA3,0x90,0x19,0xA8,0x6C,0x09,0xD0,0xF0,0x86
};

/* MULx.
 * Input V: an 8-bit input.
 * Input c: an 8-bit input.
 * Output : an 8-bit output.
 * See section 3.1.1 for details.
 */

u8 MULx(u8 V, u8 c)
{
   if ( V & 0x80 )
    return ( (V << 1) ^ c);
   else
    return ( V << 1);
}

/* MULxPOW.
 * Input V: an 8-bit input.
 * Input i: a positive integer.
 * Input c: an 8-bit input.
 * Output : an 8-bit output.
 * See section 3.1.2 for details.
 */

u8 MULxPOW(u8 V, u8 i, u8 c)
```

```
{
   if ( i == 0)
    return V;
   else
    return MULx( MULxPOW( V, i-1, c ), c);
}

/* The function MUL alpha.
 * Input c: 8-bit input.
 * Output : 32-bit output.
 * See section 3.4.2 for details.
 */

u32 MULalpha(u8 c)
{
   return ( ( ((u32)MULxPOW(c,  23, 0xa9)) << 24 ) |
        ( ((u32)MULxPOW(c, 245, 0xa9)) << 16 ) |
        ( ((u32)MULxPOW(c,  48, 0xa9)) <<  8 ) |
        ( ((u32)MULxPOW(c, 239, 0xa9))       )    ) ;
}

/* The function DIV alpha.
 * Input c: 8-bit input.
 * Output : 32-bit output.
 * See section 3.4.3 for details.
 */

u32 DIValpha(u8 c)
{
   return ( ( ((u32)MULxPOW(c,  16, 0xa9)) << 24 ) |
        ( ((u32)MULxPOW(c,  39, 0xa9)) << 16 ) |
        ( ((u32)MULxPOW(c,   6, 0xa9)) <<  8 ) |
        ( ((u32)MULxPOW(c,  64, 0xa9))       )    ) ;
}

/* The 32x32-bit S-Box S1
 * Input: a 32-bit input.
 * Output: a 32-bit output of S1 box.
 * See section 3.3.1.
 */

u32 S1(u32 w)
{
   u8 r0=0, r1=0, r2=0, r3=0;
   u8 srw0 = SR[ (u8)((w >> 24) & 0xff) ];
   u8 srw1 = SR[ (u8)((w >> 16) & 0xff) ];
   u8 srw2 = SR[ (u8)((w >>  8) & 0xff) ];
   u8 srw3 = SR[ (u8)((w)       & 0xff) ];

   r0 = ( ( MULx( srw0 , 0x1b) )  ^
      ( srw1 ) ^
      ( srw2 ) ^
      ( (MULx( srw3, 0x1b)) ^ srw3 )
    );

   r1 = ( ( ( MULx( srw0 , 0x1b) ) ^ srw0 ) ^
      ( MULx(srw1, 0x1b) ) ^
      ( srw2 ) ^
      ( srw3 )
    );

   r2 = ( ( srw0 ) ^
      ( ( MULx( srw1 , 0x1b) ) ^ srw1 ) ^
      ( MULx(srw2, 0x1b) ) ^
      ( srw3 )
    );

   r3 = ( ( srw0 ) ^
```

```
      ( srw1 ) ^
      ( ( MULx( srw2 , 0x1b) ) ^ srw2 ) ^
      ( MULx( srw3, 0x1b) )
    );

   return ( ( ((u32)r0) << 24 ) | ( ((u32)r1) << 16 ) | ( ((u32)r2) << 8 ) |
( ((u32)r3) ) );
}

/* The 32x32-bit S-Box S2
 * Input: a 32-bit input.
 * Output: a 32-bit output of S2 box.
 * See section 3.3.2.
 */
u32 S2(u32 w)
{
   u8 r0=0, r1=0, r2=0, r3=0;
   u8 sqw0 = SQ[ (u8)((w >> 24) & 0xff) ];
   u8 sqw1 = SQ[ (u8)((w >> 16) & 0xff) ];
   u8 sqw2 = SQ[ (u8)((w >>  8) & 0xff) ];
   u8 sqw3 = SQ[ (u8)((w)       & 0xff) ];

   r0 = ( ( MULx( sqw0 , 0x69) )  ^
          ( sqw1 ) ^
          ( sqw2 ) ^
          ( (MULx( sqw3, 0x69)) ^ sqw3 )
        );

   r1 = ( ( ( MULx( sqw0 , 0x69) ) ^ sqw0 ) ^
          ( MULx(sqw1, 0x69) ) ^
          ( sqw2 ) ^
          ( sqw3 )
        );

   r2 = ( ( sqw0 ) ^
          ( ( MULx( sqw1 , 0x69) ) ^ sqw1 ) ^
          ( MULx(sqw2, 0x69) ) ^
          ( sqw3 )
        );

   r3 = ( ( sqw0 ) ^
          ( sqw1 ) ^
          ( ( MULx( sqw2 , 0x69) ) ^ sqw2 ) ^
          ( MULx( sqw3, 0x69) )
        );

   return ( ( ((u32)r0) << 24 ) | ( ((u32)r1) << 16 ) | ( ((u32)r2) << 8 ) |
( ((u32)r3) ) );
}

/* Clocking LFSR in initialization mode.
 * LFSR Registers S0 to S15 are updated as the LFSR receives a single clock.
 * Input F: a 32-bit word comes from output of FSM.
 * See section 3.4.4.
 */

void ClockLFSRInitializationMode(u32 F)
{
   u32 v = ( ( (LFSR_S0 << 8) & 0xffffff00 )         ^
             ( MULalpha( (u8)((LFSR_S0>>24) & 0xff) ) )   ^
             ( LFSR_S2 )                             ^
             ( (LFSR_S11 >> 8)  & 0x00ffffff )        ^
             ( DIValpha( (u8)( ( LFSR_S11) & 0xff ) ) ) ^
           ( F )
         );
   LFSR_S0 = LFSR_S1;
   LFSR_S1 = LFSR_S2;
   LFSR_S2 = LFSR_S3;
```

```
   LFSR_S3 = LFSR_S4;
   LFSR_S4 = LFSR_S5;
   LFSR_S5 = LFSR_S6;
   LFSR_S6 = LFSR_S7;
   LFSR_S7 = LFSR_S8;
   LFSR_S8 = LFSR_S9;
   LFSR_S9 = LFSR_S10;
   LFSR_S10 = LFSR_S11;
   LFSR_S11 = LFSR_S12;
   LFSR_S12 = LFSR_S13;
   LFSR_S13 = LFSR_S14;
   LFSR_S14 = LFSR_S15;
   LFSR_S15 = v;
}

/* Clocking LFSR in keystream mode.
 * LFSR Registers S0 to S15 are updated as the LFSR receives a single clock.
 * See section 3.4.5.
 */

void ClockLFSRKeyStreamMode()
{
   u32 v = ( ( (LFSR_S0 << 8) & 0xffffff00 )          ^
             ( MULalpha( (u8)((LFSR_S0>>24) & 0xff) ) )    ^
             ( LFSR_S2 )                              ^
             ( (LFSR_S11 >> 8)  & 0x00ffffff )         ^
             ( DIValpha( (u8)( ( LFSR_S11) & 0xff ) ) )
       );
   LFSR_S0 = LFSR_S1;
   LFSR_S1 = LFSR_S2;
   LFSR_S2 = LFSR_S3;
   LFSR_S3 = LFSR_S4;
   LFSR_S4 = LFSR_S5;
   LFSR_S5 = LFSR_S6;
   LFSR_S6 = LFSR_S7;
   LFSR_S7 = LFSR_S8;
   LFSR_S8 = LFSR_S9;
   LFSR_S9 = LFSR_S10;
   LFSR_S10 = LFSR_S11;
   LFSR_S11 = LFSR_S12;
   LFSR_S12 = LFSR_S13;
   LFSR_S13 = LFSR_S14;
   LFSR_S14 = LFSR_S15;
   LFSR_S15 = v;
}

/* Clocking FSM.
 * Produces a 32-bit word F.
 * Updates FSM registers R1, R2, R3.
 * See Section 3.4.6.
 */

u32 ClockFSM()
{
   u32 F = ( ( LFSR_S15 + FSM_R1 ) & 0xffffffff ) ^ FSM_R2 ;
   u32 r = ( FSM_R2 + ( FSM_R3 ^ LFSR_S5 ) ) & 0xffffffff ;

   FSM_R3 = S2(FSM_R2);
   FSM_R2 = S1(FSM_R1);
   FSM_R1 = r;

   return F;
}

/* Initialization.
 * Input k[4]: Four 32-bit words making up 128-bit key.
 * Input IV[4]: Four 32-bit words making 128-bit initialization variable.
 * Output: All the LFSRs and FSM are initialized for key generation.
```

```
 * See Section 4.1.
 */

void Initialize(u32 k[4], u32 IV[4])
{
   u8 i=0;
   u32 F = 0x0;

   LFSR_S15 = k[3] ^ IV[0];
   LFSR_S14 = k[2];
   LFSR_S13 = k[1];
   LFSR_S12 = k[0] ^ IV[1];

   LFSR_S11 = k[3] ^ 0xffffffff;
   LFSR_S10 = k[2] ^ 0xffffffff ^ IV[2];
   LFSR_S9  = k[1] ^ 0xffffffff ^ IV[3];
   LFSR_S8  = k[0] ^ 0xffffffff;

   LFSR_S7 = k[3];
   LFSR_S6 = k[2];
   LFSR_S5 = k[1];
   LFSR_S4 = k[0];

   LFSR_S3 = k[3] ^ 0xffffffff;
   LFSR_S2 = k[2] ^ 0xffffffff;
   LFSR_S1 = k[1] ^ 0xffffffff;
   LFSR_S0 = k[0] ^ 0xffffffff;

   FSM_R1 = 0x0;
   FSM_R2 = 0x0;
   FSM_R3 = 0x0;

   for(i=0;i<32;i++)
   {
    F = ClockFSM();
    ClockLFSRInitializationMode(F);
   }

}

/* Generation of Keystream.
 * input n: number of 32-bit words of keystream.
 * input z: space for the generated keystream, assumes
 *          memory is allocated already.
 * output: generated keystream which is filled in z
 * See section 4.2.
 */

void GenerateKeystream(u32 n, u32 *ks)
{
   u32 t = 0;
   u32 F = 0x0;

   ClockFSM(); /* Clock FSM once. Discard the output. */
   ClockLFSRKeyStreamMode(); /* Clock LFSR in keystream mode once. */

   for ( t=0; t<n; t++)
   {
    F = ClockFSM();             /* STEP 1 */
    ks[t] = F ^ LFSR_S0;         /* STEP 2 */
                   /* Note that ks[t] corresponds to z_{t+1} in section 4.2
*/
    ClockLFSRKeyStreamMode();   /* STEP 3 */
   }

}
/*----------------------------------------------------------------------*/
```