

---

---

सूचना सुरक्षा — लाइटवेट क्रिप्टोग्राफी  
भाग 8 प्रामाणिक एन्क्रिप्शन

Information Security — Lightweight  
Cryptography  
Part 8 Authenticated Encryption

ICS 35.030

© BIS 2023  
©ISO/IEC 2022



भारतीय मानक ब्यूरो  
BUREAU OF INDIAN STANDARDS  
मानक भवन, 9 बहादुर शाह ज़फर मार्ग, नई दिल्ली - 110002  
MANAK BHAVAN, 9 BAHADUR SHAH ZAFAR MARG  
NEW DELHI - 110002

[www.bis.gov.in](http://www.bis.gov.in) [www.standardsbis.in](http://www.standardsbis.in)

## NATIONAL FOREWORD

This Indian Standard (Part 8) which is identical to ISO/IEC 29192-8 : 2022 'Information security — Lightweight cryptography — Part 8: Authenticated encryption' issued by the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) **was** adopted by the Bureau of Indian Standards on the recommendations of the Information Systems Security and Privacy Sectional Committee and approval of the Electronics and Information Technology Division Council.

This Indian Standard is published in several parts. The other parts in this series are:

Part 1	General
Part 2	Block ciphers
Part 3	Stream ciphers
Part 4	Mechanisms using asymmetric techniques
Part 5	Hash-functions
Part 6	Message authentication codes (MACs)
Part 7	Broadcast authentication protocols

The text of ISO/IEC standard has been approved as suitable for publication as an Indian Standard without deviations. Certain conventions are however not identical to those used in Indian Standards. Attention is particularly drawn to the following:

- a) Wherever the words 'International Standard' appears referring to this standard, they should be read as 'Indian Standard'; and
- b) Comma (,) has been used as a decimal marker while in Indian Standards, the current Practice is to use a point (.) as the decimal marker.

# Contents

Page

<b>Introduction</b> .....	<b>iv</b>
<b>1 Scope</b> .....	<b>1</b>
<b>2 Normative references</b> .....	<b>1</b>
<b>3 Terms and definitions</b> .....	<b>1</b>
<b>4 Symbols and abbreviated terms</b> .....	<b>3</b>
<b>5 Grain-128A</b> .....	<b>5</b>
5.1 Introduction to Grain-128A .....	5
5.2 Internal state .....	6
5.3 Encryption and MAC generation procedure .....	7
5.4 Decryption and MAC verification procedure .....	8
5.5 Sub-functions .....	9
5.5.1 Initialization function Init .....	9
5.5.2 MAC Initialization function Imac .....	10
5.5.3 Next-state function Next .....	11
5.5.4 Pre-output function Prt .....	11
5.5.5 Keystream function Strm .....	11
5.5.6 Function Upmac .....	12
5.5.7 Function Fmac .....	12
<b>Annex A (normative) Object identifiers</b> .....	<b>13</b>
<b>Annex B (informative) Numerical examples</b> .....	<b>14</b>
<b>Annex C (informative) Security considerations</b> .....	<b>16</b>
<b>Bibliography</b> .....	<b>17</b>

## Introduction

This document specifies authenticated encryption tailored for implementation in constrained environments. Data transmitted from one party to another is often vulnerable against various attacks such as eavesdropping or malicious alterations. Similarly, data at rest usually requires protection.

Encryption mechanisms as specified in the ISO/IEC 18033 series and ISO/IEC 10116 provide solutions against eavesdropping. Integrity protection is usually guaranteed with a message authentication code (MAC) algorithm, such as those defined in the ISO/IEC 9797 series. In addition, ISO/IEC 19772 describes several authenticated encryption mechanisms, that is to say mechanisms that efficiently combine the encryption and MAC operations.

Nonetheless, some applications including radiofrequency identification (RFID) tags, smart cards, secure batteries, health-care systems and sensor networks, encounter several constraints. Chip area, energy consumption, execution time, program code, RAM size and communication bandwidth are typically critical for the applications listed above. The ISO/IEC 29192 series specifies lightweight cryptography suitable for these constrained environments. ISO/IEC 29192-2 and ISO/IEC 29192-3 respectively define lightweight block ciphers and stream ciphers. Both can be used to provide confidentiality. Regarding protection against alteration, lightweight MAC algorithms are defined in ISO/IEC 29192-6.

In this document, lightweight authenticated encryption mechanisms are defined. Similar to ISO/IEC 19772, they provide confidentiality, integrity and optionally data origin authentication. They differ from those specified in the aforementioned document, in that they have been specifically designed for constrained environments.

This document specifies a unique method. In the future, other methods may be added to this document, including lightweight authenticated encryption with additional data (AEAD) methods, based either on block ciphers or stream ciphers.

*Indian Standard***INFORMATION SECURITY — LIGHTWEIGHT CRYPTOGRAPHY  
PART 8 AUTHENTICATED ENCRYPTION****1 Scope**

This document specifies one method for authenticated encryption suitable for applications requiring lightweight cryptographic mechanisms.

This method processes a data string with the following security objectives:

- a) data confidentiality, i.e. protection against unauthorized disclosure of data,
- b) data integrity, i.e. protection that enables the recipient of data to verify that it has not been modified.

Optionally, this method can provide data origin authentication, i.e. protection that enables the recipient of data to verify the identity of the data originator.

The method specified in this document is based on a lightweight stream cipher, and requires the parties of the protected data to share a secret key for this algorithm. Key management is outside the scope of this document.

NOTE Key management techniques are defined in the ISO/IEC 11770 series.

**2 Normative references**

There are no normative references for this document.

**3 Terms and definitions**

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

**3.1****authenticated encryption**

(reversible) transformation of data by a cryptographic algorithm to produce ciphertext that cannot be altered by an unauthorized entity without detection, i.e. it provides data confidentiality, data integrity, and optionally data origin authentication

[SOURCE: ISO/IEC 19772:2020, 3.2, modified — The definition was slightly modified to make the data origin authentication optional.]

### 3.2 **authenticated encryption mechanism**

cryptographic technique used to protect the confidentiality, guarantee the integrity of data and optionally the data origin and which consists of two component processes: an *encryption* (3.6) algorithm and a *decryption* (3.5) algorithm

[SOURCE: ISO/IEC 19772:2020, 3.3, modified — The definition was slightly modified to make the data origin authentication optional.]

### 3.3 **ciphertext** data which has been transformed to hide its information content

[SOURCE: ISO/IEC 18033-1:2021, 3.7]

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

[SOURCE: ISO/IEC 9797-1:2011, 3.4]

### 3.5 **decryption** reversal of a corresponding *encryption* (3.6)

[SOURCE: ISO/IEC 9797-1:2011, 3.5]

### 3.6 **encryption** reversible operation by a cryptographic algorithm converting data into *ciphertext* (3.3) so as to hide the information content of the data

[SOURCE: ISO/IEC 9797-1:2011, 3.6]

### 3.7 **initialization value** value used in defining the starting point of an *encryption* (3.6) process

[SOURCE: ISO/IEC 18033-4:2011, 3.7]

### 3.8 **key** sequence of symbols that controls the operation of a cryptographic transformation

[SOURCE: ISO/IEC 9797-1:2011, 3.7, modified — Note was removed.]

### 3.9 **keystream function** function that takes as input, the current *state* (3.17) of the *keystream generator* (3.10) and (optionally) part of the previously generated *ciphertext* (3.3), and gives as output the next part of the keystream

[SOURCE: ISO/IEC 18033-4:2011, 3.9]

### 3.10 **keystream generator** state-based process (i.e. as a finite state machine) that takes as input, a *key* (3.8), an *initialization value* (3.7), and if necessary the *ciphertext* (3.3), and gives as output a keystream (i.e. a sequence of bits or blocks of bits) of arbitrary length

[SOURCE: ISO/IEC 18033-4:2011, 3.10, modified — "initialization vector" changed to "initialization value".]

### 3.11 message authentication code MAC

string of bits which is the output of a *MAC algorithm* (3.12)

[SOURCE: ISO/IEC 9797-1:2011, 3.9, modified — Note was removed.]

### 3.12 message authentication code algorithm MAC algorithm

algorithm for computing a function which maps strings of bits and a *secret key* (3.16) to fixed-length strings of bits, satisfying the following two properties:

- for any key and any input string, the function can be computed efficiently;
- for any fixed key, and given no prior knowledge of the key, it is computationally infeasible to compute the function value on any new input string, even given knowledge of a set of input strings and corresponding function values, where the value of the *i*th input string might have been chosen after observing the value of the first *i*-1 function values (for integers *i* > 1)

[SOURCE: ISO/IEC 9797-1:2011, 3.10, modified — Notes were removed]

### 3.13 next-state function

function that takes as input, the current *state* (3.17) of the *keystream generator* (3.10) and (optionally) part of the previously generated *ciphertext* (3.3), and gives as output a new *state* (3.17) for the *keystream generator* (3.10)

[SOURCE: ISO/IEC 18033-4:2011, 3.12]

### 3.14 plaintext

cleartext  
unencrypted information

[SOURCE: ISO/IEC 18033-1:2021, 3.20]

### 3.15 pre-output stream

pseudo-random bits, which are used for the *encryption* (3.6) and the *decryption* (3.5) of the message, and the generation of the *message authentication code* (3.11)

### 3.16 secret key

*key* (3.8) used with symmetric cryptographic techniques by a specified set of entities

[SOURCE: ISO/IEC 18033-1:2021, 3.25]

### 3.17 state

internal state of a *keystream generator* (3.10)

[SOURCE: ISO/IEC 29192-3:2012, 3.12]

## 4 Symbols and abbreviated terms

For the purposes of this document, the following symbols and abbreviated terms apply.

ACCU      Dedicated accumulator register for the MAC (*t* bits).

<i>AM</i>	Authenticated message, the concatenation of the ciphertext <i>C</i> and the <i>MAC</i> . $AM = C    MAC$ .
AND	Bitwise logical AND operation.
<i>AUTH</i> <sup>(<i>t</i>)</sup>	Dedicated register for the MAC computation.
<i>a<sub>i</sub></i>	Variable forming part of the internal state of a keystream generator.
<i>b<sub>i</sub></i>	Variable forming part of the internal state of a keystream generator.
<i>C</i>	Ciphertext.
<i>C<sub>i</sub></i>	Ciphertext bit.
<b>Fmac</b>	Function which finalizes the MAC computation.
<b>Imac</b>	Function which initializes the MAC registers.
<b>Init</b>	Function which generates the initial internal state of a keystream generator.
<i>IV</i>	Initialization value.
<i>K</i>	Key.
<i>l</i>	Length of a plaintext or ciphertext block (in bits).
Len	Function that returns the number of bits in a string.
LFSR	Linear feedback shift register.
MAC	Message authentication code. MAC is a <i>t</i> -bit string.
<i>M<sub>i</sub></i>	Message bit.
<i>n</i>	Length of the authenticated message (AM) (in bits).
<b>Next</b>	Next-state function of a keystream generator.
NLFSR	Nonlinear feedback shift register.
OR	Bitwise logical OR operation.
<i>P</i>	Plaintext.
<i>P<sub>i</sub></i>	Plaintext bit.
<b>Prt</b>	Function that generates a pre-output of the stream cipher.
<i>r<sub>i</sub></i>	Variable forming part of the internal state of a keystream generator.
SHIFT	Dedicated shift register for the MAC ( <i>t</i> bits).
<i>s<sub>i</sub></i>	Variable forming part of the internal state of a keystream generator.
<b>Strm</b>	Keystream function of a keystream generator.
<i>S</i> <sup>(<i>t</i>)</sup>	Internal state of keystream generator.
<i>t</i>	MAC length (in bits).
<b>Upmac</b>	Function which updates the MAC registers.



$Y$	Pre-output stream.
$Y^{(i)}$	Pre-output bit.
$Z$	Keystream.
$Z^{(i)}$	Keystream bit.
$0^i$	Block of $i$ zero bits.
$1^i$	Block of $i$ one bits.
$\oplus$	Bitwise XOR (eXclusive OR) operation.
$\parallel$	Concatenation of strings, i.e. if $A$ and $B$ are blocks of data, then $A\parallel B$ is the block obtained concatenating $A$ and $B$ in the order specified.
$X _s$	Left-truncation of the block of bits $X$ : if $X$ has a bit-length greater than or equal to $s$ , then $X _s$ is the $s$ -bit block consisting of the left-most $s$ bits of $X$ .
$X ^s$	Right-truncation of the block of bits $X$ : if $X$ has a bit-length greater than or equal to $s$ , then $X ^s$ is the $s$ -bit block consisting of the right-most $s$ bits of $X$ .

## 5 Grain-128A

### 5.1 Introduction to Grain-128A

Grain-128A is a synchronous stream cipher with an add-on module that generates a message authentication code (MAC).

It is composed of two sub-modules that work conjointly:

- a stream cipher module that generates the key stream for the encryption/decryption of the message;
- a MAC module that constitutes the MAC algorithm.

Grain-128A has a 128-bit long key,  $K$ , and a 96-bit initialization value,  $IV$ . It generates a  $t$ -bit MAC.

As a precondition, the recipients will have received  $K$  and  $IV$  in a secure way as pre-shared parameters.

For this mechanism,  $t$  shall be at least equal to 32 and the MAC shall apply to the entire plaintext.

After the initialization of the system with the  $K$  and the  $IV$ , the cipher generates a pre-output stream. This pre-output stream is split into two parts:

- the even bits compose the keystream to encrypt/decrypt the message;
- the odd bits are used to generate the MAC.

NOTE 1 Grain-128A document [12] defines two modes of operation: with or without a MAC. The MAC is disabled when  $IV_0 = 0$  and conversely enabled when  $IV_0 = 1$ . This document only specifies the authenticated mode, i.e. the MAC is always supported. Accordingly, the value of  $IV_0$  is fixed to '1'.

As a synchronous keystream generator, Grain-128A follows the general models of stream ciphers defined in ISO/IEC 18033-4, with supplementary functions to take the MAC generation into consideration.

Grain-128A finite-state machine is defined by:

- an initialization function, **Init**, which takes as input a key,  $K$ , and an initialization value,  $IV$ , and outputs an initial state  $S^{(-2^*t)}$  for the keystream generator;

NOTE 2 As for any stream cipher, the uniqueness of the *IV* for a given key, *K*, is a crucial requirement. The reuse of an *IV* under the same key leads to trivial attacks causing plaintext recovery and impersonation.

- b) a MAC initialization function, **Imac**, which initializes the MAC registers;
- c) a next-state function, **Next**, which takes as input the current state of the pre-output stream generator  $S^{(i)}$ , and outputs the next state of the pre-output stream generator  $S^{(i+1)}$ ;
- d) a pre-output function, **Prt**, which takes as input a state of the pre-output generator  $S^{(i)}$ , and outputs pre-output bits  $Y^{(i)}$ ;
- e) a keystream function, **Strm**, that outputs a key stream bit  $Z^{(i)}$ ;
- f) an update MAC function, **Upmac**, which takes as input the current MAC registers, a pre-output bit  $Y^{(i)}$ , a bit of the message  $M_i$  and outputs the next state of the MAC registers;
- g) a final MAC function, **Fmac**, which completes the MAC computation and outputs the MAC of the message.

The output function is the binary-additive function. That is to say, the operation to combine the plaintext with the key stream is the bitwise XOR.

The encryption of plaintext bit  $P_i$  by a keystream bit  $Z^{(i)}$  is given by:

$$C_i = P_i \oplus Z^{(i)}$$

Inversely, the decryption of a ciphertext  $C_i$  by a keystream bit  $Z^{(i)}$  is given by:

$$P_i = C_i \oplus Z^{(i)}$$

[Annex B](#) provides some numerical examples for  $t = 32$  and  $t = 64$ . [Annex A](#) defines object identifiers that shall be used to identify the lightweight authenticated encryption algorithms specified in this document. [Annex C](#) exposes some security considerations on Grain-128A.

## 5.2 Internal state

Grain-128A state is composed of two sub-modules.

- a) The state variable  $S^{(i)}$  which is subdivided into two 128-bit registers:

$$S^{(i)} = (NLFSR^{(i)}, LFSR^{(i)})$$

where

$$NLFSR^{(i)} = (b_0^{(i)}, b_1^{(i)}, \dots, b_{127}^{(i)})$$

$$LFSR^{(i)} = (s_0^{(i)}, s_1^{(i)}, \dots, s_{127}^{(i)})$$

where  $b_j$  and  $s_j$  are bits (for  $j = 0, 1, 2, \dots, 127$ ).

- b) The MAC variable  $AUTH^{(i)}$  which is subdivided into two  $t$ -bit registers:

$$AUTH^{(i)} = (ACCU^{(i)}, SHIFT^{(i)})$$

where

$$ACCU^{(i)} = (a_0^{(i)}, a_1^{(i)}, \dots, a_{t-1}^{(i)})$$

$$SHIFT^{(i)} = (r_0^{(i)}, r_1^{(i)}, \dots, r_{t-1}^{(i)})$$

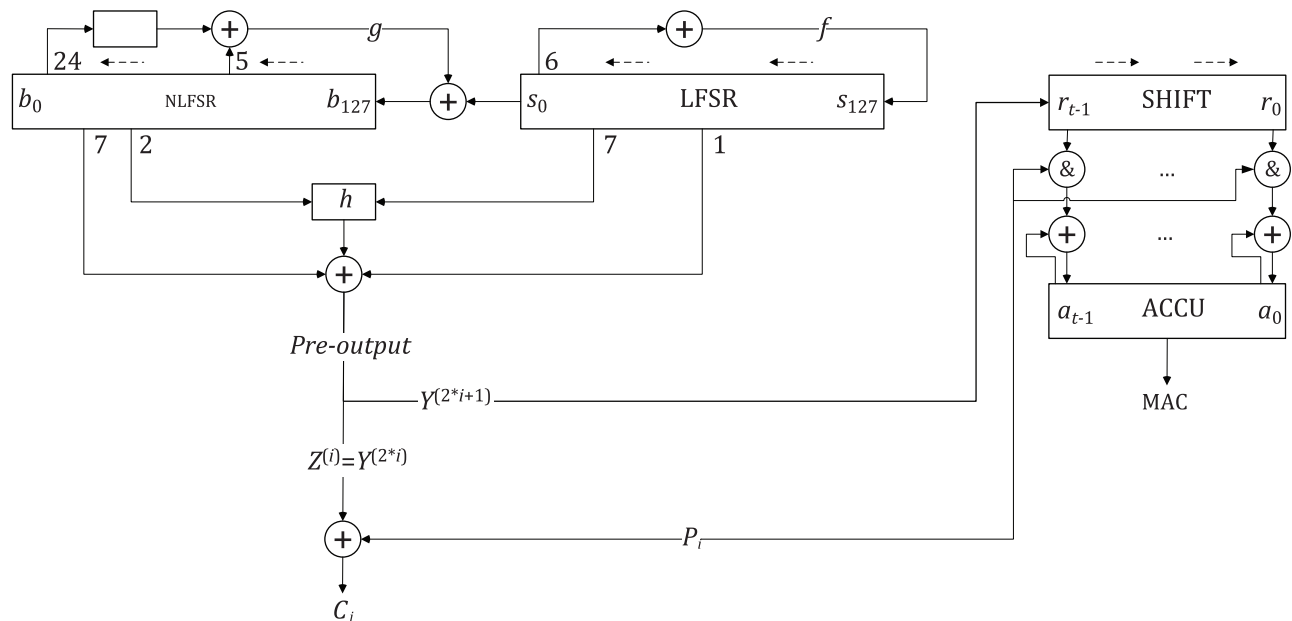
where  $a_j$  and  $r_j$  are bits (for  $j = 0, 1, 2, \dots, t-1$ )

[Figure 1](#) illustrates the building blocks of Grain-128A. The algorithm is composed of four registers: NLFSR, LFSR, SHIFT and ACCU registers.

On the top left of [Figure 1](#), the feedbacks of the registers LFSR and NLFSR are outlined together with the generation of the pre-output stream. The small figures on top of and below the LFSR and NLFSR registers indicate the number of bits extracted from the registers. The operations are fully described in the sub-functions **Next**, **Strm** and **Prt**.

[Figure 1](#) also shows how the pre-output stream is split in two sub-streams. Even bits compose the keystream to encrypt the data, while odd bits update the MAC module.

On the right of [Figure 1](#), the SHIFT and the ACCU registers compose the module dedicated to the MAC generation.



**Figure 1 — An overview of the building blocks of Grain-128A**

### 5.3 Encryption and MAC generation procedure

The originator shall perform the following steps to protect a data string  $P$ .

Inputs:

- $P = P_0 P_1 \dots P_{l-1}$ , an  $l$ -bit-string to encrypt and authenticate
- $IV$  is a 96-bit string that was pre-shared between recipients
- The 128-bit key,  $K$ , that was securely shared beforehand.

Output:  $AM = C_0 C_1 \dots C_{l-1} \parallel MAC$ , the authenticated encryption  $(l+t)$ -bit string of  $P$ .

- a) Initialize the state variable with key,  $K$ , and the initialization value,  $IV$ :
  - $S^{(-2*t)} = \mathbf{Init}(K, IV)$
- b) Initialize the MAC variable:
  - $(AUTH^{(0)}, S^{(0)}) = \mathbf{Imac}(S^{(-2*t)})$

- c) For every bit of the message  $P_i$ , generate a keystream bit  $Z^{(i)}$  and update the MAC state.
- For  $i = 0, 1, \dots, l-1$ :
    - Encrypt the message bit:
      - $Y^{(2*i)} = \mathbf{Prt}(S^{(2*i)})$
      - $Z^{(i)} = \mathbf{Strm}(Y^{(2*i)})$
      - $C_i = P_i \oplus Z^{(i)}$
      - $S^{(2*i+1)} = \mathbf{Next}(S^{(2*i)})$
    - Update the MAC variable:
      - $Y^{(2*i+1)} = \mathbf{Prt}(S^{(2*i+1)})$
      - $AUTH^{(i+1)} = \mathbf{Upmac}(AUTH^{(i)}, Y^{(2*i+1)}, P_i)$
      - $S^{(2*i+2)} = \mathbf{Next}(S^{(2*i+1)})$
- d) Finalize the MAC computation:
- $MAC = \mathbf{Fmac}(AUTH^{(l)})$
- e) Concatenate the encrypted message  $C$  and the  $MAC$  to produce the  $(l+t)$ -bit string  $AM$ , the authenticated-encrypted version of  $P$ .
- $AM = C || MAC$ , where  $C = C_0 C_1 \dots C_{l-1}$  is an  $l$ -bit-string.
- f) Output  $AM$ .

#### 5.4 Decryption and MAC verification procedure

The recipient shall perform the following steps to decrypt and verify an  $(l+t)$ -bit authenticated-encrypted string  $AM$ .

Inputs:

- $AM = C_0 C_1 \dots C_{l-1} || MAC$ , where  $AM$  is an  $l+t$ -bit string.
- $IV$  is a 96-bit string that was pre-shared between recipients.
- The 128-bit key,  $K$ , that was securely shared beforehand.

Output:  $P = P_0 P_1 \dots P_{l-1}$ , the decryption of  $C$  or INVALID in case of incorrect  $MAC$

- a) If the length of  $AM$  is less than  $t$  bits then halt and output INVALID. Otherwise let:
- $n = \text{Len}(AM)$
  - $l = n - t$
  - $C = AM|_l$ , where  $C = C_0 C_1 \dots C_{l-1}$  and where  $C_i$  are bits for  $i = 0, \dots, l-1$
  - $MAC' = AM|_t$
- b) Initialize the state variable with key,  $K$ , and the initialization value,  $IV$ :
- $S^{(-2*t)} = \mathbf{Init}(K, IV)$
- c) Initialize the MAC variable:
- $(AUTH^{(0)}, S^{(0)}) = \mathbf{Imac}(S^{(-2*t)})$

- d) For every bit of the message  $C_i$ , generate a key stream bit  $Z^{(i)}$  and update the MAC state.
- For  $0, 1, \dots, l-1$ :
    - Decrypt the message bit:
      - $Y^{(2^*i)} = \mathbf{Prt}(S^{(2^*i)})$
      - $Z^{(i)} = \mathbf{Strm}(Y^{(2^*i)})$
      - $P_i = C_i \oplus Z^{(i)}$
      - $S^{(2^*i+1)} = \mathbf{Next}(S^{(2^*i)})$
    - Update the MAC variable:
      - $Y^{(2^*i+1)} = \mathbf{Prt}(S^{(2^*i+1)})$
      - $AUTH^{(i+1)} = \mathbf{Upmac}(AUTH^{(i)}, Y^{(2^*i+1)}, P_i)$
      - $S^{(2^*i+2)} = \mathbf{Next}(S^{(2^*i+1)})$
- e) Finalize the MAC computation:
- $MAC = \mathbf{Fmac}(AUTH^{(l)})$
- f) Compare  $MAC'$  and the computed  $MAC$ :
- If  $MAC = MAC'$  continue, otherwise halt and output INVALID.
- g) Output the  $l$ -bit string  $P = P_0 P_1 \dots P_{l-1}$ .

## 5.5 Sub-functions

### 5.5.1 Initialization function Init

The internal state  $S^{(i)}$  of Grain-128A is initialized using the following **Init** function. The NLFSR is fed with the key. The LFSR is fed with the initialization value  $IV$  appended with a 32-bit fixed padding value  $1^{31}0^1$ . The first bit of the  $IV$  is forced to 1 to indicate the authenticated mode ( $IV_0 = 1$ ). As a warm-up, the state is clocked 256 times where the pre-output is reinjected into the two shift registers LFSR and NLFSR.

Inputs: 128-bit key  $K$ , 96-bit initialization value,  $IV$ .

Output: The state variable  $S^{(-2^*t)}$

- a) Set the  $NLFSR^{(-256-2^*t)}$  registers as follows:
- For  $i = 0, \dots, 127$ , set  $b_i^{(-256-2^*t)} = K_i$
- b) Set the  $LFSR^{(-256-2^*t)}$  register as follows:
- For  $i = 0, \dots, 95$ , set  $s_i^{(-256-2^*t)} = IV_i$
  - For  $i = 96, \dots, 126$ , set  $s_i^{(-256-2^*t)} = 1$
  - $s_{127}^{(-256-2^*t)} = 0$
- c) Clock 256 times
- For  $i = 0, \dots, 255$ :
    - $Y^{(-256-2^*t+i)} = \mathbf{Prt}(S^{(-256-2^*t+i)})$

$$S^{(-255-2^*t+i)} = \mathbf{Next}(S^{(-256-2^*t+i)})$$

$$b_{127}^{(-255-2^*t+i)} = b_{127}^{(-256-2^*t+i)} \oplus Y^{(-256-2^*t+i)}$$

$$s_{127}^{(-255-2^*t+i)} = s_{127}^{(-256-2^*t+i)} \oplus Y^{(-256-2^*t+i)}$$

d) Output:  $S^{(-2^*t)}$

Figure 2 illustrates the function **Init**. It shows how the two registers LFSR and NLFSR are updated and how the pre-output stream is generated. The double lines indicate that the pre-output stream bits are reinjected into the LFSR and the NLFSR during the **Init** procedure.

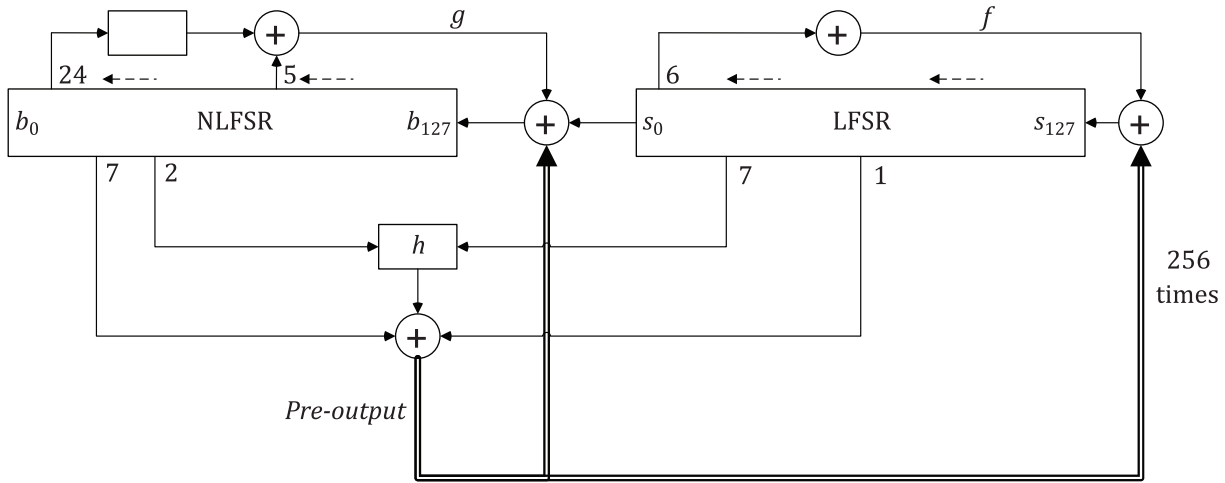


Figure 2 — Initialization of Grain-128A

### 5.5.2 MAC Initialization function **Imac**

The MAC state  $AUTH^{(i)}$  is initialized using the following **Imac** function. The ACCU buffer is set with the  $t$  first pre-output bits  $Y^{(i)}$  generated after the initialization. The SHIFT register is set with the next  $t$  pre-output bits  $Y^{(i)}$ . It also updates the state  $S$ .

Input:  $S^{(-2^*t)}$

Outputs:  $AUTH^{(0)}$ ,  $S^{(0)}$

a) Set the  $ACCU^{(0)}$  register as follows:

— For  $i = 0, \dots, t-1$ :

$$a_i^{(0)} = \mathbf{Prt}(S^{(-2^*t+i)})$$

$$S^{(-2^*t+i+1)} = \mathbf{Next}(S^{(-2^*t+i)})$$

b) Set the  $SHIFT^{(0)}$  register as follows:

— For  $i = 0, \dots, t-1$ :

$$r_i^{(0)} = \mathbf{Prt}(S^{(-t+i)})$$

$$S^{(-t+i+1)} = \mathbf{Next}(S^{(-t+i)})$$

Output:  $AUTH^{(0)}$  and  $S^{(0)}$ .

### 5.5.3 Next-state function Next

LFSR is a linear shift register with the feedback polynomial  $f$ .

$$f(X) = 1 + X^{32} + X^{47} + X^{58} + X^{90} + X^{121} + X^{128}$$

NLFSR is a nonlinear shift register with the feedback polynomial  $g$ .

$$g(X) = 1 + X^{32} + X^{37} + X^{72} + X^{102} + X^{128} + X^{44}X^{60} + X^{61}X^{125} + X^{63}X^{67} + X^{69}X^{101} + X^{80}X^{88} + X^{110}X^{111} + X^{115}X^{117} + X^{46}X^{50}X^{58} + X^{103}X^{104}X^{106} + X^{33}X^{35}X^{36}X^{40}$$

In addition,  $s_0^{(i)}$  bit of the LFSR is injected into the NLFSR adding  $s_0^{(i)}$  to the NLFSR feedback  $g$ .

Input:  $S^{(i)}$ .

Output:  $S^{(i+1)}$ .

a) Compute the feedback  $g$  of NLFSR<sup>(i)</sup> as follows:

$$\begin{aligned} - \quad b_{127}^{(i+1)} = & s_0^{(i)} \oplus b_0^{(i)} \oplus b_{26}^{(i)} \oplus b_{56}^{(i)} \oplus b_{91}^{(i)} \oplus b_{96}^{(i)} \oplus (b_3^{(i)} \text{ AND } b_{67}^{(i)}) \oplus (b_{11}^{(i)} \text{ AND } b_{13}^{(i)}) \\ & \oplus (b_{17}^{(i)} \text{ AND } b_{18}^{(i)}) \oplus (b_{27}^{(i)} \text{ AND } b_{59}^{(i)}) \oplus (b_{40}^{(i)} \text{ AND } b_{48}^{(i)}) \oplus (b_{61}^{(i)} \text{ AND } b_{65}^{(i)}) \oplus (b_{68}^{(i)} \text{ AND } \\ & b_{84}^{(i)}) \oplus (b_{88}^{(i)} \text{ AND } b_{92}^{(i)} \text{ AND } b_{93}^{(i)} \text{ AND } b_{95}^{(i)}) \oplus (b_{22}^{(i)} \text{ AND } b_{24}^{(i)} \text{ AND } b_{25}^{(i)}) \oplus (b_{70}^{(i)} \text{ AND } \\ & b_{78}^{(i)} \text{ AND } b_{82}^{(i)}) \end{aligned}$$

b) Compute the feedback  $f$  of LFSR<sup>(i)</sup> as follows:

$$- \quad s_{127}^{(i+1)} = s_0^{(i)} \oplus s_7^{(i)} \oplus s_{38}^{(i)} \oplus s_{70}^{(i)} \oplus s_{81}^{(i)} \oplus s_{96}^{(i)}$$

c) Shift the two registers:

$$- \quad \text{For } j = 0 \text{ to } 126, \text{ set } b_j^{(i+1)} = b_{j+1}^{(i)}$$

$$- \quad \text{For } j = 0 \text{ to } 126, \text{ set } s_j^{(i+1)} = s_{j+1}^{(i)}$$

d) Output  $S^{(i+1)}$ .

### 5.5.4 Pre-output function Prt

The **Prt** function generates a pre-output bit  $Y^{(i)}$  from the state  $S^{(i)}$  using a Boolean function.

Input:  $S^{(i)}$ .

Output: Pre-output bit  $Y^{(i)}$ .

a) Compute  $h$  as follows:

$$- \quad h = (b_{12}^{(i)} \text{ AND } s_8^{(i)}) \oplus (s_{13}^{(i)} \text{ AND } s_{20}^{(i)}) \oplus (b_{95}^{(i)} \text{ AND } s_{42}^{(i)}) \oplus (s_{60}^{(i)} \text{ AND } s_{79}^{(i)}) \oplus (b_{12}^{(i)} \text{ AND } b_{95}^{(i)} \text{ AND } s_{94}^{(i)})$$

b) Compute  $Y^{(i)}$  as follows:

$$- \quad Y^{(i)} = h \oplus b_2^{(i)} \oplus b_{15}^{(i)} \oplus b_{36}^{(i)} \oplus b_{45}^{(i)} \oplus b_{64}^{(i)} \oplus b_{73}^{(i)} \oplus b_{89}^{(i)} \oplus s_{93}^{(i)}$$

c) Output  $Y^{(i)}$ .

### 5.5.5 Keystream function Strm

The **Strm** function generates the keystream bit  $Z^{(i)}$  to add with the message to encrypt or decrypt.

Input:  $Y^{(2*i)}$ .

Output:  $Z^{(i)}$ .

a) Compute the keystream bit  $Z^{(i)}$

—  $Z^{(i)} = Y^{(2^*i)}$

b) Output  $Z^{(i)}$ .

### 5.5.6 Function Upmac

The MAC registers are updated as a function of the message bit  $M_i$  and the pre-output bit  $Y^{(2^*i+1)}$ .

Inputs:  $AUTH^{(i)}$ ,  $Y^{(2^*i+1)}$ ,  $M_i$ .

Output:  $AUTH^{(i+1)}$ .

a) Update the  $ACCU^{(i)}$  register:

— For  $j = 0$  to  $t-1$ , set  $a_j^{(i+1)} = a_j^{(i)} \oplus (M_i \text{ AND } r_j^{(i)})$

b) Update the  $SHIFT^{(i)}$  register:

— For  $j = 0$  to  $t-2$ , set  $r_j^{(i+1)} = r_{j+1}^{(i)}$

—  $r_{t-1}^{(i+1)} = Y^{(2^*i+1)}$

c) Output  $AUTH^{(i+1)}$ .

### 5.5.7 Function Fmac

The **Fmac** function completes the MAC computation. It pads the message with an additional bit set to 1, which is equivalent to add bitwise the  $SHIFT$  and the  $ACCU$  registers. The final MAC is the new value of the  $ACCU$  register.

Input:  $AUTH^{(i)}$ .

Output:  $MAC$

a) Update the  $ACCU^{(i)}$  register as follows:

— For  $j = 0$  to  $t-1$ , set  $a_j^{(i+1)} = a_j^{(i)} \oplus r_j^{(i)}$

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



## Annex A (normative)

### Object identifiers

This annex lists the object identifiers assigned to the lightweight authenticated encryption algorithms specified in this document.

```

LightweightCryptography-8 {
iso(1) standard(0) lightweight-cryptography(29192) part8(8)
asn1-module(0) algorithm-object-identifiers(0)

DEFINITIONS EXPLICIT TAGS ::= BEGIN
-- EXPORTS All; --
-- IMPORTS None; --

OID ::= OBJECT IDENTIFIER -- Alias

-- Synonyms --
is29192-8 OID ::= {iso(1) standard(0) lightweight-cryptography(29192) part8(8)}

-- Lightweight authenticated encryption mechanisms

lightweight-Authenticated-Encryption OID ::= {is29192-8 mechanism(1)}

grain-128A OID ::= {lightweight-Authenticated-Encryption 1}

LightweightCryptographyIdentifier ::= SEQUENCE {
    algorithm ALGORITHM.&id({AuthenticatedEncryptionAlgorithms}),
    parameters
ALGORITHM.&Type({AuthenticatedEncryptionAlgorithms }{@algorithm})
    OPTIONAL
}

authenticatedEncryption ALGORITHM ::= {
    PARMS MacLengthID
    DYN-PARMS InitializationVector
    IDENTIFIED BY grain-128A }

AuthenticatedEncryptionAlgorithms ALGORITHM ::= {
    authenticatedEncryption, ... }

MacLength ::= INTEGER

MacLengthID ::= CHOICE {
    int MacLength,
    oid OID
}

InitializationVector ::= OCTET STRING (SIZE (12))

-- ALGORITHM information object class
ALGORITHM ::= CLASS {
    &Type OPTIONAL,
    &DynParms OPTIONAL,
    &id OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
    [PARMS &Type]
    [DYN-PARMS&DynParms ]
    IDENTIFIED BY &id }

END

```

## Annex B (informative)

### Numerical examples

#### B.1 General

This annex provides numerical examples for each mechanism in big-endian hexadecimal notation.

#### B.2 Numerical examples of Grain-128A

##### Case $t = 32$

*Key* =    00000000    00000000    00000000    00000000

*IV* =    00000000    00000000    00000000

—	message =	(0 octet)	Ciphertext =	(0 octet)	tag = 4ff6a6c1
—	message = 00	(1 octet)	Ciphertext = 0d	(1 octet)	tag = ebdbd53e
—	message = ff	(1 octet)	Ciphertext = f2	(1 octet)	tag = 77c0fb94
—	message = 1234	(2 octets)	Ciphertext = 1f1f	(2 octets)	tag = ccf86228
—	message = 123456789a	(5 octets)	Ciphertext = 1f1f495626	(5 octets)	tag = 678f3c3f

*Key* =    01234567    89abcdef    fedcba98    76543210

*IV* =    ccbbaa99    88776655    44332211

—	message =	(0 octet)	Ciphertext =	(0 octet)	tag = 8af0c528
—	message = 00	(1 octet)	Ciphertext = 5b	(1 octet)	tag = b1cd3942
—	message = ff	(1 octet)	Ciphertext = a4	(1 octet)	tag = a7266d64
—	message = 1234	(2 octets)	Ciphertext = 4953	(2 octets)	tag = 505c31a2
—	message = 123456789a	(5 octets)	Ciphertext = 4953a8b691	(5 octets)	tag = 8d177f5f

##### Case $t = 64$

*Key* =    00000000    00000000    00000000    00000000

*IV* =    00000000    00000000    00000000

— message =	(0 octet)	Ciphertext =	(0 octet)
tag = 57b96fed 4b02cd4a			
— message = 00	(1 octet)	Ciphertext = bc	(1 octet)
tag = a412f970 a6e03906			
— message = ff	(1 octet)	Ciphertext = 43	(1 octet)
tag = 0a8b8b04 0241953d			
— message = 1234	(2 octets)	Ciphertext = aeb7	(2 octets)
tag = 6c1074bb 921726e0			
— message = 123456789a	(5 octets)	Ciphertext = aeb78c06fc	(5 octets)
tag = d26ecba2 9b945971			

Key =   01234567       89abcdef       fedcba98       76543210  
 IV =    ccbbaa99       88776655       44332211

— message =	(0 octet)	Ciphertext =	(0 octet)
tag = 7a87686f 7c0075c1			
— message = 00	(1 octet)	Ciphertext = 0b	(1 octet)
tag = c6607eae 3b483d93			
— message = ff	(1 octet)	Ciphertext = f4	(1 octet)
tag = adc28cee f98ffa5d			
— message = 1234	(2 octets)	Ciphertext = 1997	(2 octets)
tag = f53a3b4c 43b2e476			
— message = 123456789a	(5 octets)	Ciphertext = 1997270f22	(5 octets)
tag = be9ea6a7 ae4bee82			

## Annex C (informative)

### Security considerations

Grain-128A is an all-in-one algorithm that provides authenticated encryption.

Assuming the  $IV$  is not reused, it claims an authentication security level of  $2^t$  where  $t$  is the length of the MAC. There is currently no known attack on Grain-128A authenticated mode that recovers the key or the state in a more efficient way than a simple exhaustive key search attack. Security level of encryption is 128.

The algorithm was designed for extremely constrained environments. The size of the MAC will be chosen to obtain the best trade-off performance/security.

For most lightweight applications,  $t = 64$  is strongly recommended and offers a suitable protection against guessing the MAC and against forgery attacks.

For extremely optimized applications, for example when the payload size is restricted or when the bandwidth is very limited,  $t = 32$  may be considered. In that case, it is recommended to restrict the number of computations with a given key. The duration of the messages validity should also be analysed with care.

Grain-128A targets lightweight applications and does not target a quantum-safe security level.

## Bibliography

- [1] ISO/IEC 9797 (all parts), *Information technology — Security techniques — Message authentication codes (MACs)*
- [2] ISO/IEC 10116:2017, *Information technology — Security techniques — Modes of operation for an n-bit block cipher*
- [3] ISO/IEC 11770 (all parts), *Information security — Key management*
- [4] ISO/IEC 18033 (all parts), *Information security — Encryption algorithms*
- [5] ISO/IEC 19772:2020, *Information security — Authenticated encryption*
- [6] ISO/IEC 29167-13:2015, *Information technology — Automatic identification and data capture techniques — Part 13: Crypto suite Grain-128A security services for air interface communications*
- [7] ISO/IEC 29192-2:2019, *Information security — Lightweight cryptography — Part 2: Block ciphers*
- [8] ISO/IEC 29192-3, *Information technology — Security techniques — Lightweight cryptography — Part 3: Stream ciphers*
- [9] ISO/IEC 29192-6:2019, *Information technology — Lightweight cryptography — Part 6: Message authentication codes (MACs)*
- [10] DINUR I., SHAMIR A., “Breaking Grain-128 with dynamic cube attacks” - Fast Software Encryption, pages 167–187. Springer Berlin Heidelberg, 2011.
- [11] ÅGREN M., HELL M., JOHANSSON T., MEIER W., “Grain-128a: a new version of Grain-128 with optional authentication”- Symmetric Key Encryption Workshop 2011, DTU, Denmark, February 2011
- [12] ÅGREN M., HELL M., JOHANSSON T., MEIER W., “Grain-128a: a new version of Grain-128 with optional authentication”- International Journal of Wireless and Mobile Computing, Vol. 5, No. 1, pp. 48-59, 2011.
- [13] HELL M., JOHANSSON T., MEIER W., SÖNNERUP J., YOSHIDA H., “Grain-128AEAD - a lightweight AEAD streamcipher”- NIST Lightweight Cryptography, Round 1 Submission (2019)
- [14] WEGMAN M, CARTER L., “New hash functions and their use in authentication and set equality” - Journal of Computer and System Sciences, 22(3):265 – 279, 1981.
- [15] WANG P., ZHENG K-Y, “Security analysis of authentication mechanism in Grain-128A” - Journal of Cryptologic Research 2018, Vol. 5 Issue (1): 94-100
- [16] WANG Q., HAO Y., TODO Y., LI C., ISOBE T., MEIER W., “Improved division property based cube attacks exploiting algebraic properties of superpoly” - Advances in Cryptology – CRYPTO 2018, pages 275–305. Springer International Publishing, 2018.
- [17] TODO Y., ISOBE T., MEIER W., AOKI K., ZHANG B., “Fast correlation attack revisited” - Advances in Cryptology- CRYPTO 2018, pages 129–159. Springer International Publishing, 2018.





## Bureau of Indian Standards

BIS is a statutory institution established under the *Bureau of Indian Standards Act, 2016* to promote harmonious development of the activities of standardization, marking and quality certification of goods and attending to connected matters in the country.

### Copyright

BIS has the copyright of all its publications. No part of these publications may be reproduced in any form without the prior permission in writing of BIS. This does not preclude the free use, in the course of implementing the standard, of necessary details, such as symbols and sizes, type or grade designations. Enquiries relating to copyright be addressed to the Head (Publication & Sales), BIS.

### Review of Indian Standards

Amendments are issued to standards as the need arises on the basis of comments. Standards are also reviewed periodically; a standard along with amendments is reaffirmed when such review indicates that no changes are needed; if the review indicates that changes are needed, it is taken up for revision. Users of Indian Standards should ascertain that they are in possession of the latest amendments or edition by referring to the website-[www.bis.gov.in](http://www.bis.gov.in) or [www.standardsbis.in](http://www.standardsbis.in).

This Indian Standard has been developed from Doc No.: LITD 17 (22419).

### Amendments Issued Since Publication

Amend No.	Date of Issue	Text Affected

## BUREAU OF INDIAN STANDARDS

### Headquarters:

Manak Bhavan, 9 Bahadur Shah Zafar Marg, New Delhi 110002

Telephones: 2323 0131, 2323 3375, 2323 9402

Website: [www.bis.gov.in](http://www.bis.gov.in)

### Regional Offices:

	Telephones
Central : 601/A, Konnectus Tower -1, 6 <sup>th</sup> Floor, DMRC Building, Bhavbhuti Marg, New Delhi 110002	{ 2323 7617
Eastern : 8 <sup>th</sup> Floor, Plot No 7/7 & 7/8, CP Block, Sector V, Salt Lake, Kolkata, West Bengal 700091	{ 2367 0012 2320 9474
Northern : Plot No. 4-A, Sector 27-B, Madhya Marg, Chandigarh 160019	{ 265 9930
Southern : C.I.T. Campus, IV Cross Road, Taramani, Chennai 600113	{ 2254 1442 2254 1216
Western : Plot No. E-9, Road No.-8, MIDC, Andheri (East), Mumbai 400093	{ 2821 8093

**Branches :** AHMEDABAD. BENGALURU. BHOPAL. BHUBANESHWAR. CHANDIGARH. CHENNAI. COIMBATORE. DEHRADUN. DELHI. FARIDABAD. GHAZIABAD. GUWAHATI. HIMACHAL PRADESH. HUBLI. HYDERABAD. JAIPUR. JAMMU & KASHMIR. JAMSHEDPUR. KOCHI. KOLKATA. LUCKNOW. MADURAI. MUMBAI. NAGPUR. NOIDA. PANIPAT. PATNA. PUNE. RAIPUR. RAJKOT. SURAT. VISAKHAPATNAM.