ETSI TS 104 053-3 V1.1.1 (2024-07)

TETRA Air Interface Security, Algorithms specification; Part 3: TETRA and Authentication and Key Management Algorithms TAA1

Reference
DTS/TCCE-06213

Keywords
air interface, DMO, security, TETRA, V+D

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee TETRA and Critical Communications Evolution (TCCE).

The present document is part 3 of a multi-part deliverable covering the specifications of the TETRA standard encryption, authentication and key management algorithms, as identified below:

Part 1: "TETRA Encryption Algorithms Set A";

Part 2: "TETRA Encryption Algorithms TEA Set B";

Part 3: "TETRA and Authentication and Key Management Algorithms TAA1";

Part 4: "TETRA Authentication and Key Management Algorithms TAA2".

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document consists of a specification for a set of algorithms TAA1 which may be used in authentication and key management mechanisms for the Terrestrial European Trunked Radio (TETRA). TAA1 is an acronym for "TETRA Authentication and Key Management Algorithms set 1". These specifications are detailed in clause 5.

The present document includes addenda 1, 2 and 3 of the algorithm specifications which adds some algorithms and corrects errors in the original V.1 specification.

The block cipher that is used for this TAAl set is the HURDLE-II algorithm. This is described in the present document at clause 6.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document. "Some referenced ENs are also published as Technical Specifications. In all cases, the latest version of such a document, either EN or TS, should be taken as the referenced document.

[1] ETSI TS 100 392-7: "Terrestrial Trunked Radio (TETRA); Voice plus Data (V+D); Part 7: Security".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

Not applicable.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

TAA1	TETRA Authentication and Authentication Algorithm
TETRA	Terrestrial Trunked Radio

4 Introduction

4.0 General

The set of algorithms TAA1 described in the present document are the associated algorithms used for providing TETRA air interface authentication and key management as specified in detail by the ETSI TS 300-392-7 [1] Security.

The present document is organized as follows:

- Clause 5 provides the specification of all TAA1 algorithms, starting with the definition of shared block structures and expansion and shrinking methods for blocks of input/output bits. A number of the algorithms have inputs and outputs of the same length and meet the same requirements, and are therefore specified group-wise.
- Clause 6 specifies the HURDLE-II algorithm used with TAA1.

4.1 TAA1 introduction

The set of algorithms described is designed to support easy software implementation. All algorithms specified below, with the exception of the rather simple algorithms TB1 up to TB7, make use of a block cipher with an input and output consisting of 64-bits and running under a 128-bit key. The block cipher that is used for this TAA1 set is the HURDLE-II Algorithm described in clause 6 of the present document.

4.2 HURDLE-II introduction

The algorithm HURDLE-II specified in the present document is a 64-bit block cipher with a 128-bit key, designed for use with the TETRA key management and authentication algorithms.

The HURDLE-II design is based on the Feistel structure and consists of an iterated round function and a key schedule algorithm. The key schedule is tailored to 16 iterations of the round function. The Feistel structure, a well tried and tested template for producing secure block ciphers, allows encryption and decryption to be achieved using essentially the same code. The present document is organized as follows:

- clauses 6.2 to 6.5 provide a full specification of the functional components of the HURDLE-II algorithm;
- clause 6.6 specifies how these functions are used to generate the block cipher.

4.3 Comments on TAA1 specification

4.3.1 Illustration of Hurdle-II modes

Figures 1 and 2 illustrate the block structures BL1 and BL2. In these figures the use of the Block Cipher Hurdle-II illustrated by BC is depicted.

What the figures do not explicitly show is that Hurdle-II is used in encryption mode in Figure 1 and in decryption mode in Figure 2 (but note that the text in clauses 5.2 and 5.3 describe the use of the encryption and decryption mode).

4.3.2 Specification of BL2, TA32, TA52, TA 82 and TA92

Clause 5.3 describes the block structure BL2 without formally specifying the length of the output of BL2. Figure 2 shows the output as being 128 bits (16 bytes) which are split in two parts of 120 bits and 8 bits respectively.

Clauses 5.9.3, 5.11.3 and 5.15.3 refer to a 120-bit output of BL2. In fact, what is meant with this is the 120 bits which is the result of deleting from the actual 16-byte BL2 output:

$$O_{15}O_{14}O_{13}O_{12}O_{11}O_{10}O_9O_8O_7O_6O_5O_4O_3O_2O_1O_0$$

the last byte O_0 as indicated in Figure 2.

The convention of a 16-byte output of BL2 is applied.

4.3.3 Requirements on the use of Redundancy in TA31, TA51, TA81 and TA91

The requirements to the TA31, TA51, TA81 and TA91 and corresponding algorithms which are specified in (TA3* and TA5*) and (TA8* and TA9*) should be clarified as follows.

To the input of the TA31, TA51, TA81 and TA91 redundancy shall be added. The output of these algorithms shall be used as input to the respective TA32, TA52, TA82 and TA92 algorithms. The resulting output of these TA32, TA52, TA82 and TA92 should contain the correct redundancy. This is indicated by a Boolean Output.

4.3.4 Meaning of Boolean Output in TA32, TA52, TA82 and TA92

TA31, TA51, TA81 and TA91 are encryption algorithms that encrypt values to which first redundancy is added.

TA32, TA52, TA82 and TA92 are the corresponding decryption algorithms. Each of these algorithms includes a Boolean output that is referred to as the Manipulation Flag. These Boolean outputs are manipulation detection bits and they may be used to check if the original redundancy is also present in the decrypted value: FALSE = redundancy present and correct (not manipulated), TRUE = redundancy incorrect (manipulated).

5 TAA1 Algorithms Set

5.1 Notation

Throughout the present document bit strings are used. The bits of these strings are numbered from right to left, beginning at zero. Bit strings are divided into strings of bytes, again numbered from right to left starting with zero. E.g. the 128-bit input K is written with bytes:

$$K(127), K(126),. . .,K(2), K(1), K(0),$$

with bytes

$$K_{15}\ K_{14}\ K_{13}\ K_{12}\ . . .\ K_4\ K_3\ K_2\ K_1\ K_0,$$

where the j-th byte is denoted as bit string:

$$K_j(7), K_j(6), K_j(5), K_j(4), K_j(3), K_j(2), K_j(1), K_j(0),$$

and

$$K_j(i) = K(8 \times j + i), \text{ for } j = 0, 1, 2, \ldots, 15 \text{ and } i = 0, 1, 2, \ldots, 7.$$

The symbol $\oplus$ denotes the bytewise addition modulo two (exclusive or); i.e. rightmost bit of byte *x* is added modulo two to rightmost bit of byte *y*, ..., leftmost bit of byte *x* to leftmost bit of byte *y*.

x (i)1	*x(i)2*	*x(i)1 ⊕ x(i)2*
0	0	0
0	1	1
1	0	1
1	1	0
x(i) is bit *i* of byte *x* *y(i)* is bit *i* of byte *y* *i = 0,...7*		

5.2 Basic block structure BL1

Most of the TAAl algorithms, with exception of TBl up to TB4 and TA61, consist of the basic block structure BLl shown in Figure 1.

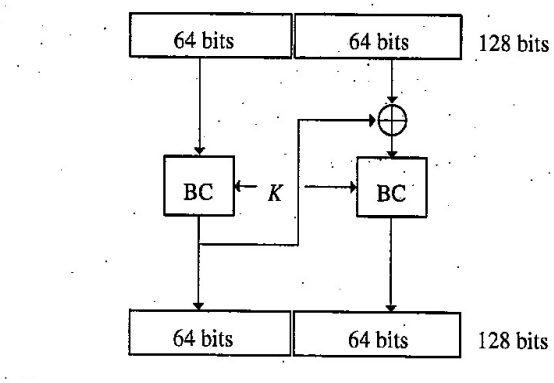


Figure 1: The basic structure BL1

The input consists of two 64-bit blocks. The leftmost block is enciphered with· a 64-bit block cipher BC using a 128-bit key K (BC is the HURDLE-II block cipher). The output of this encryption is XOR-ed with the rightmost block and the result of this is again enciphered using K as the key. The two encrypted output blocks from both BCs are the 128-bit output of the algorithm core. The block ciphers will be used in two modes, in encryption and decryption mode.

5.3 Other block Structure BL2

In three algorithms (TA32, TA52 and TA82, see below), instead of the XOR- function described in clause 5.2 above, the leftmost 56 bits of the leftmost input block are XOR-ed with leftmost 56 bits of the enciphered rightmost block (see Figure 2). The rightmost 8 bits of this enciphered block are XOR-ed with the rightmost 8 bits of the leftmost input block.

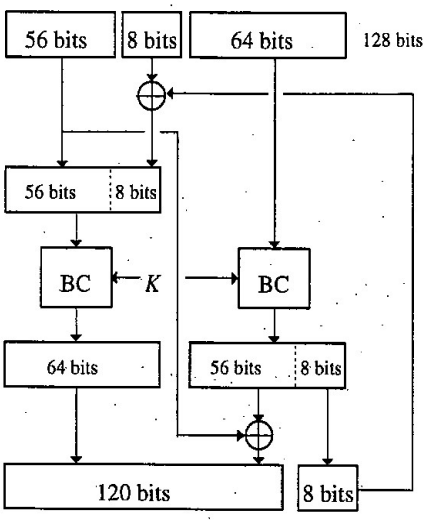


Figure 2: Other structure BL2

5.4 Expansion of bit blocks

5.4.0 General

The number of inputs and their block lengths are specific for each (or groups) of the TAA l algorithms. Several input blocks are expanded before they are offered to the algorithm core in order to obtain the required block lengths.

In several algorithms, input blocks consisting of 80 or 88 bits need to be expanded to 120 or 128 bits. Four expansion methods, EXP1, EXP2, EXP3 and EXP4 which are described below, are used for this.

Both methods, EXP2 and EXP4, expand 80-bit blocks to 128-bit blocks. EXP4 is used for key blocks and is therefore different from EXP2.

5.4.1 Expansion EXP1

The input block consisting of 80 bits is expanded to 120 bits as follows (see also Figure 3 below, where B is the original 80-bit string):

- split the 80-bit input block into 5 pairs of bytes;
- for each pair, compute the XOR of both bytes and insert the result at the right- hand side of the pair.

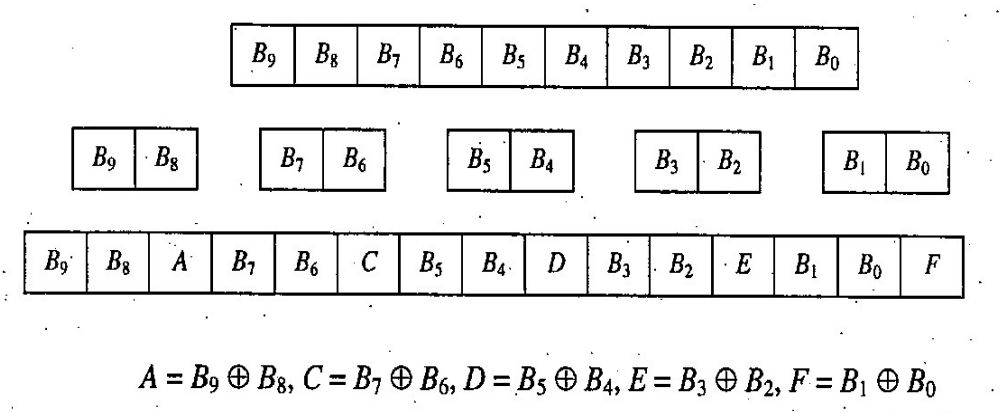


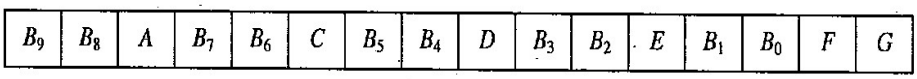
Figure 3: Expansion EXP1 (from 80 to 120 bits)

5.4.2 Expansion EXP2

The input block consisting of 80 bits is expanded to 128 bits. The expansion method EXP2 is an EXP1 expansion with one additional step (see also Figure 4):

- expansion EXP1;
- compute the byte-wise addition (modulo- 256) of all XOR results (5 bytes) obtained in the previous step, and insert the result at the right-hand side of the 120 bits.

Expansion EXP1



$G = A + C + D + E + F \bmod 256$ (additional step).

Figure 4: Expansion EXP2

5.4.3 Expansion EXP3

The string of 88 bits is expanded to 120 bits as follows (see also Figure 5, where the original bit string is denoted by B):

- Split the 88-bit string as shown in Figure 5.
- Compute the values A, C, D and E, and insert these into the 88-bit block as shown in Figure 5.

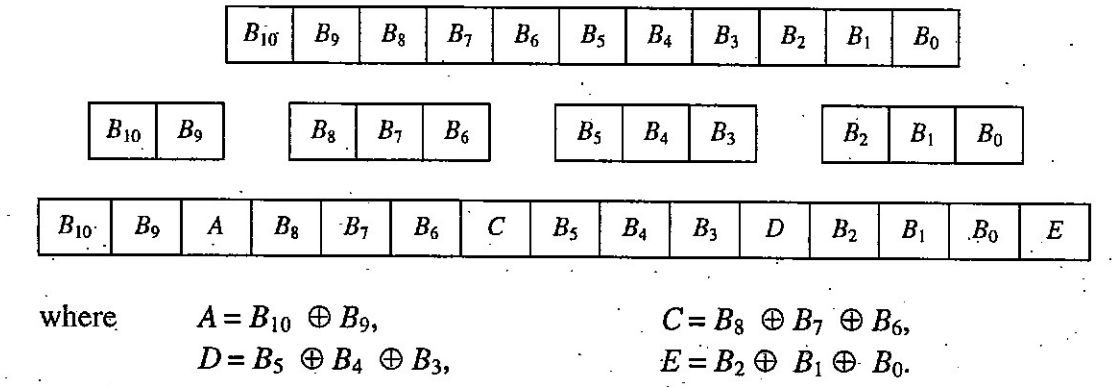
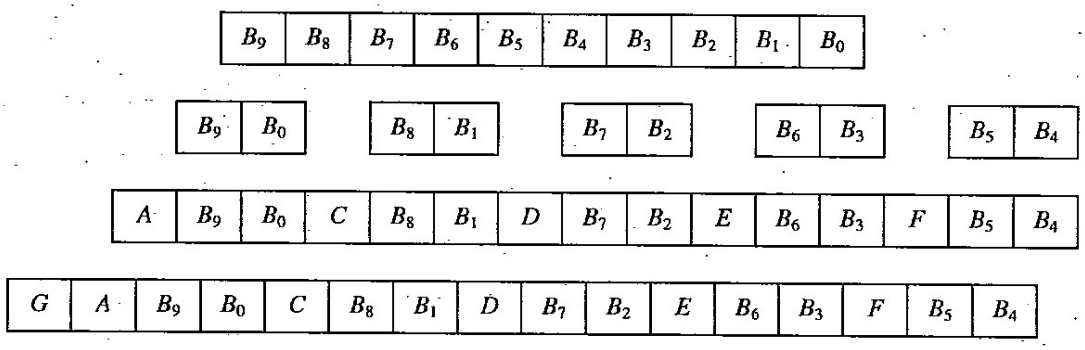


Figure 5: Expansion EXP3 (from 88 to 120 bits)

5.4.4 Expansion EXP4

The 80-bit input blocks, which are used as keys, are expanded to 128 bits, the length of the key to the block cipher. EXP4 is done as follows (see also Figure 6, where the original 80-bit string is denoted by B):

- split the 80-bit input block into 5 pairs of bytes;
- for each pair, compute the byte-wise addition (modulo 256) of both bytes and insert the result at the left-hand side of the pair of bytes;
- compute the XOR of all results of the byte-wise additions (5 bytes) obtained in the previous step, and insert the result at the left-hand side of the 120 bits.



$$A = B_9 + B_0 \bmod 256,\ C = B_8 + B_1 \bmod 256,\ D = B_7 + B_2 \bmod 256,$$
$$E = B_6 + B_3 \bmod 256,\ F = B_5 + B_4 \bmod 256$$
$$\text{and } G = A \oplus C \oplus D \oplus E \oplus F$$

Figure 6: Expansion EXP4 (from 80 to 128 bits)

5.5 Shrinking of blocks

5.5.0 General

The outputs of the algorithm core are shrunk (and split) to the required outputs of specific block lengths. In several algorithms, blocks consisting of 120 or 128 bits need to be shrunk to 80 or 88 bits. Three shrinking methods SHR l, SHR2 and SHR3, which are described below, are used for this.

5.5.1 Shrinking SHR1

The block consisting of 120 bits, denoted as B, is shrunk to 80 bits by taking only the bytes numbered B_{14}, B_{13}, B_{11}, B_{10}, B_8, B_7, B_5, B_4, B_2 and B_1.

5.5.2 Shrinking SHR2

The block consisting of 120 bits, denoted as B, is shrunk to 88 bits by taking only the bytes B_{14}, B_{13}, B_{11}, B_{10}, B_9, B_7, B_6, B_5, B_3, B_2 *and* B_1.

5.5.3 Shrinking SHR3

The block consisting of 128 bits is shrunk to 80 bits by throwing away the leftmost 24 bits as well as the rightmost 24 bits.

5.6 Algorithms TA11, TA21 and TA41

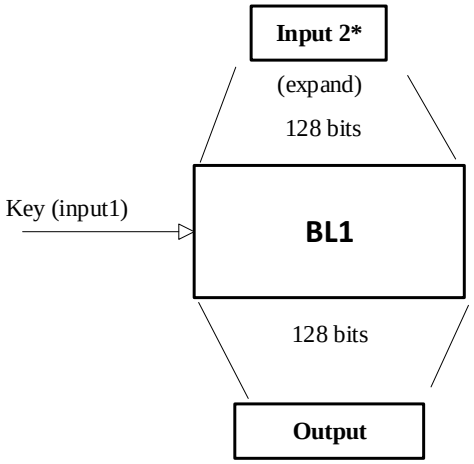
5.6.0 General

The three algorithms TA11, TA21 and TA41 have the same input-output properties and meet the same requirements. The inputs and output are of the following lengths:

Parameter	Size
Input 1 Input 2	128 bits 80 bits
Output	128 bits

The algorithms TA11, TA21 and TA41 are based on the BL1 structure as shown in Figure 7:

- Input 1 is used as the key to BL1.
- Input 2 is expanded to 128 bits and is used as the data input to BL1.
- The Output is the output of BL1.



*Reversed for clause 5.6.2

Figure 7: The TA11, TA21 and TA41 Algorithm

5.6.1 Input 2 of TA11, TA4l

The input 2 is expanded using EXP2 defined in clause 5.4.2.

5.6.2 Input 2 of TA21

The bytes of Input 2 are first reversed, i.e. if Input 2 is $S_9, S_8 \ldots S_1, S_0$ then its reverse **S' is:**

$$S'_I = S_{9-1}, \qquad 0 \leq I \leq 9.$$

The reversed string ***S'*** is defined in clause 5.4.2 then expanded using EXP2.

5.7 Algorithms TA12 and TA22

5.7.0 General

The algorithms TA12 and TA22 have the same input-output properties and meet the same requirements. The Inputs and Outputs are:

Parameter	Size
Input 1 Input 2	128 bits 80 bits
Output 1 Output 2	32 bits 80 bits

The algorithms are based on the BLl structure as shown in Figure 8.

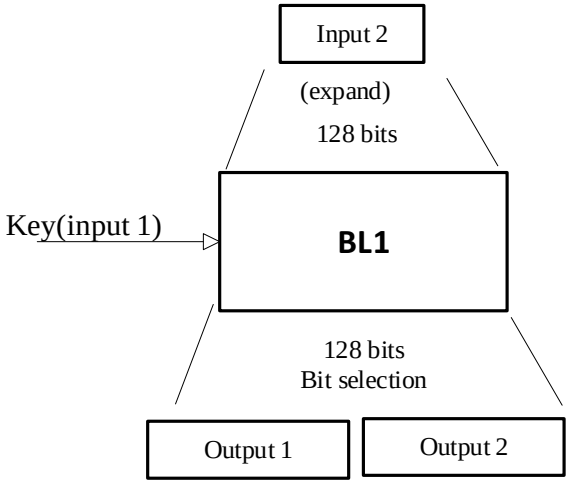


Figure 8: The TA12 and TA22 Algorithm

Input 1 is used as the key to BLl.

Input 2 is expanded to 128 bits and used as the data input to BLl. The two Outputs are derived from the 128-bit output of BLl.

5.7.1 Input expansion

The input 2 is expanded using EXP2 defined in clause 5.4.2.

5.7.2 Output 1 derivation

If the bytes of the 128-bit output of BL1 are numbered B_0 to B_{15} from right to left then the 32-bit Output 1 is:

$B_{15} \oplus B_{12}$	B_9	B_6	$B_3 \oplus B_0$

5.7.3 Output 2 derivation

Output 2 consists of the remaining 10 bytes of the 128-bit output of BL1:

B_{14}	B_{13}	B_{11}	B_{10}	B_8	B_7	B_5	B_4	B_2	B_1

5.8 Algorithm TA31

5.8.0 General

The Inputs and Output of the algorithm are:

Parameter	Size
Input 1 Input 2 Input 3	80 bits 16 bits 80 bits
Output	120 bits

The algorithm is based on the BLl structure as shown in Figure 9.

Input l is expanded and used as the input to BLl.

Input 2 is combined with Input 3 and used to form the key.

The Output is derived from the output of BLl.

5.8.1 Input 1 expansion

The 128-bit data input to BL1is derived from Input1 in the following manner.

Input 1is expanded to 120 bits using EXP1 as defined in clause 5.4.1. A zero byte is then appended to the right-hand end.

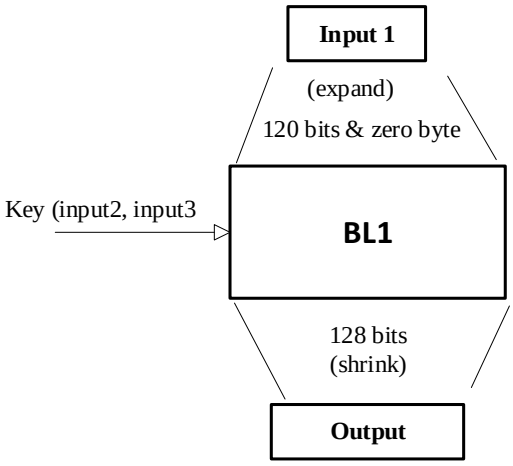


Figure 9: The TA31 Algorithm

5.8.2 Key derivation

The 128-bit key is derived from Input 2 and Input 3 as follows. If Input 2 is denoted as A and Input 3 as B, then:

An 80-bit string C is formed by concatenating five copies of Input A:

$$C = A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0$$

C is XOR-ed with B and the result is expanded to 128 bits using EXP4 defined in clause 5.4.4:

$$K = EXP4 (C \oplus B)$$

5.8.3 Output derivation

The 120-bit Output is:

$$O_{15}O_{14}O_{13}O_{12}O_{11}O_{10}O_9O_7O_6O_5O_4O_3O_2O_1O_0$$

where O is the 128-bit output of BLl. Note that O_8 is discarded.

5.9 Algorithm TA32

5.9.0 General

The Inputs and Outputs of the algorithm are:

Parameter	Size
Input 1 Input 2 Input 3	120 bits 80 bits 16 bits
Output 1 Output 2	80 bits 1 bit

The algorithm is based on the BL2 structure and is shown in Figure 10.

Input 1 is expanded and used as the input to BL2.

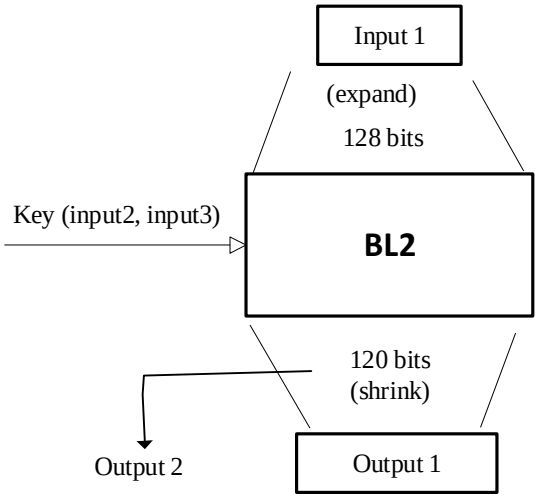


Figure 10: The TA32 Algorithm

Input 2 is combined with Input 3 and used to form the key.

Output 1 and Output 2 are derived from the 120-bit output of BL2.

5.9.1 Input 1 expansion

The 128-bit data input to BL2 is derived from Input 1 in the following manner.

If Input 1 is denoted by B, then the expansion is:

$$B_{14}B_{13}B_{12}B_{11}B_{10}B_9B_8 \text{ (zero byte) } B_7B_6B_5B_4B_3B_2B_1B_0$$

5.9.2 Key derivation

The 128-bit key is derived in the same way as described in clause 5.8.2, however, for TA32 the mentioned Input 2 and Input 3 has to be interchanged.

5.9.3 Output derivation

The 120-bit output of BL2 is shrunk to the 80-bit Output 1 using SHR1 defined in clause 5.5.1.

The one-bit Output 2 is the binary result of a check of the 120-bit output of BL2, denoted *B14 ….. Bo,* before shrinking:

Output 2 = False if $Bi = Bi_{+1} \oplus B_{i+2}$ for i = 0, 3, 6, 9, 12

Output 2 = True otherwise.

5.10 Algorithm TA5l

5.10.0 General

The Inputs and Output of the algorithm are:

Parameter	Size
Input 1	80 bits
Input 2	16 bits
Input 3	128 bits
Input4	5 bits
Output	120 bits

The algorithm is based on the BL 1 structure as shown in Figure 11.

The combination of Input 1 and Input 4 is expanded and used as the input to BLl.

Input 2 is combined with Input 3 and used to form the key.

The Output is derived from the output of BL1.

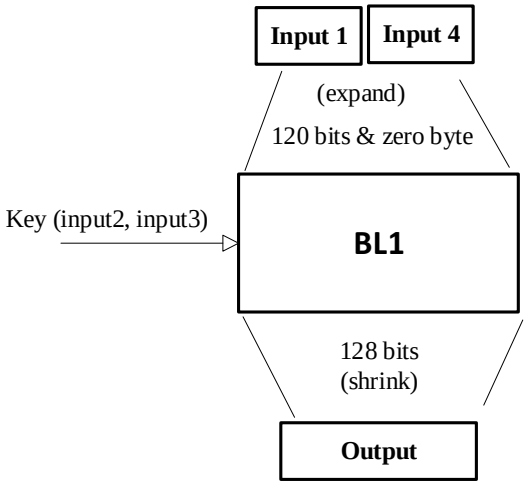


Figure 11: The TA51 Algorithm

5.10.1 Data input to BL1

The 128-bit data input to BLl is derived from Input 1 and Input 4 in the following manner.

Input 1 and Input 4 are concatenated as follows:

- (Input 4 is proceeded by three zeroes).

- This string of 88 bits is expanded to 120 bits using EXP3 defined in clause 5.4.3.

Input 1	000	Input 4

5.10.2 Key derivation

The 128-bit key is derived from Input 2 and Input 3 as follows. If Input 2 is denoted as A and Input 3 as B, then:

An128-bit string C is formed by concatenating eight copies of Input A.

$$C = A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0 A_1 A_0$$

- C is XOR-ed with B and the result.
- $K = C \oplus B$ is used as the key.

5.10.3 Output derivation

The 120-bit Output is derived as described in clause 5.8.3.

5.11 Algorithm TA52

5.11.0 General

The Inputs and Outputs of the algorithm are:

Parameter	Size
Input 1	120 bits
Input 2	128 bits
Input 3	16 bits
Output 1	80 bits
Output 2	1 bit
Output 3	5 bits

The algorithm is based on the BL2 structure and is shown in Figure 12.

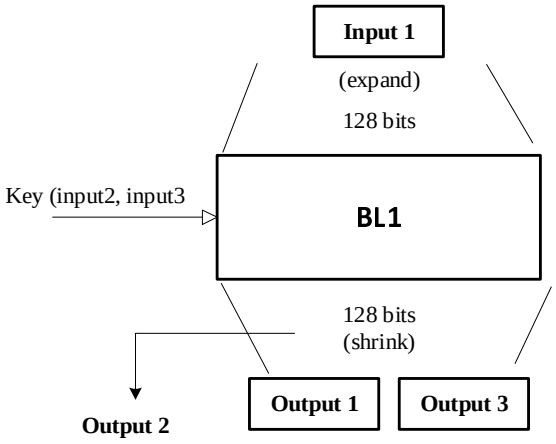


Figure 12: The TA52 Algorithm

Input 1 is expanded and used as the input to BL2.

Input 2 is combined with Input 3 and used to form the key.

Output 1, Output 2 and Output 3 are derived from the 120-bit output of BL2.

5.11.1 Input 1 expansion

Input 1 is expanded to 128 bits as described in clause 5.9.1.

5.11.2 Key derivation

The 128-bit key is derived in the same way as described in clause 5.10.2, however for TA52 the mentioned Input 2 and Input 3 has to be interchanged.

5.11.3 Output derivation

The 120-bit output of BL2 is shrunk to the 88-bits using SHR2 defined in clause 5.5.1. The leftmost 80 bits are the Output 1; the rightmost 5 bits the Output 3.

The one-bit Output 2 is the binary result of a check of the 120-bit output of BL2, denoted B 14... Bo, before shrinking:

Output 2 = False if $B_i = B_{i+1} \oplus B_{i+2} \oplus B_{i+3}$ for $i =$ 0, 4, 8 and

$B_{12} = B_{13} \oplus B_{14}$, and

if the leftmost three bits of byte B 1 are zero.

Output 2=True otherwise.

5.12 The Algorithm TA61

5.12.0 General

The Inputs and Output of the algorithm are:

Parameter	Size
Input 1 Input 2	80 bits 24 bits
Output	24 bits

The structure of TA61 is shown in Figure 13.

Input 1 is shrunk to 64 bits and used as the input to BC. Input 1 is also expanded and used as the key to BC.

Input 2 is XOR-ed in three steps with three 24-bit strings derived from the BC output. The XOR results of step 1 and 2 are permuted first before they are offered to the next step. The result of the last step is the Output of the algorithm.

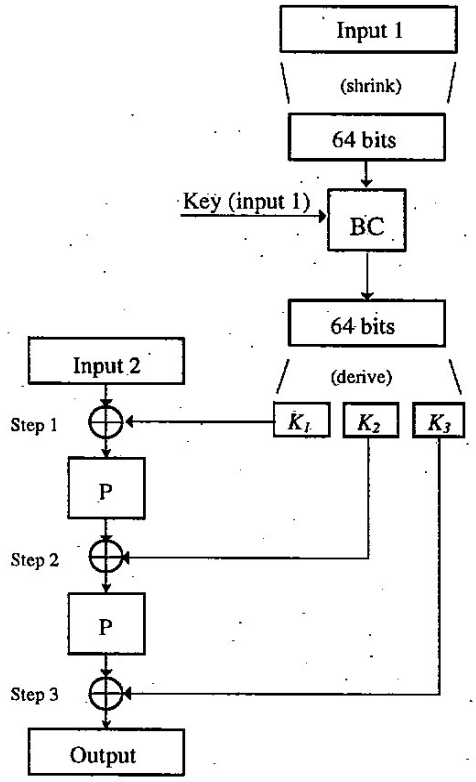


Figure 13: The TA61 Algorithm

5.12.1 Input 1 shrinking

The 64-bit data input to BC is derived from Input 1 in the following manner. If Input 1 is denoted as A, then the 64-bit data input to BC is:

$$(A_9 \oplus A_7)(A_8 \oplus A_6)(A_7 \oplus A_5), \ldots, (A_3 \oplus A_1)(A_2 \oplus A_0).$$

5.12.2 Key derivation

The Input 1 is expanded to a 128-bit key by using EXP4 defined in clause 5.4.4.

5.12.3 K-string derivation

The three 24-bit strings Ki, K2 and K3 are derived from the BC output in the following manner:

If we denote the BC output by:

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

then

$K_1 =$	B_7	B_4	B_1
$K_2 =$	B_6	B_3	B_0
$K_3 =$	B_5	B_2	B_7

5.12.4 Permutation P

The permutation P on the 24-bit strings (3 bytes) is done as follows. If the input to the permutation is:

a_2	a_1	a_0

where a_2, a_1 and a_0 are bytes, then the output is

$S(2a_2+2a_1-a_0)$	$S(2a_2+2a_0-a_1)$	$S(2a_1+2a_0-a_2)$

where + and - mean normal byte-by-byte addition and subtraction (i.e. modulo 256), and S is the permutation on bytes, shown in the table below, that is also used in HURDLE-II (see [3]).

(Table entries are given row-wise in hexadecimal in the order S[00], S[01], . . S[ff].)

```
f4 65 01 00 ba 7a a7 47 98 dd 9d ad 96 5d aa 3d
58 c0 72 d8 66 4c 3e e0 80 55 de 90 2a 4b 83 a0
51 39 ed 6c 8a 2c 56 60 4a 1f d0 70 6e 33 8b 26
2e 6f 89 48 5e 40 c3 a4 a9 cf 22 50 e1 15 0c ab
d5 f8 5f 36 04 a6 4e 92 1e 2b 88 30 93 45 67 16
8c 68 23 38 61 25 1a 81 63 cb c1 13 41 37 0e 97
5b ca 57 24 4d 17 c4 b9 b3 ef 8d 52 32 2f ec 20
d9 11 d1 28 79 da fb e9 bb 06 77 db fc fe cd 84
1d a1 54 1b b0 e4 cc 7c 2d 27 31 49 f5 02 69 53
4f 44 df 18 5c 0f bc 9b 94 bd dc 0b a2 c7 09 ac
c6 9f 82 1c 05 46 c2 34 3c 0d 3b ce b7 be 08 9c
6b ee e5 87 af bf f2 eb 7b 07 64 c5 b6 ae 9a 95
35 a5 59 12 9e a3 b8 8e 5a f7 62 d2 3a a8 7d 85
f6 c8 71 29 d6 d7 43 f9 78 76 73 10 91 19 0a 99
f0 e6 3f 14 f1 e2 b1 86 b4 f3 74 fa 6a b2 21 6d
ea b5 e7 e3 c9 d3 8f 03 75 e8 d4 42 fd 7e ff 7f
```

5.13 The Algorithm TA71

5.13.0 General

The Inputs and Output of the algorithm are:

Parameter	Size
Input 1 Input 2	80 bits 80 bits
Output	80 bits

The algorithm is based on the BL1 structure as shown in Figure 14.

Input 1 and Input 2 are combined and used as the data input to BL1.

Input 1 is combined with Input 2 and used to form the key.

The Output is derived from the output of BL1.

5.13.1 Data input

The 128-bit data input to BL1 is derived from Input 1 and Input 2 in the following manner: Input 1 is XOR-ed with Input 2 and the resultant 80-bit value is expanded to 128 bits using EXP2 as defined in clause 5.4.2.

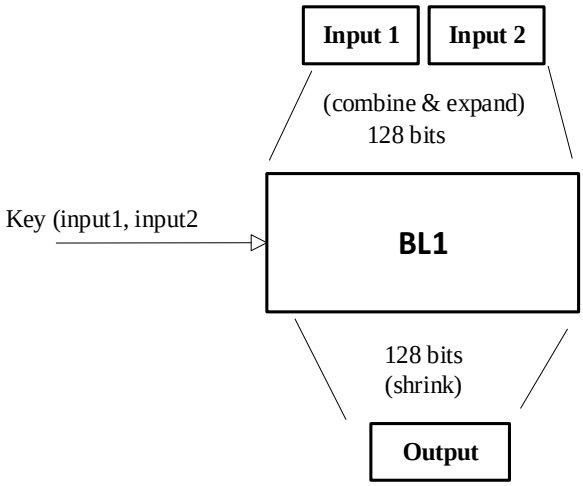


Figure 14: The TA71 Algorithm

5.13.2 Key input

The 128-bit key is derived from Input 1 and Input 2 in the following manner:

If Input 1 is denoted as A and Input 2 as B, then the key K is given by:

$$K = A_9 A_8 A_7 A_6 A_5 A_4 (A_3 \oplus B_9)(A_2 \oplus B_8)(A_1 \oplus B_7)(A_0 \oplus B_6) B_5 B_4 B_3 B_2 B_1 B_0$$

5.13.3 Output derivation

The 128-bit output of BLl is shrunk to the 80-bit Output using SHR3 defined in clause 5.5.3.

5.14 Algorithm TA81

5.14.0 General

The basic block structure BL1 specified in clause 5.2 is used to construct TA81.

The Inputs and Output of the TA81 algorithm are:

Parameter	Size
Input 1 Input 2 Input 3 Input 4	80 bits 16 bits 128 bits 16 bits
Output	120 bits

The algorithm is based on the BL1 structure as shown in Figure 15 below.

The combination of Input 1 and Input 4 is expanded and used as the input to BL1.

Input 2 is combined with Input 3 and used to form the key.

The Output is derived from the output of BL1.

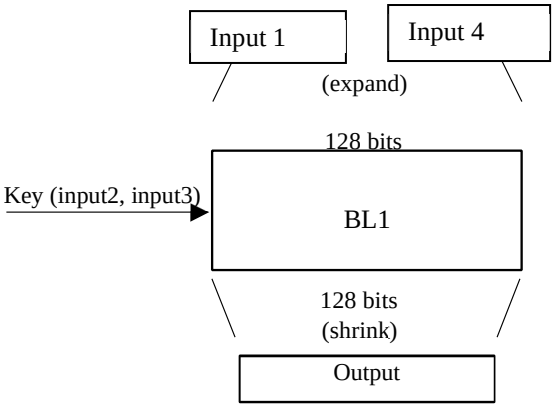
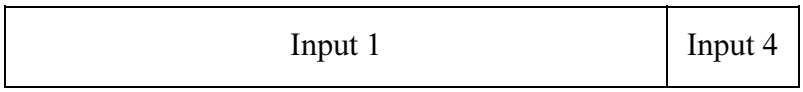


Figure 15: The TA81 Algorithm

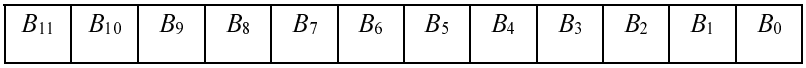
5.14.1 Data input to BL1

The 128-bit data input to BL1 is derived from Input 1 and Input 4 in the following manner:

1) Input 1 and Input 4 are concatenated as follows:

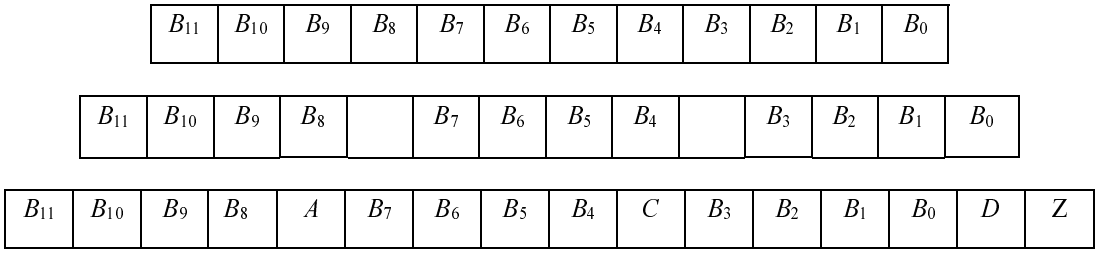


This 96-bit string denoted as a 12-byte string.



2) This 12-byte string is expanded to 16-byte string as described below:

- Compute the values A, C, and D, insert these into the 12-byte block and add an all zero byte Z = (00000000) as shown in Figure 16 below.



where:
$A = B_{11} \oplus B_{10} \oplus B_9 \oplus B_8$,
$C = B_7 \oplus B_6 \oplus B_5 \oplus B_4$,
$D = B_3 \oplus B_2 \oplus B_1 \oplus B_0$, $Z = (00000000)$.

Figure 16: Expansion from 12 to 16 bytes

5.14.2 Key derivation

Derive the 128-bit key from Input 2 and Input 3 as follows. If Input 2 is denoted as A (2 bytes) and Input 3 as B (16 bytes), then:

a 128-bit string *C* is formed by concatenating eight copies of Input A.

$$C = A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0$$

C is XOR-ed with *B* and the result.

$K=C \oplus B$ is used as the key.

5.14.3 Output derivation

Derive the 120-bit Output as described in clause 5.8.3 that is.

The 120-bit Output is*:*

$$0_{15}0_{14}0_{13}0_{12}0_{11}0_{10}0_9 0_7 0_6 0_5 0_4 0_3 0_2 0_1 0_0$$

where *0* is the 128-bit output of BL1 and 0_j is the j-th byte of *0*.

Note that byte O_8 is discarded.

5.15 Algorithm TA82

5.15.0 General

The basic block structure BL2 specified in clause 5.3 is used to construct TA82.

The Inputs and Outputs of the TA82 algorithm are:

Parameter	Size
Input 1 Input 2 Input 3	120 bits 128 bits 16 bits
Output 1 Output 2 Output 3	80 bits 1 bit 16 bits

The algorithm is based on the BL2 structure and is shown in Figure 17.

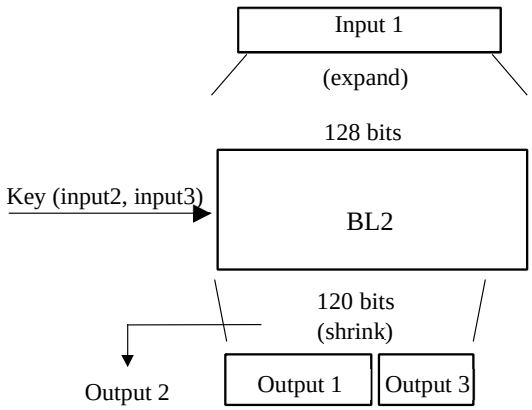


Figure 17: The TA82 Algorithm

Input 1 is expanded and used as the input to BL2.

Input 2 is combined with Input 3 and used to form the key.

Output 1, Output 2 and Output 3 are derived from the 128-bit output of BL2.

5.15.1 Input 1 expansion

Input 1 (length 120 bits) is expanded to 128 bits as described in clause 5.9.1. That is if Input 1 is denoted by *B* (15 bytes), then the expansion is:

$$B_{14}B_{13}B_{12}B_{11}B_{10}B_9B_8 \text{ (zero byte) } B_7B_6B_5B_4B_3B_2B_1B_0$$

5.15.2 Key derivation

The 128-bit key is derived in the same way as described in clause 5.14.1. However, for TA82 the mentioned Input 2 and Input 3 has to be interchanged, that is derive the 128-bit key from Input 2 and Input 3 as follows:

If Input 3 is denoted as *A* (2 bytes) and Input 2 as *B* (16 bytes), then:

a 128-bit string *C* is formed by concatenating eight copies of Input *A*.

$$C = A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0$$

*C i*s XOR-ed with *B* and the result

$K=C \oplus B$ is used as the key.

5.15.3 Output derivation

Denote the 16-byte output (see also clause 5.3 describing BL2) as:

$$B_{15}\ B_{14}\ B_{13}\ B_{12}\ B_{11}\ B_{10}\ B_9\ B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$$

The 10 bytes (80 bits) $B_{15}\ B_{14}\ B_{13}\ B_{12}\ B_{10}\ B_9\ B_8\ B_7\ B_5\ B_4$ are set as the Output 1.

The 2 bytes (16 bits) $B_3\ B_2$ are set as the Output 3.

The one-bit Output 2 is the binary result of a check of the output of BL2:

- Output 2 = FALSE if $Bi = B_{4+i} \oplus B_{3+i} \oplus B_{2+i} \oplus B_{1+i}$. for i = 1, 6,11
- Output 2 = TRUE otherwise.

5.16 Algorithm TA91

5.16.0 General

The basic block structure BL1 specified in clause 5.2 is used to construct TA91.

The Inputs and Output of the TA91 algorithm are:

Parameter	Size
Input 1 Input 2 Input 3	96 bits 16 bits 128 bits
Output	120 bits

The algorithm is based on the BL1 structure as shown in Figure 18 below.

Input 1 is expanded and used as the input to BL1.

Input 2 is combined with Input 3 and used to form the key.

The Output is derived from the output of BL1.

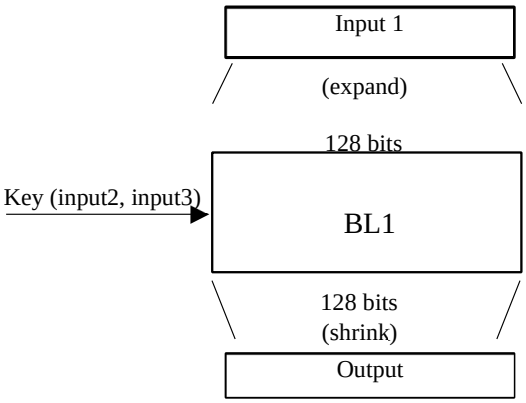


Figure 18: The TA91 Algorithm

5.16.1 Data input to BL1

The 128-bit data input to BL1 is derived from Input 1 in the following manner:

1) Input 1 is denoted as a 12-byte string.

B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

2) This 12-byte string is expanded to 16-byte string as described below.

- Compute the values A, C, and D, insert these into the 12-byte block and add an all zero byte Z = (00000000) as shown in Figure 19 below.

B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

B_{11}	B_{10}	B_9	B_8		B_7	B_6	B_5	B_4		B_3	B_2	B_1	B_0

B_{11}	B_{10}	B_9	B_8	A	B_7	B_6	B_5	B_4	C	B_3	B_2	B_1	B_0	D	Z

where:
$A = B_{11} \oplus B_{10} \oplus B_9 \oplus B_8$,
$C = B_7 \oplus B_6 \oplus B_5 \oplus B_4$,
$D = B_3 \oplus B_2 \oplus B_1 \oplus B_0$,
Z = (00000000).

Figure 19: Expansion from 12 to 16 bytes

5.16.2 Key derivation

Derive the 128-bit key from Input 2 and Input 3 as follows. If Input 2 is denoted as A (2 bytes) and Input 3 as B (16 bytes), then:

a 128-bit string C is formed by concatenating eight copies of Input A.

$$C = A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0$$

C is XOR-ed with B and the result.

$K=C \oplus B$ is used as the key.

5.16.3 Output derivation

The 120-bit Output is derived as described in clause 5.8.3 that is:

The 120-bit Output is:

$$O_{15}O_{14}O_{13}O_{12}O_{11}O_{10}O_9O_7O_6O_5O_4O_3O_2O_1O_0$$

where:

O is the 128-bit output of BL1; and

O_j is the j-th byte of O.

Note that byte O_8 is discarded.

5.17 Algorithm TA92

5.17.0 General

The basic block structure BL2 specified in clause 5.3 is used to construct TA92.

The Inputs and Outputs of the TA92 algorithm are:

Parameter	Size
Input 1 Input 2 Input 3	120 bits 128 bits 16 bits
Output 1 Output 2	96 bits 1 bit

The algorithm is based on the BL2 structure and is shown in Figure 20.

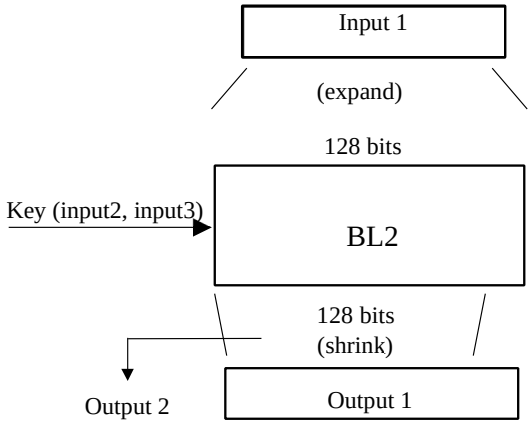


Figure 20: The TA92 Algorithm

Input 1 is expanded and used as the input to BL2.

Input 2 is combined with Input 3 and used to form the key.

Output 1 and Output 2 are derived from the 128-bit output of BL2.

5.17.1 Input 1 expansion

Input 1 (length 120 bits) is expanded to 128 bits as described in clause 5.9.1. That is if Input 1 is denoted by *B* (15 bytes), then the expansion is:

$B_{14}B_{13}B_{12}B_{11}B_{10}B_9B_8$ (zero byte) $B_7B_6B_5B_4B_3B_2B_1B_0$

5.17.2 Key derivation

The 128-bit key is derived in the same way as described in clause 5.16.2 of this addendum. However, for TA92 the mentioned Input 2 and Input 3 has to be interchanged, that is derive the 128-bit key from Input 2 and Input 3 as follows:

Input 3 is denoted as A (2 bytes) and Input 2 as B (16 bytes), then:

a 128-bit string *C* is formed by concatenating eight copies of Input *A*.

$$C = A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0A_1A_0$$

C is XOR-ed with *B* and the result $K=C \oplus B$ is used as the key.

5.17.3 Output derivation

Denote the 16-byte output see also clause 4.3.2 of BL2 as:

$$B_{15}\ B_{14}\ B_{13}\ B_{12}\ B_{11}\ B_{10}\ B_9\ B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$$

The 12 bytes (96 bits) $B_{15}\ B_{14}\ B_{13}\ B_{12}\ B_{10}\ B_9\ B_8\ B_7\ B_5\ B_4\ B_3\ B_2$ are set as the Output 1.

The one-bit Output 2 is the binary result of a check of the output of BL2:

- Output 2 = FALSE if $B_i = B_{4+i} \oplus B_{3+i} \oplus B_{2+i} \oplus B_{1+i}$. for *i* is 1 and 6 and 11.
- Output 2 = TRUE otherwise.

5.18 Algorithm TA101

The Inputs and Output of the TA101 algorithm are:

Parameter	Size
Input 1 (KS) Input 2 (GCK0) Input 3 (MNI)	128 bits 80 bits 24 bits
Output (KSv)	128 bits

TA101 uses the basic block structure BL1 of the TAA1 algorithm set.

An intermediate 80-bit value Input 4 (INT) is generated by concatenating 3 copies of Input 3 and a single string of 8 binary zeros, and XORing this string with Input 2. The leftmost bit of the first copy of Input 3 shall form the leftmost bit of the concatenated string, and the 8 zeros shall form the rightmost bits of the concatenated string. This is illustrated in Figure 21.

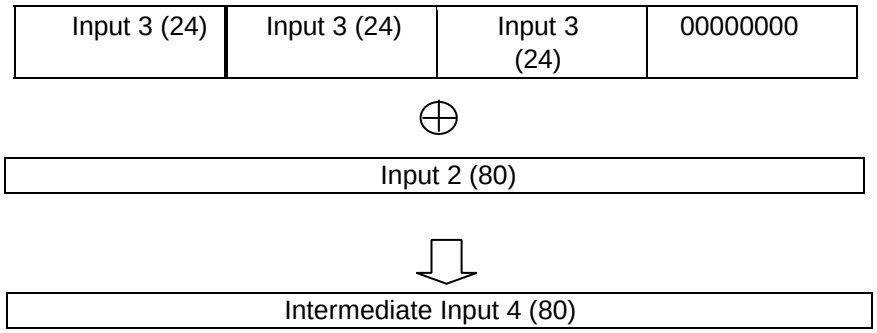


Figure 21

That is, where:

- Input 1, KS is a binary vector of 128 bits, labelled KS[127]….KS[0]
- Input 2, GCK0 is a binary vector of 80 bits, labelled GCK0[79]….GCK0[0]
- Input 3, MNI is a binary vector of 24 bits, labelled MNI[23]….MNI[0]
- Input 4, INT is a binary vector of 80 bits, labelled INT[79]….INT[0]

and where [0] in each case represents the least significant bit:

$$INT[I] = (GCK0[I] + MNI[I - 56]) \bmod 2, \quad I = 79, 78, \ldots, 56$$
$$INT[I] = (GCK0[I] + MNI[I - 32]) \bmod 2, \quad I = 55, 54, \ldots, 32$$
$$INT[I] = (GCK0[I] + MNI[I - 8]) \bmod 2, \quad I = 31, 30, \ldots, 8$$
$$INT[I] = (GCK0[I]), \quad I = 7, 6, \ldots, 0.$$

This intermediate Input 4, INT[79] … INT[0], is expanded using the EXP2 expansion defined in the TAA1 algorithm set, and the 128-bit result used as the data input to BL1. Input 1, KS[127]….KS[0], forms the key input to BL1. The output from BL1 becomes the TA101 output value KSv[127] … KSv[0].

5.19 Algorithm TB1

The Input and Output of the algorithm are:

Parameter	Size
Input	16-32 bits
Output	128 bits

The Output of this algorithm is the Input repeated until 128 bits are obtained starting with the leftmost bit of the Input as the leftmost bit of the Output.

5.20 Algorithm TB2

The Input and Output of the algorithm are:

Parameter	Size
Input	128 bits
Output	128 bits

The Output of this algorithm is the Input.

5.21 Algorithm TB3

The Inputs and Output of the algorithm are:

Parameter	Size
Input 1 Input 2	128 bits 16-32 bits
Output	128bits

The Output of this algorithm is the Input 1 XOR-ed with the repeated Input 2.

Input 2 is repeated until 128 bits are obtained starting with the leftmost bit of the Input 2 as the leftmost bit of the Output.

5.22 Algorithm TB4

The Input and Output. of the algorithm are:

Parameter	Size
Input 1 Input 2	80 bits 80 bits
Output	80 bits

The Output of this algorithm is the Input 1 XOR-ed with Input 2.

5.23 Algorithm TB5

The Inputs and Output of the TB5 algorithm are:

Parameter	Size
Input 1 Input 2 Input 3 Input 4	80 bits 14 bits 12 bits 6 bits
Output	80 bits

The Output of this algorithm is input 1 XOR-ed with an intermediate value. The intermediate value consists of inputs 2, 3 and 4 concatenated with inputs 3 and 4 repeated; with the leftmost bits of input 2 forming the leftmost bits of the intermediate value:

Input 2	Input 3	Input 4	Input 3	Input 4	Input 3	Input 4	Input 3

That is:

$$
\begin{aligned}
&ECK[I] = (CK[I] + CN[I]) \bmod 2, && I = 0, 1, \ldots, 11\\
&ECK[I] = (CK[I] + CC[I-12]) \bmod 2, && I = 12, 13, \ldots, 17\\
&ECK[I] = (CK[I] + CN[I-18]) \bmod 2, && I = 18, 19, \ldots, 29\\
&ECK[I] = (CK[I] + CC[I-30]) \bmod 2, && I = 30, 31, \ldots, 35\\
&ECK[I] = (CK[I] + CN[I-36]) \bmod 2, && I = 36, 37, \ldots, 47\\
&ECK[I] = (CK[I] + CC[I-48]) \bmod 2, && I = 48, 49, \ldots, 53\\
&ECK[I] = (CK[I] + CN[I-54]) \bmod 2, && I = 54, 55, \ldots, 65
\end{aligned}
$$

ECK [I] = (CK[I] + LA [I-66]) mod 2, I = 66, 67, …, 79

5.24 Algorithm TB6

The Inputs and Output of the TB6 algorithm are:

Parameter	Size
Input 1 (CK) Input 2 (CN) Input 3 (SSI)	80 bits 12 bits 24 bits
Output (ECK)	80 bits

The Output of this algorithm is input 1 XOR-ed with an intermediate output. The intermediate output consists of the concatenation in turn of inputs 2,3, 2, 3 and 3*; with the most significant bits of input 2 forming the most significant bits of the intermediate output, and where input 3* is the least significant 8 bits of input 3:

Input 2	Input 3	Input 2	Input 3	Input 3*

That is, where:

CK is a binary vector of 80 bits, labelled CK[79]….CK[0]
CN is a binary vector of 12 bits, labelled CN[11]….CN[0]
SSI is a binary vector of 24 bits, labelled SSI[23]….SSI[0]
ECK is a binary vector of 80 bits, labelled ECK[79]….ECK[0]

and where [0] in each case represents the least significant bit:

ECK [I] = (CK[I] + CN [I-68]) mod 2 , I = 79, 78, …,68
ECK [I] = (CK[I] + SSI [I - 44]) mod 2 , I = 67, 66, …, 44
ECK [I] = (CK[I] + CN [I -32]) mod 2 , I = 43, 42, …, 32
ECK [I] = (CK[I] + SSI [I - 8]) mod 2 , I = 31, 30, …, 8
ECK [I] = (CK[I] + SSI [I]) mod 2 , I = 7, 6, …, 0

5.23 Algorithm TB7

The Input and Output of the TB7 algorithm are:

Parameter	Size
Input	96 bits
Output	128 bits

The 16-byte Output of this algorithm is obtained by adding 4 bytes to the 12-byte Input as described below:

1) Denote the 12-byte input at shown below:

B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

2) Compute the 4 bytes *A*, *C*, *D* and *E*, insert these into the 12 byte in Figure 22 below:

B_{11}	B_{10}	B_9	A	B_8	B_7	B_6	C	B_5	B_4	B_3	D	B_2	B_1	B_0	E

where:
$A = B_{11} \oplus B_{10} \oplus B_9$,
$C = B_8 \oplus B_7 \oplus B_6$,
$D = B_5 \oplus B_4 \oplus B_3$,
$E = B_2 \oplus B_1 \oplus B_0$.

Figure 22: Expansion from 12 to 16 bytes

6 The HURDLE-II Algorithm

6.0 General

The functional structure of the HURDLE-II algorithm with the 16 round functions and the 16 round keys is shown in Figure 23 of clause 6.7.

6.1 Notation

Throughout the present documentbit strings are used. These are written with most significant bit on the left and least significant bit on the right. The bits of such a string are numbered from right to left, beginning at zero. Bit strings are divided into strings of bytes, again with most significant byte leftmost and with byte 0 rightmost.

The inputs to the HURDLE-II algorithm are a 64-bit string P with bytes:

$$P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0$$

and a 128-bit key *K* with bytes $K_{15} K_{14} \ldots K_1 K_0$.

The 16 rounds of encryption are numbered by 1, 2, . . . 16. K^i indicates the 96-bit round key used in round i (these round keys are derived from *K* using the HURDLE-II key schedule algorithm, as defined in detail below).

The symbol ⊕ denotes the bytewise addition modulo two (exclusive or); i.e. msb of byte x is added to msb of byte y, ..., lsb of byte x to lsb of byte y.

$x(i)$	$y(i)$	$x(i) \oplus y(i)$
0	0	0
0	1	1
1	0	1
1	1	0

$x(i)$ is bit i of byte x
$y(i)$ is bit i of byte y

$i = 0,..., 7$

6.2 Encryption

The encryption of input P under key K is carried out as follows. Input P is divided into two 4-byte strings, L^0 and R^0, with:·

$$L^0 = P_7 P_6 P_5 P_4 , \qquad R^0 = P_3 P_2 P_1 P_0$$

Then for each integer i with $1 \leq i \leq 16$, is defined as:

$$L^i = R^{i-1}, \qquad R^i = L^{i-1} \oplus f(R^{i-1}, K^i) \tag{1}$$

Equations (1) constitute the i-th round function of HURDLE-II, where f denotes the round function with R^{i-1} and round key K^i as inputs (see clause 6.4).

The encryption of P under K is the 8-byte string $C = R^{16}L^{16}$ (note swapping of halves) offered at the end of the sixteenth round, i.e. after the execution of the last round function. See Figure 23 of clause 6.7.

6.3 Decryption

Decryption of the ciphertext C is achieved by encrypting it using as round keys K^{16}...., K^1.

That is, following exactly the same procedure as above, but using round in round instead of round key K^{17-i}. The decryption round keys are easily derived from the key K as described in clause 6.5.

6.4 The f Function of HURDLE-II

6.4.0 General

The round function f of HURDLE-II, has as inputs a 4-byte value $X = X_3\ X_2\ X_1\ X_0$ and a 12-byte round key, denoted by $K^i = K^i_{11}\ K^i_{10}$ …… K^i_0 (see clause 6.5).

The computation of the f function proceeds in the following stages (see Figure 24 of clause 6.7):

- expansion of the 4-byte input
- addition of round key and chained byte substitution
- nibble selection
- bit permutation

6.4.1 Data Expansion

The 4-byte input data $X = X_3\ X_2\ X_1\ X_0$ is expanded to the following 12-byte string E:

$$E_{11}E_{10}E_9E_8E_7E_6E_5E_4E_3E_2E_1E_0 = X_1X_3X_0X_2X_3X_1X_2X_0X_3X_2X_1X_0$$

6.4.2 Addition of Round Key and Chained Byte Substitution

E and the round key K^i the following 12-byte value $T = T_{11}T_{10}\ldots\ldots\ldots T_0$ is computed where:

$$T_0 = S[(E_0 + K_0^i) \bmod 256]$$

and

$$T_j = S[((E_j + K_j^i) \bmod 256) \oplus T_{j-1}], \qquad 1 \leq j \leq 11, \tag{2}$$

with S as the byte permutation defined in Table 1 of clause 6.7. The table entries are given row-wise in hexadecimal in the order S [00], S[01], ……S [ff].

Thus, the expanded version of the data is added modulo 256 with the round key bytes and the resulting values are used to obtain bytes from the look-up table S box in a chained manner. This part of the f function is illustrated in Figure 25 of clause 6.7.

6.4.3 Nibble selection

The 12-byte output T is reduced to the following 32-bit string:

$$U = U_{11}U_{10}U_9U_8U_7U_6U_5U_4$$

where, for each $0 \leq q < 12$, Uq denotes the 4 least significant bits of Tq, i.e. the rightmost nibble of Tq.

6.4.4 Bit Permutation

The final 4-byte output of the function f is obtained by performing a bit permutation π on the 32-bit string U to produce a 4-byte value $Y = Y_3\ Y_2\ Y_1\ Y_o$.

The permutation π is given in Table 2 of 6.7 and its action on the bits of U is as follows: if the bits of U and Y are numbered from left to right by 31to 0, then bit j of Y is equal to bit i of U if π [j] = i. In Table 2, the entries are arranged row-wise in the order π [0], π [l],.., π [31].

The permutation π has a compact description enabling a fast software implementation: for each $0 \leq m < 8$ and each $0 \leq n < 4$, bit m of Y_n is equal to bit n of U_{m+4}, i.e. bit n of T_{m+4}.

The final 4-byte output of the function f is equal to $Y = Y_3Y_2\ Y_1\ Y_0$.

6.5 Key Schedule Algorithm

In this clause the key schedule algorithm for HURDLE-II is defined. Recall that the 16-byte key is denoted by:

$$K = K_{15}K_{14}\ldots K_0$$

and the12 byte round keys by K^1,……, K^{16}

A 16 -byte constant is *specified* as $D = D_{15}\ D_{14}\ \ldots\ D_0$ by:

$$D = 3c\ a7\ ec\ 25\ 79\ 57\ df\ c0\ 38\ 0a\ 33\ le\ f3\ 8c\ f4\ f7$$

(hexadecimal values for each byte). An integer sequence is also specified as $a = a_1\ a_2, \ldots, a_{16}$ of length 16 by:

$$a = 5, 5, 5, 5, 3, 7, 5, 5, 5, 5, 7, 3, 5, 5, 5, 5.$$

Finally, if $Q = Q_{15}\ Q_{14}\ldots\ Q_0$ is a 16-byte string and l is an integer with $0 \leq; l < 16$, then the left shift of Q is defined by l bytes, denoted E^lQ,to be the 16 -byte string:

$$E^lQ = Q_{15-1}\ldots Q_0Q_{15}\ldots Q_{15-l+1}$$

(modulo 16 with byte subscripts is used).

For each i with $1 \leq i \leq 17$, a 16-byte value is defined as $Q^i=Q^i_{15}\ Q^i_{14}\ldots\ Q^i_0$ *by:*

$$Q^1 = K$$

$$Q^i = E^{a-1}\ Q^{i-1} \oplus D \qquad (2 \leq i \leq 17).$$

Thus, the bytes of Q^i are obtained by rotating the bytes of Q^{i-1} and then adding bit by bit modulo 2 the constant D.

Then K^i,the round key in round i is defined to be:

$$K^i = Q^i_{11}Q^i_{10}\ldots Q^i_0$$

that is, K^i is formed from the 12 least significant bytes of Q^i.

6.6 Block cipher Generation

6.6.1 Summary

The block cipher generation consists of the following phases: the initial loading of *K* and the running of the key scheduling process, the loading of *P* and the proper generation of the block cipher concerned. Thereafter, because the HURDLE-II algorithm remains in continuous encryption mode, subsequent *P* inputs can.be offered without needing to reset the key after each block cipher generation.

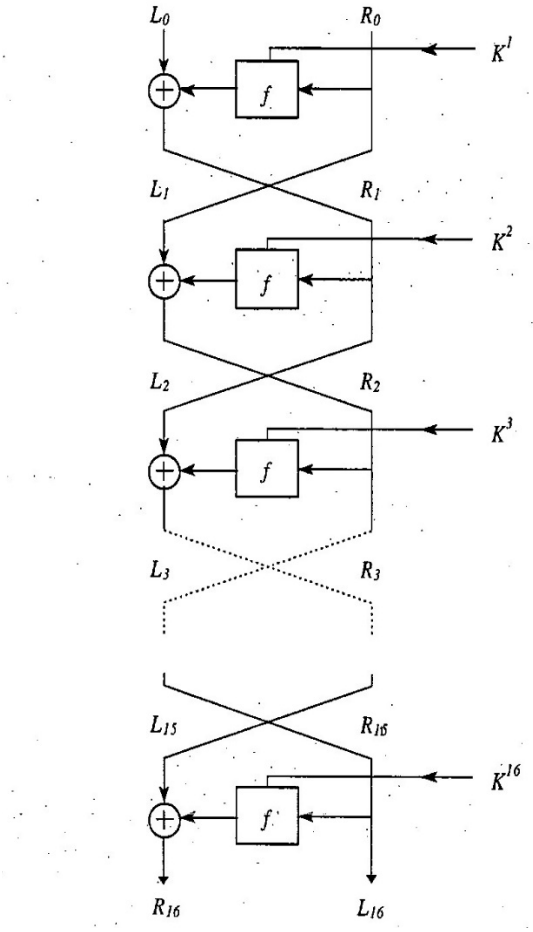
6.6.2 K loading and Key Scheduling

After loading of the 16-byte key *K* the key scheduling process is performed .to determine the sixteen 12-byte round keys K^i as described in clause 6.5.

6.6.3 Block Cipher Generation

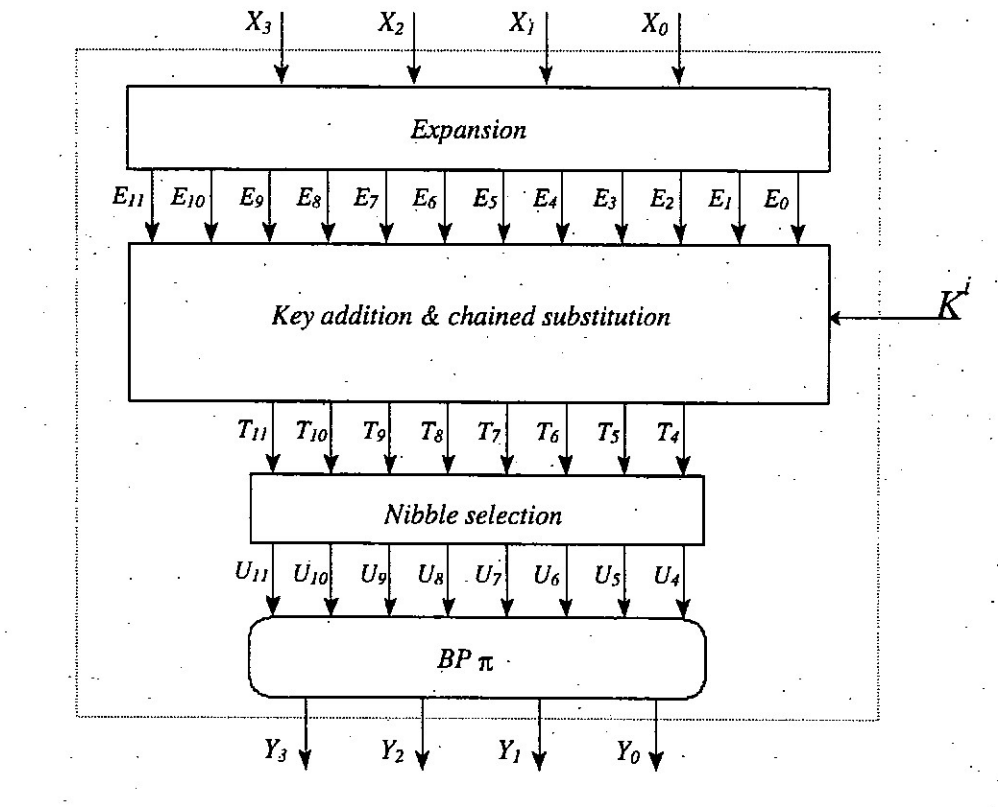
The loaded 64-bit input *P/C* is divided into two 4-byte strings and encrypted/ decrypted in sixteen rounds under the influence. of the round keys as described in clauses 6.2 and 6.3. All stages of the round function are explained in detail in clause 6.4.

6.7 Figures of HURDLE-II Algorithm



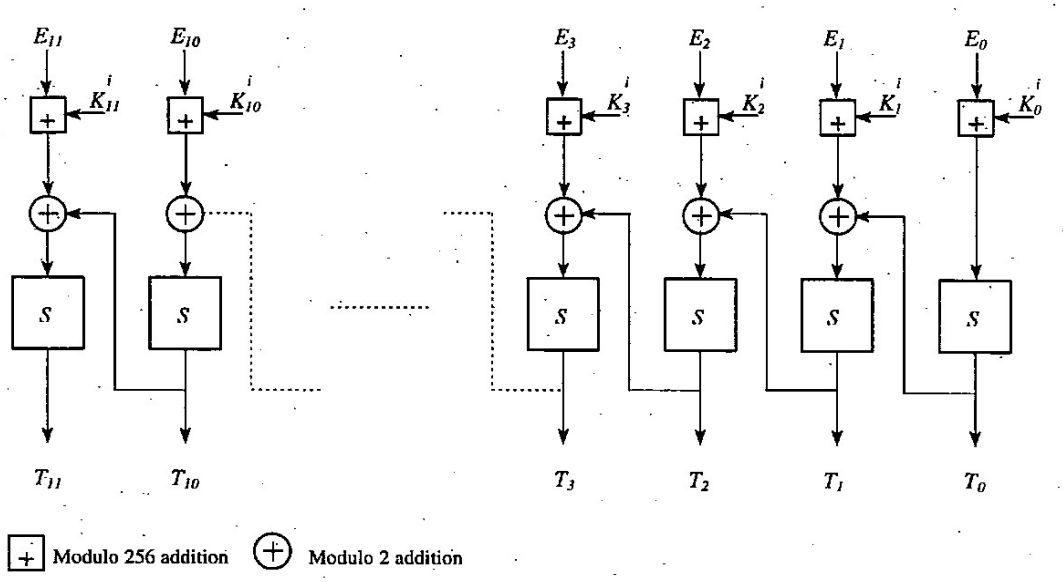
f Round function of HURDLE-II (see Figure 24)
Li, Ri Two 4-byte strings (subscript 0 = input; 16 = output)
K Round key

Figure 23: HURDLE-II block cipher



X	32-bit input string
E	12 byte expanded input data
K	2 byte round key
T	8 most significant output bytes of the key addition & chained substitution function
U_i	Rightmost nibble of T_i;
$BP\pi$	Bit permutation on 32-bit string
Y	32-bit output string

Figure 24: f Function of HURDLE-II



K^i_j Byte j of 12 byte round key K^i
s Byte permutation (256 bytes)

Figure 25: Addition of round key and chained S substitution

Table 1: The byte permutation S

f4	65	01	00	ba	7a	a7	47	98	dd	9d	ad	96	5d	aa	3d
58	c0	72	d8	66	4c	3e	e0	80	55	de	90	2a	4b	83	a0
51	39	ed	6c	8a	2c	56	60	4a	1f	d0	70	6e	33	8b	26
2e	6f	89	48	5e	40	c3	a4	a9	cf	22	50	e1	15	0c	ab
d5	f8	5f	36	04	a6	4e	92	1e	2b	88	30	93	45	67	16
8c	68	23	38	61	25	1a	81	63	cb	c1	13	41	37	0e	97
5b	ca	57	24	4d	17	c4	b9	b3	ef	8d	52	32	2f	ec	20
d9	11	d1	28	79	da	fb	e9	bb	06	77	db	fc	fe	cd	84
1d	a1	54	1b	b0	e4	cc	7c	2d	27	31	49	f5	02	69	53
4f	44	df	18	5c	0f	bc	9b	94	bd	dc	0b	a2	c7	09	ac
c6	9f	82	1c	05	46	c2	34	3c	0d	3b	ce	b7	be	08	9c
6b	ee	e5	87	af	bf	f2	eb	7b	07	64	c5	b6	ae	9a	95
35	a5	59	12	9e	a3	b8	8e	5a	f7	62	d2	3a	a8	7d	85
f6	c8	71	29	d6	d7	43	f9	78	76	73	10	91	19	0a	99
f0	e6	3f	14	f1	e2	b1	86	b4	f3	74	fa	6a	b2	21	6d
ea	b5	e7	e3	c9	d3	8f	03	75	e8	d4	42	fd	7e	ff	7f

Table 2: The bit permutation π

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

Annex A (informative): Bibliography

- ETSI TS 100 396-6: "Terrestrial Trunked Radio (TETRA); Direct Mode Operation (DMO); Part 6: Security".
- ETSI TS 101 052-1: "Rules for the management of the TETRA standard authentication and key management algorithm sets; Part 1: TAA1".

History

Document history		
V1.1.1	July 2024	Publication