INTERNATIONAL STANDARD

ISO/IEC 29167-13

First edition
2015-05-15

Information technology — Automatic identification and data capture techniques —

Part 13: **Crypto suite Grain-128A security services for air interface communications**

Technologies de l'information — Techniques automatiques d'identification et de capture de donnees —

Partie 13: Services de sécurité par suite cryptographique Grain-128A pour communications par interface radio

Reference number
ISO/IEC 29167-13:2015(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 31, *Automatic identification and data capture techniques*.

ISO/IEC 29167 consists of the following parts, under the general title *Information technology — Automatic identification and data capture techniques*:

— *Part 1: Security services for RFID air interfaces*

— *Part 10: Crypto suite AES-128 security services for air interface communications*

— *Part 11: Crypto suite PRESENT-80 security services for air interface communications*

— *Part 12: Crypto suite ECC-DH security services for air interface communications*

— *Part 13: Crypto suite Grain-128A security services for air interface communications*

— *Part 14: Crypto suite AES OFB security services for air interface communications*

— *Part 16: Crypto suite ECDSA-ECDH security services for air interface communications*

— *Part 17: Crypto suite cryptoGPS security services for air interface communications*

— *Part 19: Crypto suite RAMON security services for air interface communications*

The following part is under preparation:

— *Part 15: Crypto suite XOR security services for air interface communications*

Introduction

This part of ISO/IEC 29167 specifies the security services of a Grain-128A crypto suite that is based on a lightweight stream cipher. It is important to know that all security services are optional. Every manufacturer has the liberty to choose which services will be implemented on a Tag (e.g. Tag-only authentication).

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning radio-frequency identification technology given in the clauses identified below.

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

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

Impinj, Inc.
701 N 34th Street, Suite 300 **Seattle, WA 98103 USA**

The latest information on IP that might be applicable to this part of ISO/IEC 29167 can be found at www.iso.org/patents.

Information technology — Automatic identification and data capture techniques —

Part 13: **Crypto suite Grain-128A security services for air interface communications**

1 Scope

This part of ISO/IEC 29167 defines the Crypto Suite for Grain-128A for the ISO/IEC 18000 air interface standards for radio frequency identification (RFID) devices. Its purpose is to provide a common crypto suite for security for RFID devices that might be referred by ISO committees for air interface standards and application standards

This part of ISO/IEC 29167 specifies a crypto suite for Grain-128A for air interface for RFID systems. The crypto suite is defined in alignment with existing air interfaces.

This part of ISO/IEC 29167 defines various authentication methods and methods of use for the cipher. A tag and an interrogator might support one, a subset, or all of the specified options, clearly stating what is supported.

2 Conformance

2.1 Claiming conformance

To claim conformance with this part of ISO/IEC 29167, an Interrogator or Tag shall comply with all relevant clauses of this part of ISO/IEC 29167, except those marked as “optional”.

2.2 Interrogator conformance and obligations

To conform to this part of ISO/IEC 29167, an Interrogator shall

— implement the mandatory commands defined in this part of ISO/IEC 29167 and conform to the relevant part of ISO/IEC 18000.

To conform to this part of ISO/IEC 29167, an Interrogator can

— implement any subset of the optional commands defined in this part of ISO/IEC 29167.

To conform to this part of ISO/IEC 29167, the Interrogator shall not

— implement any command that conflicts with this part of ISO/IEC 29167, or

— require the use of an optional, proprietary, or custom command to meet the requirements of this part of ISO/IEC 29167.

2.3 Tag conformance and obligations

To conform to this part of ISO/IEC 29167, a Tag shall

— implement the mandatory commands defined in this part of ISO/IEC 29167 for the supported types and conform to the relevant part of ISO/IEC 18000.

To conform to this part of ISO/IEC 29167, a Tag can

— implement any subset of the optional commands defined in this part of ISO/IEC 29167.

To conform to this part of ISO/IEC 29167, a Tag shall not

— implement any command that conflicts with this part of ISO/IEC 29167, or

— require the use of an optional, proprietary, or custom command to meet the requirements of this part of ISO/IEC 29167.

3 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 19762 (all parts), *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary*

ISO/IEC 29167-1, *Information technology — Automatic identification and data capture techniques — Part 1: Security services for RFID air interfaces*

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762 (all parts) apply.

5 Symbols and abbreviated terms

5.1 Symbols

$xxxx_b$	binary notation
$xxxx_h$	hexadecimal notation
$\|\|$	concatenation of syntax elements in the order written

5.2 Abbreviated terms

CRC	Cyclic Redundancy Check
CS	Cryptographic Suite
CSI	Cryptographic Suite Indicator
IA	Interrogator Authentication
IV	Initialization Vector
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
MA	Mutual Authentication
MAC	Message Authentication Code
MSB	Most Significant Bit

NFSR	Nonlinear Feedback Shift Register
RFU	Reserved for Future Use
TA	Tag Authentication

6 Cipher introduction

Many stream ciphers have been proposed over the years, and new designs are published as cryptanalysis enhances our understanding of how to design safer and more efficient primitives. While the NESSIE [1] project failed to name a stream cipher “winner” after evaluating several new designs in 2000-2003, the eSTREAM [2] project finally decided on two portfolios of promising candidates. One of these portfolios was aimed at hardware attractive constructions, and Grain [3] is one of three finalists.

Grain is notable for its extremely small hardware representation. During the initial phase of the eSTREAM project, the original version, Grain v0, was strengthened after some observations [4]. The final version is known as Grain v1.

Like the other eSTREAM portfolio ciphers, Grain v1 is modern in the sense that it allows for public IVs, yet they only use 80-bit keys. Recognizing the emerging need for 128-bit keys, Grain-128 supporting 128-bit keys and 96-bit IVs was proposed [5]. The design is akin to that of 80-bit Grain, but noticeably, the nonlinear parts of the cipher have smaller degrees than their counterparts in Grain v1.

A new version of Grain-128, namely Grain-128A, has been specified [6]. The new stream cipher has native support for Message Authentication Code (MAC) generation and is expected to be comparable to the old version in hardware performance. MAC generation does not affect the keystream generated by Grain-128A.

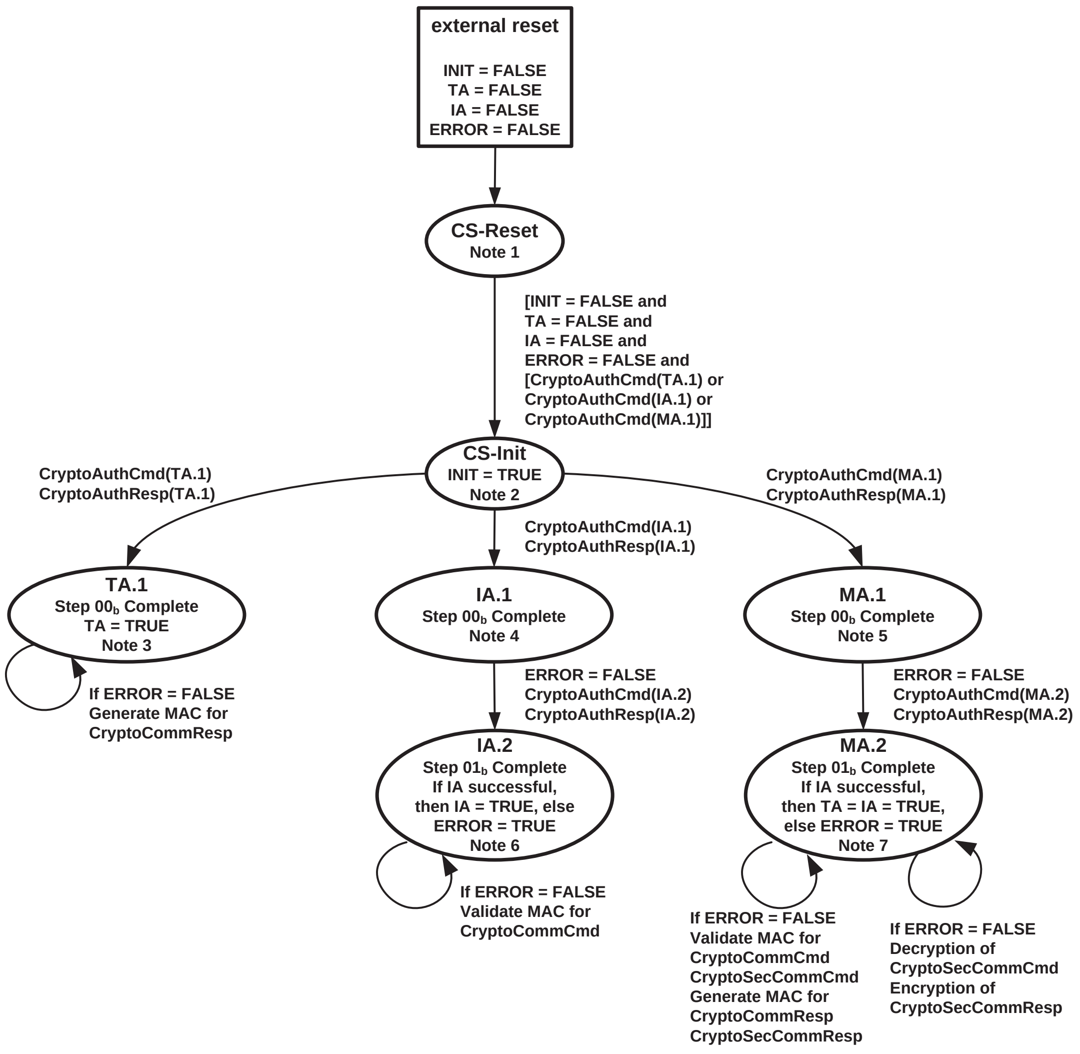
Grain-128A uses slightly different nonlinear functions in order to strengthen it against the known attacks and observations on Grain-128. The changes are modest and provide for a high confidence in Grain-128A, as the cryptanalysis carries over from Grain-128.

7 Parameter definition

Table 1 — Definition of Parameters

Parameter	Description
AuthMethod[1:0]	Authentication method specified by the Interrogator to be used by the Tag
CSFeatures[7:0]	Optional features supported by the Tag
IKeystream	Interrogator keystream used for authentication
IRandomNumber[47:0]	48-bit Interrogator random number used for crypto engine initialization
IV[95:0]	96-bit Initialization Vector
KeyID[7:0]	Specifies the 128-bit crypto key having the ID number = KeyID
MAC32[31:0]	32-bit Message Authentication Code
MAC64[63:0]	64-bit Message Authentication Code
Method[1:0]	Authentication method
Options[3:0]	Optional features specified by the Interrogator to be used by the Tag
Step[1:0]	Step number in the authentication method
TKeystream	Tag keystream used for authentication
TRandomNumber[47:0]	48-bit Tag random number used for crypto engine initialization

8 State diagram



Note 1. Any CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state.

Note 2. Initialization of the keystream generator and MAC generator shall be performed as described in Section 9.

Note 3. Any CryptoAuthCmd, CryptoSecCommCmd, or CryptoKeyUpdate in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state.

Note 4. Any CryptoAuthCmd other than IA.2, CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state.

Note 5. Any CryptoAuthCmd other than MA.2, CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state.

Note 6. Any CryptoAuthCmd, CryptoSecCommCmd, or CryptoKeyUpdate in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state. A CryptoCommCmd shall generate a MAC and authenticate the CryptoCommCmd.

Note 7. Any CryptoAuthCmd in this state shall be a Crypto Error condition (ERROR = TRUE) and cause the state machine to remain in this state. A CryptoCommCmd shall generate a MAC and authenticate the CryptoCommCmd and shall generate a MAC for use in the CryptoCommResp. A CryptoSecCommCmd shall be decrypted and generate a MAC for authenticating the CryptoSecCommCmd and the CryptoSecCommResp shall be encrypted and generate a MAC for use in the CryptoSecCommResp.

Figure 1 — Tag Crypto Engine State Diagram

The state-transition tables are provided in Annex A.

9 Initialization and resetting

The Tag's air interface protocol logic shall provide an external reset to the Tag crypto engine which shall set **INIT** = FALSE, **TA** = FALSE, **IA** = FALSE, and **ERROR** = FALSE before a transition to the **CS-Reset** state.

The **CS-Reset** state shall process crypto commands from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto command and payload for any error conditions. An error condition occurs for any CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate command. The Tag shall check a CryptoAuthCmd payload for any error conditions. An error condition in the payload occurs when Step ≠ 00_b, or the KeyID value is not supported by the Tag, or AuthMethod = 00_b and the Tag does not support Tag authentication, or AuthMethod = 00_b and the Options selected are not supported by the Tag CSFeatures, or AuthMethod = 01_b and the Tag does not support Interrogator authentication, or AuthMethod = 01_b and Options ≠ 0000_b, or AuthMethod = 10_b and Options ≠ 0000_b, or AuthMethod = 11_b and the Tag does not support a vendor defined authentication.

If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **CS-Reset** state.

If no error condition exists, the Tag shall transition to the **CS-Init** state to start processing the CryptoAuthCmd and initializes the keystream and MAC generators in the following manner. The key and the initialization vector (IV) shall be used to initialize the cipher. Denote the bits of the key as k_i, $0 \le i \le 127$ and the IV bits IV_i, $0 \le i \le 95$. The IV shall be generated using IRandomNumber and TRandomNumber such that IV[95:0] = TRandomNumber[47:0] || IRandomNumber[47:0]. The 128 NFSR elements are loaded with the key bits, $b_i = k_i$, $0 \le i \le 127$, and the first 96 LFSR elements are loaded with a one and the IV bits, $s_0 = 1$ $s_i = IV_i$, $1 \le i \le 95$. The last 32 bits of the LFSR are filled with 2 bits for authentication information followed by ones and a zero, s_{96} = Tag being authenticated, s_{97} = Interrogator being authenticated, $s_i = 1$, $98 \le i \le 126$, $s_{127} = 0$. Then, the cipher is clocked 256 times without producing any keystream. Instead the pre-output function is fed back and XORed with the input, both to the LFSR and to the NFSR. The keystream from the pre-output function is ready for use and the cipher is now clocked to initialize the MAC generator, either 64 times for a 32-bit MAC generator or 128 times for a 64-bit MAC generator. The Tag crypto engine shall set **INIT** = TRUE and the keystream and MAC generators are ready for use to support authentication and communication security services. While **INIT** = TRUE, the output streams of the keystream generator and the MAC generator shall retain state information from one crypto engine operation until the next crypto engine operation.

10 Authentication

10.1 General

A primary use for the Grain-128A CS is to perform authentication of Tags, Interrogators, or both. The authentication method to be performed shall be specified by the 2-bit value AuthMethod[1:0] which is defined in Table 2. Some of the authentication methods require multiple steps to be performed in a specific sequence. The current step in the sequence shall be specified by the 2-bit value Step[1:0] and represents steps 0, 1, 2, and 3 as defined in Table 3. All authentication methods start with step 0 and then the step increments sequentially as needed. Step 0 for all authentication methods shall be initiated by the Interrogator. During step 0 of an authentication method, the Tag shall provide an 8-bit value CSFeatures[7:0] which is defined in Table 5 and used to indicate which of the optional Grain-128A CS features are supported by the Tag. During step 0 or 1 of an authentication method, the Interrogator shall provide a 4-bit value Options[3:0] which is defined in Table 4 and used to indicate which optional features should be used by the Tag.

Table 2 — Definition of AuthMethod[1:0]

Value	Description
00_b	Tag authentication
01_b	Interrogator authentication
10_b	Mutual authentication
11_b	Vendor defined

Table 3 — Definition of Step[1:0]

Value	Description
00_b	Step 0
01_b	Step 1
10_b	Step 2
11_b	Step 3

Table 4 — Definition of Options[3:0]

Name	Description
Options[3]	Vendor defined
Options[2]	Vendor defined
Options[1]	0 = Disable Secure Authenticated Communication, 1 = Enable Secure Authenticated Communication
Options[0]	0 = Use MAC32, 1 = Use MAC64

Table 5 — Definition of CSFeatures[7:0]

Name	Description
CSFeatures[7]	Vendor defined
CSFeatures[6]	0 = Encrypted read of hidden memory not supported, 1 = Encrypted read of hidden memory supported
CSFeatures[5]	0 = Key update not supported, 1 = Key update supported
CSFeatures[4]	0 = Secure authenticated communication not supported, 1 = Secure authenticated communication supported
CSFeatures[3]	0 = MAC64 not supported, 1 = MAC64 supported
CSFeatures[2]	0 = MAC32 not supported, 1 = MAC32 supported
CSFeatures[1]	0 = IA not supported, 1 = IA supported
CSFeatures[0]	0 = TA not supported, 1 = TA supported

10.2 Tag Authentication (TA)

10.2.1 General

The Tag authentication method uses a challenge-response protocol having one pair of message exchange as shown in Figure 2. The Grain-128A CS is initialized using a crypto key specified by the Interrogator and an IV consisting of a 96-bit random number. The Interrogator and Tag each provide half of the bits used to generate the IV. Once the Grain-128A CS is initialized, the resulting keystream from the Interrogator and the Tag are compared to authenticate the Tag. The details of the Tag authentication process are provided in the following sections.

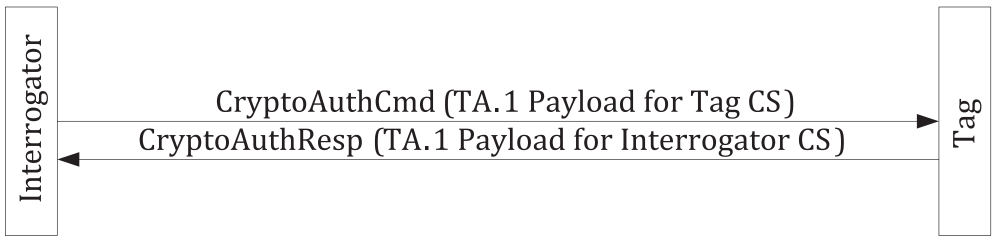

Interrogator
CryptoAuthCmd (TA.1 Payload for Tag CS)
CryptoAuthResp (TA.1 Payload for Interrogator CS)
Tag

Figure 2 — TA Message Exchange

10.2.2 CryptoAuthCmd(TA.1 Payload for Tag CS)

The Interrogator shall generate a 48-bit random number for use as IRandomNumber and save it for subsequent use. The Interrogator shall issue the challenge to the Tag with the TA.1 Payload for the Tag CS as defined in Table 6 which includes the desired options to be used, the KeyID for the crypto key to be used, and the Interrogator random number.

Table 6 — TA.1 Payload for Tag CS

	AuthMethod	Step	Options	KeyID	IRandomNumber
# of bits	2	2	4	8	48
description	00_b	00_b	As desired	nn_h	Interrogator random number

10.2.3 CryptoAuthResp(TA.1 Payload for Interrogator CS)

The Tag shall generate a 48-bit random number for use as TRandomNumber. The Tag crypto engine shall be initialized for Tag authentication as specified in Clause 9 using TRandomNumber, IRandomNumber, and the crypto key specified by KeyID. The crypto engine shall then generate the Tag keystream TKeystream[63:0]. The Tag shall respond to the challenge from the Interrogator with the TA.1 Payload for the Interrogator CS as defined in Table 7 which includes the CS features of the Tag, the Tag random number, and the Tag keystream. The Tag shall transition to the **TA.1** state after the response to the Interrogator and shall set **TA** = TRUE. Tag authentication Step ‘00_b’ is now complete.

Table 7 — TA.1 Payload for Interrogator CS

	CSFeatures	TRandomNumber	TKeystream
# of bits	8	48	64
description	CS features of the Tag	Tag random number	Tag keystream

10.2.4 Final Interrogator Processing

The Interrogator crypto engine shall be initialized for Tag authentication as specified in Clause 9 using TRandomNumber, the saved IRandomNumber, and the crypto key specified by KeyID. The crypto engine shall then generate the Interrogator keystream IKeystream[63:0]. The Interrogator shall use

IKeystream and TKeystream to authenticate the Tag and accepts the Tag as valid if TKeystream[63:0] = IKeystream[63:0].

10.3 Interrogator Authentication (IA)

10.3.1 General

The Interrogator authentication method uses a challenge-response protocol having two pairs of message exchange as shown in Figure 3. The Grain-128A CS is initialized using a crypto key specified by the Interrogator and an IV consisting of a 96-bit random number. The Interrogator and Tag each provide half of the bits used to generate the IV. Once the Grain-128A CS is initialized, the resulting keystream from the Interrogator and the Tag are compared to authenticate the Interrogator. The details of the Interrogator authentication process are provided in the following sections.

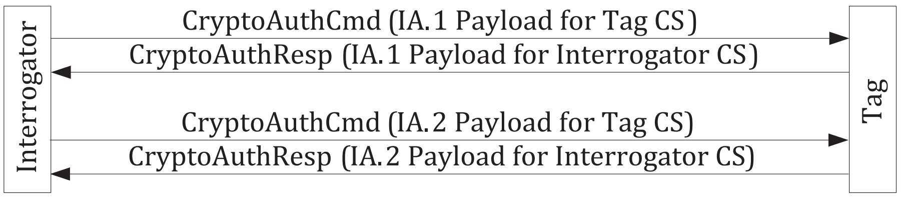

Interrogator
Tag
CryptoAuthCmd (IA.1 Payload for Tag CS)
CryptoAuthResp (IA.1 Payload for Interrogator CS)
CryptoAuthCmd (IA.2 Payload for Tag CS)
CryptoAuthResp (IA.2 Payload for Interrogator CS)

Figure 3 — IA Message Exchange

10.3.2 CryptoAuthCmd(IA.1 Payload for Tag CS)

The Interrogator shall generate a 48-bit random number for use as IRandomNumber and save it for subsequent use. The Interrogator shall request a challenge from the Tag using the IA.1 Payload for the Tag CS as defined in Table 8 which includes the KeyID for the crypto key to be used and the Interrogator random number.

Table 8 — IA.1 Payload for Tag CS

	AuthMethod	Step	Options	KeyID	IRandomNumber
# of bits	2	2	4	8	48
description	01_b	00_b	0000_b	nn_h	Interrogator random number

10.3.3 CryptoAuthResp(IA.1 Payload for Interrogator CS)

The Tag shall generate a 48-bit random number for use as TRandomNumber in the following IA.1 Payload for the Interrogator CS. The Tag crypto engine shall be initialized for Interrogator authentication as specified in Clause 9 using TRandomNumber, IRandomNumber, and the crypto key specified by KeyID. The Tag shall respond with the challenge to the Interrogator with the IA.1 Payload for the Interrogator CS as defined in Table 9 which includes the CS features of the Tag and the Tag random number. The Tag shall transition to the **IA.1** state after the response to the Interrogator. Interrogator authentication Step '00_b' is now complete.

Table 9 — IA.1 Payload for Interrogator CS

	CSFeatures	TRandomNumber
# of bits	8	48
description	CS features of the Tag	Tag random number

10.3.4 CryptoAuthCmd(IA.2 Payload for Tag CS)

The Interrogator crypto engine shall be initialized for Interrogator authentication as specified in Clause 9 using TRandomNumber, the saved IRandomNumber, and the crypto key specified by KeyID. The crypto engine shall then generate the Interrogator keystream IKeystream[63:0]. The Interrogator shall then respond to the challenge from the Tag with the IA.2 Payload for the Tag CS as defined in Table 10 which includes the desired options to be used, the KeyID for the crypto key to be used, and the Interrogator keystream. The KeyID value shall be the same as used in the IA.1 Payload for the Tag CS.

Table 10 — IA.2 Payload for Tag CS

	AuthMethod	Step	Options	KeyID	IKeystream
# of bits	2	2	4	8	64
description	01_b	01_b	As desired	Same as used for IA.1 Payload	Interrogator keystream

10.3.5 CryptoAuthResp(IA.2 Payload for Interrogator CS)

The **IA.1** state shall process crypto commands from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto command and payload for any error conditions. An error condition occurs for any CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate command. The Tag shall check a CryptoAuthCmd payload for any error conditions. An error condition in the payload occurs when AuthMethod ≠ 01_b, Step ≠ 01_b, or the KeyID value is not the same as used for the IA.1 payload, or the Options selected are not supported by the Tag CSFeatures.

If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **IA.1** state.

If no error condition exists, the Tag crypto engine shall generate the Tag keystream TKeystream[63:0]. The Tag shall then use IKeystream and TKeystream to authenticate the Interrogator and accepts the Interrogator as valid if TKeystream[63:0] = IKeystream[63:0]. The Tag shall respond to the Interrogator with the IA.2 Payload for the Interrogator CS as defined in Table 11 which includes the status information whether the Interrogator authentication succeeded or failed. If the Interrogator authentication succeeded, the Tag shall transition to the **IA.2** state after the response to the Interrogator and set **IA** = TRUE. Otherwise, the Interrogator authentication failed and the Tag shall transition to the **IA.2** state after the response to the Interrogator and set **ERROR** = TRUE. Interrogator authentication Step '01_b' is now complete.

Table 11 — IA.2 Payload for Interrogator CS

	IA Status
# of bits	1
description	0 = OK (succeeded), 1 = KO (failed)

10.4 Mutual Authentication (MA)

10.4.1 General

The mutual authentication method uses a challenge-response protocol having two pairs of message exchange as shown in Figure 4 and is based on Interrogator authentication first. The Grain-128A CS is initialized using a crypto key specified by the Interrogator and an IV consisting of a 96-bit random number. The Interrogator and Tag each provide half of the bits used to generate the IV. Once the Grain-128A CS is initialized, the resulting keystream from the Interrogator and the Tag are compared to authenticate the Tag and the Interrogator. The details of the mutual authentication process are provided in the following sections.

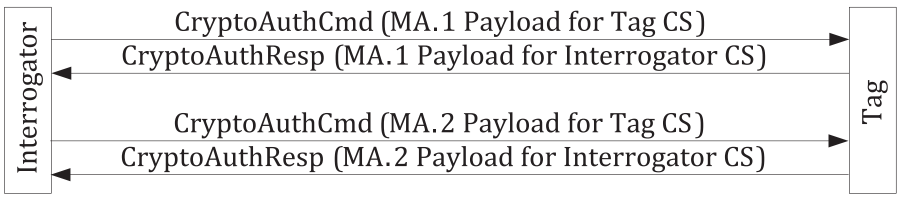

Interrogator
Tag
CryptoAuthCmd (MA.1 Payload for Tag CS)
CryptoAuthResp (MA.1 Payload for Interrogator CS)
CryptoAuthCmd (MA.2 Payload for Tag CS)
CryptoAuthResp (MA.2 Payload for Interrogator CS)

Figure 4 — MA Message Exchange

10.4.2 CryptoAuthCmd (MA.1 Payload for Tag CS)

The Interrogator shall generate a 48-bit random number for use as IRandomNumber and save it for subsequent use. The Interrogator shall request a challenge from the Tag using the MA.1 Payload for the Tag CS as defined in Table 12 which includes the KeyID for the crypto key to be used and the Interrogator random number.

Table 12 — MA.1 Payload for Tag CS

	AuthMethod	Step	Options	KeyID	IRandomNumber
# of bits	2	2	4	8	48
description	10_b	00_b	0000_b	nn_h	Interrogator random number

10.4.3 CryptoAuthResp(MA.1 Payload for Interrogator CS)

The Tag shall generate a 48-bit random number for use as TRandomNumber in the following MA.1 Payload for the Interrogator CS. The Tag crypto engine shall be initialized for mutual authentication as specified in Clause 9 using TRandomNumber, IRandomNumber, and the crypto key specified by KeyID. The Tag shall respond with the challenge to the Interrogator with the MA.1 Payload for the Interrogator CS as defined in Table 13 which includes the CS features of the Tag and the Tag random number. The Tag shall transition to the **MA.1** state after the response to the Interrogator. Mutual authentication Step '00_b' is now complete.

Table 13 — MA.1 Payload for Interrogator CS

	CSFeatures	TRandomNumber
# of bits	8	48
description	CS features of the Tag	Tag random number

10.4.4 CryptoAuthCmd(MA.2 Payload for Tag CS)

The Interrogator crypto engine shall be initialized for mutual authentication as specified in Clause 9 using TRandomNumber, the saved IRandomNumber, and the crypto key specified by KeyID. The crypto engine shall then generate the Interrogator keystream IKeystream[63:0]. The Interrogator shall then respond to the challenge from the Tag with the MA.2 Payload for the Tag CS as defined in Table 14 which includes the desired options to be used, the KeyID for the crypto key to be used, and the Interrogator keystream. The KeyID value shall be the same as used in the MA.1 Payload for the Tag CS

Table 14 — MA.2 Payload for Tag CS

	AuthMethod	Step	Options	KeyID	IKeystream
# of bits	2	2	4	8	64
description	10_b	01_b	As desired	Same as used for MA.1 Payload	Interrogator keystream

10.4.5 CryptoAuthResp(MA.2 Payload for Interrogator CS)

The **MA.1** state shall process crypto commands from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto command and payload for any error conditions. An error condition occurs for any CryptoCommCmd, CryptoSecCommCmd, or CryptoKeyUpdate command. The Tag shall check a CryptoAuthCmd payload for any error conditions. An error condition in the payload occurs when AuthMethod ≠ 10_b, Step ≠ 01_b, or the KeyID value is not the same as used for the MA.1 payload, or the Options selected are not supported by the Tag CSFeatures.

If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **MA.1** state.

If no error condition exists, the Tag crypto engine shall generate the Tag keystream TKeystream[63:0]. The Tag shall then use IKeystream and TKeystream to authenticate the Interrogator and accepts the Interrogator as valid if TKeystream[63:0] = IKeystream[63:0].

If the Interrogator is invalid, the Tag shall respond to the Interrogator with the MA.2 Payload for the Interrogator CS as defined in Table 15 with the status information that the Interrogator authentication failed and not include a Tag keystream. The Tag shall transition to the **MA.2** state after the response to the Interrogator and set **ERROR** = TRUE. Mutual authentication Step '01_b' is now complete.

If the Interrogator is valid, the Tag crypto engine shall generate a new value for the Tag keystream TKeystream[127:64]. The Tag shall respond to the Interrogator with the MA.2 Payload for the Interrogator CS as defined in Table 15 with the status information that the Interrogator authentication succeeded and include the updated Tag keystream for Tag authentication by the Interrogator. The Tag shall transition to the **MA.2** state after the response to the Interrogator and set **TA** = **IA** = TRUE. Mutual authentication Step '01_b' is now complete.

Table 15 — MA.2 Payload for Interrogator CS

	IA Status	TKeystream
# of bits	1	0 when IA failed, 64 when IA succeeded
description	0 = OK (succeeded), 1 = KO (failed)	Tag keystream

10.4.6 Final Interrogator Processing

The Interrogator shall check the authentication status from the Tag and if it is OK, the Interrogator crypto engine shall generate a new value for the Interrogator keystream IKeystream[127:64]. The Interrogator shall use TKeystream and the updated IKeystream to authenticate the Tag and accepts the Tag as valid if TKeystream[127:64] = IKeystream[127:64].

11 Communication

11.1 General

Authentication integrity shall be maintained for an Interrogator authentication and a Mutual authentication. It is optional to maintain the authentication integrity of a Tag authentication.

Authentication integrity shall be performed using authenticated communication and/or secure authenticated communication. A Message Authentication Code (MAC) shall be used to provide the integrity protection. The Interrogator selects between using a MAC32 or a MAC64 via the Options parameter during the Tag authentication process in 10.2.2, the Interrogator authentication process in 10.3.4, or the mutual authentication process in 10.4.4.

11.2 Authenticated Communication

Authenticated communication is used for an air interface protocol command and/or response and includes a Message Authentication Code to maintain the integrity of a prior authentication. If a Tag is authenticated as a result of Tag authentication, then it is at the discretion of the Interrogator to maintain the integrity of the authentication during subsequent communications with the singulated Tag. The Interrogator may use authenticated communication but the Interrogator commands to the Tag cannot provide integrity protection since the Interrogator has not been authenticated. The **TA.1** state shall process crypto responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto responses for error conditions. An error condition occurs for any CryptoAuthResp or CryptoSecCommResp. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **TA.1** state. If no error condition exists, the Tag shall provide integrity protection for the Tag response in the CryptoCommResp payload. Integrity of the Tag response is shown in Table 17 and shall be performed with the addition of an 8-bit value of 00_h followed by a MAC generated by the Tag crypto engine MAC generator. The Tag shall remain in the **TA.1** state after the response is sent to the Interrogator. The Interrogator crypto engine MAC generator shall generate a MAC for the Tag response within the CryptoCommResp payload to authenticate the Tag response. The Interrogator accepts the Tag response as valid from the authenticated Tag if the MAC from the Tag equals the MAC from the Interrogator.

If an Interrogator is authenticated as a result of Interrogator authentication, then it shall maintain the integrity of the authentication during subsequent communications with the singulated Tag. Tag replies to the authenticated Interrogator cannot provide integrity protection since the Tag has not been authenticated. Integrity of Interrogator commands is shown in Table 16 and shall be performed by the addition of an 8-bit value of 00_h followed by a MAC generated by the Interrogator crypto engine MAC generator. The **IA.2** state shall process crypto commands and responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto commands for error conditions. An error condition occurs for any CryptoAuthCmd, CryptoSecCommCmd, or CryptoKeyUpdate. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **IA.2** state. If no error condition exists, the Tag crypto engine MAC generator shall generate a MAC for the Interrogator command within the CryptoCommCmd payload to authenticate the Interrogator command. The Tag accepts the Interrogator command as valid from the authenticated Interrogator if the MAC from the Interrogator equals the MAC from the Tag. If they are not equal then the Interrogator command is invalid and the Tag crypto engine shall set **ERROR** = TRUE and the Tag shall remain in the **IA.2** state.

If a Tag and Interrogator are both authenticated as a result of mutual authentication, then both shall maintain the integrity of the authentication during subsequent communications with the singulated Tag. Additionally, an Interrogator has the option to enable the use of encrypypted commands and responses when secure authenticated communication is supported by the Tag. This feature is enabled via the Options parameter during the mutual authentication process in 10.4.4. Secure authenticated communication is described in 11.3. Integrity of Interrogator commands is shown in Table 16 and shall be performed by the addition of an 8-bit value of 00_h followed by a MAC generated by the Interrogator crypto engine MAC generator. The **MA.2** state shall process crypto commands and responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto commands for error conditions. If secure authenticated communication is not enabled then an error condition occurs for any CryptoAuthCmd, CryptoSecCommCmd, or CryptoKeyUpdate. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **MA.2** state. If no error condition exists, the Tag crypto engine MAC generator shall generate a MAC for the Interrogator command within the CryptoCommCmd payload to authenticate the Interrogator command. The Tag accepts the Interrogator command as valid from the authenticated Interrogator if the MAC from the Interrogator equals the MAC from the Tag. If they are not equal then the Interrogator command is invalid and the Tag crypto engine shall set **ERROR** = TRUE and the Tag shall remain in the **MA.2** state. The **MA.2** state shall also process crypto responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto responses for error conditions. An error condition occurs for any CryptoAuthResp or CryptoSecCommResp. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **MA.2** state. If no error condition exists, the Tag shall provide integrity protection for the Tag response in the CryptoCommResp payload. Integrity of the Tag response is shown in Table 17 and shall be performed with the addition of an 8-bit value of 00_h followed by a MAC generated by the Tag crypto engine MAC generator. The Tag shall remain in the **MA.2** state after the response is sent to the Interrogator. The Interrogator crypto engine MAC generator shall generate a MAC for the Tag response within the CryptoCommResp payload to authenticate the Tag response. The Interrogator accepts the Tag response as valid from the authenticated Tag if the MAC from the Tag equals the MAC from the Interrogator.

Table 16 — Authenticated Communication for Tag CS

CryptoCommCmd(Payload)		
Interrogator command	00_h	MAC

NOTE The transmission of 00h is used for timing transitions between data and MAC.

Table 17 — Authenticated Communication for Interrogator CS

CryptoCommResp(Payload)		
Tag response	00_h	MAC

NOTE The transmission of 00h is used for timing transitions between data and MAC.

11.3 Secure Authenticated Communication

Secure authenticated communication is used for an air interface protocol command and/or response that is encrypted and includes a Message Authentication Code to maintain the integrity of a prior authentication. If a Tag and Interrogator are both authenticated as a result of mutual authentication, then both shall maintain the integrity of the authentication during subsequent communications with the singulated Tag. Additionally, an Interrogator has the option to enable the use of encrypyted commands and responses when secure authenticated communication is supported by the Tag. This feature is enabled via the Options parameter during the mutual authentication process in 10.4.4. The Interrogator shall use the crypto engine keystream generator to encrypt the Interrogator command. Integrity of the encrypted command is shown in Table 18 and shall be performed by the addition of an 8-bit value of 00_h followed by a MAC generated by the Interrogator crypto engine MAC generator. The **MA.2** state shall process crypto commands and responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto commands for error conditions. An error condition occurs for any CryptoAuthCmd or for any CryptoSecCommCmd when secure authenticated communication is not enabled. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **MA.2** state. If no error condition exists, the Tag shall process a CryptoCommCmd as defined in 11.2 and shall process a CryptoSecCommCmd as follows. The Tag shall use the crypto engine keystream generator to decrypt the Interrogator command within the CryptoSecCommCmd. The TAG crypto engine MAC generator shall generate a MAC for the Interrogator command within the CryptoSecCommCmd payload to authenticate the Interrogator command. The Tag accepts the Interrogator command as valid from the authenticated Interrogator if the MAC from the Interrogator equals the MAC from the Tag. If they are not equal then the Interrogator command is invalid and the Tag crypto engine shall set **ERROR** = TRUE and the Tag shall remain in the **MA.2** state. The **MA.2** state shall also process crypto responses from the Tag's air interface protocol logic only when **ERROR** = FALSE. The Tag shall check the crypto responses for error conditions. An error condition occurs for any CryptoAuthResp or for any CryptoSecCommResp when secure authenticated communication is not enabled. If an error condition exists then the Tag crypto engine shall set **ERROR** = TRUE and remain in the **MA.2** state. If no error condition exists, the Tag shall use the crypto engine keystream generator to encrypt the Tag response. Integrity of the encrypted response is shown in Table 19 and shall be performed by the addition of an 8-bit value of 00_h followed by a MAC generated by the Tag crypto engine MAC generator. The Tag shall remain in the **MA.2** state after the response is sent to the Interrogator. The Interrogator shall use the crypto engine keystream generator to decrypt the Tag response within the CryptoSecCommResp. The Interrogator crypto engine MAC generator shall generate a MAC for the Tag response within the CryptoSecCommResp payload to authenticate the Tag response. The Interrogator accepts the Tag response as valid from the authenticated Tag if the MAC from the Tag equals the MAC from the Interrogator.

Table 18 — Authenticated Secure Communication for Tag CS

CryptoSecCommCmd(Payload)		
Encrypted Interrogator command	00_h	MAC

NOTE The transmission of 00h is used for timing transitions between data and MAC.

Table 19 — Authenticated Secure Communication for Interrogator CS

CryptoSecCommResp(Payload)		
Encrypted Tag response	00_h	MAC

NOTE The transmission of 00h is used for timing transitions between data and MAC.

12 Key table and key update

Tags may implement an optional key table for storage of the crypto keys used for this crypto suite. If implemented, it is recommended that Tags permit an Interrogator to perform a key update in the key table using secure authenticated communication. The Interrogator shall provide the crypto key value as defined in Table 20.

Table 20 — CryptoKeyUpdate(Payload)

	KeyID	**Crypto Key**
# of bits	8	128
description	nn_h	128-bit key value

Annex A
(normative)

State transition tables

The Tag crypto engine state diagram is shown in Clause 8. State-transition Tables A.1 to A.7 define the Tag's crypto engine response to crypto commands from the Tag's air interface protocol logic.

A.1 Present state: CS-Reset

See Clause 9 for additional information.

Command	Condition	Action	Next State
all	**ERROR** = TRUE	--	CS-Reset
CrytptoAuthCmd(any Payload)	Step ≠ 00_b	Set **ERROR** = TRUE	CS-Reset
	KeyID value is not supported by the Tag	Set **ERROR** = TRUE	CS-Reset
CrytptoAuthCmd(TA.1 Payload)	Tag does not support Tag authentication	Set **ERROR** = TRUE	CS-Reset
	Options selected are not supported by the Tag CSFeatures	Set **ERROR** = TRUE	CS-Reset
	Otherwise	--	CS-Init
CrytptoAuthCmd(IA.1 Payload)	Tag does not support Interrogator authentication	Set **ERROR** = TRUE	CS-Reset
	Options ≠ 00_b	Set **ERROR** = TRUE	CS-Reset
	Otherwise	--	CS-Init
CrytptoAuthCmd(MA.1 Payload)	Options ≠ 00_b	Set **ERROR** = TRUE	CS-Reset
	Otherwise	--	CS-Init
CryptoCommCmd(Payload)	all	Set **ERROR** = TRUE	CS-Reset
CryptoSecCommCmd(Payload)	all	Set **ERROR** = TRUE	CS-Reset
CryptoKeyUpdateCmd(Payload)	all	Set **ERROR** = TRUE	CS-Reset

A.2 Present state: CS-Init

See Clause 9 for additional information.

Command	Condition	Action	Next State

CrytptoAuthCmd(TA.1 Payload)	all	Generate TRandomNumber;
		Initialize crypto engine for
		keystream generation and MAC generation;
		Set **INIT** = TRUE;
		Generate TKeystream;
		Generate CryptoAuthResp(TA.1 Payload);
		Set **TA** = TRUE;
			TA.1

CrytptoAuthCmd(IA.1 Payload)	all	Generate TRandomNumber;
		Initialize crypto engine for
		keystream generation and MAC generation;
		Set **INIT** = TRUE;
		Generate TKeystream;
		Generate CryptoAuthResp(TA.1 Payload)
			IA.1

CrytptoAuthCmd(MA.1 Payload)	all	Generate TRandomNumber;
		Initialize crypto engine for
		keystream generation and MAC generation;
		Set **INIT** = TRUE;
		Generate TKeystream;
		Generate CryptoAuthResp(TA.1 Payload)
			MA.1

A.3 Present state: TA.1

See 11.2 for additional information.

Response	Condition	Action	Next State
all	**ERROR** = TRUE	--	TA.1
CrytptoAuthResp(any Payload)	all	Set **ERROR** = TRUE	TA.1
CryptoCommResp(Payload)	**ERROR** = FALSE	Generate MAC for integrity protection of the Payload	TA.1
CryptoSecCommResp(Payload)	all	Set **ERROR** = TRUE	TA.1

A.4 Present state: IA.1

See 10.3.5 for additional information.

Command	Condition	Action	Next State
all	**ERROR** = TRUE	--	IA.1
CrytptoAuthCmd(any Payload)	AuthMethod ≠ 01_b	Set **ERROR** = TRUE	IA.1
	Step ≠ 01_b	Set **ERROR** = TRUE	IA.1
CryptoAuthCmd(IA.2 Payload)	**ERROR** = FALSE and KeyID value is not the same as used for CryptoAuthCmd(IA.1 Payload)	Set **ERROR** = TRUE	IA.1
	ERROR = FALSE and Options selected are not supported by the Tag CSFeatures	Set **ERROR** = TRUE	IA.1
	ERROR = FALSE and otherwise	Generate TKeystream; Authenticate Interrogator; If valid, set **IA** = TRUE; If invalid, Set **ERROR** = TRUE; Generate CryptoAuthResp(IA.2 Payload)	IA.2
CryptoCommCmd(Payload)	all	Set **ERROR** = TRUE	IA.1
CryptoSecCommCmd(Payload)	all	Set **ERROR** = TRUE	IA.1
CryptoKeyUpdateCmd(Payload)	all	Set **ERROR** = TRUE	IA.1

A.5 Present state: IA.2

See 11.2 for additional information.

Command	Condition	Action	Next State
all	**ERROR** = TRUE	--	IA.2
CrytptoAuthCmd(any Payload)	all	Set **ERROR** = TRUE	IA.2

CryptoCommCmd(Payload)	**ERROR** = FALSE	Generate MAC;
		Authenticate Payload;
		If invalid, set **ERROR** = TRUE
			IA.2

CryptoSecCommCmd(Payload)	all	Set **ERROR** = TRUE	IA.2
CryptoKeyUpdateCmd(Payload)	all	Set **ERROR** = TRUE	IA.2

A.6 Present state: MA.1

See 10.4.5 for additional information.

Command	Condition	Action	Next State
all	**ERROR** = TRUE	--	MA.1
CrytptoAuthCmd(any Payload)	AuthMethod ≠ 10_b	Set **ERROR** = TRUE	MA.1
	Step ≠ 01_b	Set **ERROR** = TRUE	MA.1

Command	Condition	Action	Next State
CryptoAuthCmd(MA.2 Payload)	**ERROR** = FALSE and KeyID value is not the same as used for CryptoAuthCmd(MA.1 Payload)	Set **ERROR** = TRUE	MA.1
	ERROR = FALSE and Options selected are not supported by the Tag CSFeatures	Set **ERROR** = TRUE	MA.2
	ERROR = FALSE and otherwise	Generate TKeystream; Authenticate Interrogator; If valid, set **TA** = **IA** = TRUE and generate TKeystream; If invalid, Set **ERROR** = TRUE; Generate CryptoAuthResp(MA.2 Payload)	IA.2
CryptoCommCmd(Payload)	all	Set **ERROR** = TRUE	MA.1
CryptoSecCommCmd(Payload)	all	Set **ERROR** = TRUE	MA.1
CryptoKeyUpdateCmd(Payload)	all	Set **ERROR** = TRUE	MA.1

A.7 Present state: MA.2

See 11.2 and 11.3 for additional information.

Command/Response	Condition	Action	Next State
all	**ERROR** = TRUE	--	MA.2
CrytptoAuthCmd(any Payload)	all	Set **ERROR** = TRUE	MA.2
CrytptoAuthResp(any Payload)	all	Set **ERROR** = TRUE	MA.2

CryptoCommCmd(Payload)	**ERROR** = FALSE	Generate MAC;
		Authenticate Payload;
		If invalid, set **ERROR** = TRUE
			MA.2

CryptoCommResp(Payload)	**ERROR** = FALSE	Generate MAC for integrity protection of the Payload	MA.2
CryptoSecCommCmd(Payload)	Secure authenticated communication disabled	Set **ERROR** = TRUE	MA.2

CryptoSecCommCmd(Payload)	**ERROR** = FALSE and secure authenticated communication enabled	Decrypt command in Payload;
		Generate MAC;
		Authenticate Payload;
		If invalid, set **ERROR** = TRUE
			MA.2

CryptoSecCommResp(Payload)	Secure authenticated communication disabled	Set **ERROR** = TRUE	MA.2
CryptoSecCommResp(Payload)	**ERROR** = FALSE and secure authenticated communication enabled	Encrypt response in Payload; Generate MAC for integrity protection of the Payload	MA.2
CryptoKeyUpdateCmd(Payload)	KeyID value is not supported by the Tag	Set **ERROR** = TRUE	MA.2
	ERROR = FALSE and otherwise	Update crypto key	MA.2

Annex B
(normative)

Error conditions and error handling

This annex contains a listing of the error conditions that may result during the operation of this crypto suite. The Tag crypto engine shall report an error condition to the Tag air interface protocol logic by setting **ERROR** = TRUE. The Tag crypto engine shall maintain the error condition until the Tag air interface protocol logic resets the Tag crypto engine which shall set **ERROR** = FALSE.

There are three types of error conditions reported by the Tag crypto engine assuming the Tag air interface protocol logic is in sync with the Tag crypto engine state machine:

— Type 1. A non-catastrophic error that may be reported by the Tag air interface protocol logic to the Interrogator. This error type results from errors in the Payload of a CryptoAuthCmd other than an Interrogator authentication failure.

— Type 2. A catastrophic error that may be reported by the Tag air interface protocol logic to the Interrogator. This error type results from errors in the Payload of a CryptoAuthCmd that are an Interrogator authentication failure. The Interrogator Authentication status is included in the Payload of the CryptoAuthResp.

— Type 3. A catastrophic error that may not be reported by the Tag air interface protocol logic to the Interrogator. This error type results from an error in the Payload of a CryptoCommCmd or CryptoSecCommCmd that are an Interrogator command authentication failure.

State	Condition	Error Type
CS-Reset	**ERROR** = TRUE and **TA** = FALSE and **IA** = FALSE	Type 1
TA.1	**ERROR** = TRUE and **TA** = TRUE and **IA** = FALSE	Type 3
IA.1	**ERROR** = TRUE and **TA** = FALSE and **IA** = FALSE	Type 3
IA.2	**ERROR** = TRUE and **TA** = FALSE and **IA** = FALSE	Type 2
IA.2	**ERROR** = TRUE and **TA** = FALSE and **IA** = TRUE	Type 3
MA.1	**ERROR** = TRUE and **TA** = FALSE and **IA** = FALSE	Type 3
MA.2	**ERROR** = TRUE and **TA** = FALSE and **IA** = FALSE	Type 2
MA.2	**ERROR** = TRUE and **TA** = TRUE and **IA** = TRUE	Type 3

Annex E defines the air interface protocol specific error reporting.

Annex C
(normative)

Cipher description

Grain-128A consists of a mechanism that produces a pre-output stream and MAC generation. Figure C.1 depicts an overview of the building blocks of the pre-output generator, which is constructed using three main building blocks, namely a 128-bit LFSR, a 128-bit NFSR and a pre-output function. The contents of the LFSR are denoted MSB to LSB by $s_i, s_{i+1}, \ldots, s_{i+127}$. Similarly, the content of the NFSR is denoted MSB to LSB by $b_i, b_{i+1}, \ldots, b_{i+127}$. Together, the 256 memory elements in the two shift registers represent the state of the pre-output generator.

The primitive feedback polynomial of the LFSR, denoted $f(x)$, is defined as

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

To remove any possible ambiguity we also give the corresponding update function of the LFSR as

$$s_{i+28} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

The nonlinear feedback polynomial of the NFSR, $g(x)$, is defined as

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

To once more remove any possible ambiguity we also give the rule for updating the NFSR.

$$b_{i+28} = s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18}$$
$$+ b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + b_{i+68}b_{i+84} + b_{i+88}b_{i+92}b_{i+93}b_{i+95} + b_{i+22}b_{i+24}b_{i+25} + b_{i+70}b_{i+78}b_{i+82}.$$

Note that the update rule contains the bit s_i which is output from the LFSR and masks the input to the NFSR, while it was left out in the feedback polynomial.

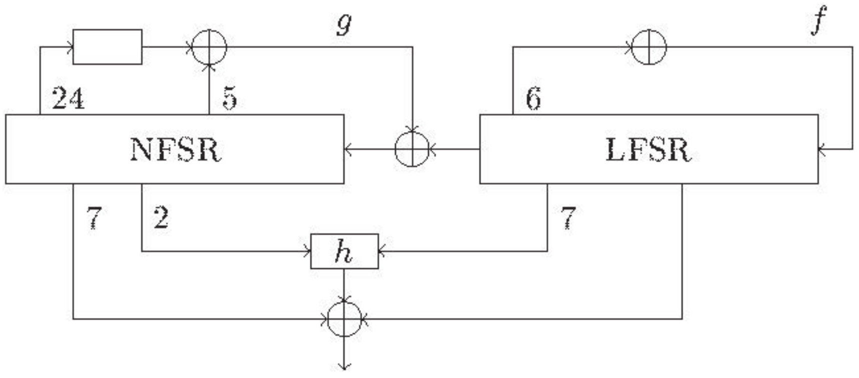


Figure C.1 — An Overview of the Pre-output Generator

Nine state variables are taken as input to a Boolean function, $h(x)$: two bits come from the NFSR and seven from the LFSR. This function is defined as

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$$

where the variables $x_0,\dots,x_8$ correspond to, respectively, the state variables b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} and s_{i+94}. The pre-output function is defined as

$$y_i = h(x) + s_{i93} + \sum_{j\in A} b_{i+j}$$

where A = {2,15,36,45,64,73,89}.

An important feature of Grain-128A is that the speed can be increased at the expense of more hardware. This requires the small feedback functions, $f(x)$ and $g(x)$, and the pre-output function to be implemented several times. To aid this, the last 31 bits of the shift registers, s_i, b_i, $97 \le i \le 127$ are not used in the respective feedback function or in the input to the pre-output function. This allows the speed to be easily multiplied by up to 32 if a sufficient amount of hardware is available.

An overview of the implementation when the speed is doubled can be seen in Figure C.2. The shift registers also need to be implemented such that each bit is shifted t steps instead of just one when the speed is increased by a factor t. The possibilities to increase the speed is limited to powers of two as t needs to divide e.g., the initialization count, which is 256, and the authentication initialization, which is another 64 basic clockings for MAC32 or 128 basic clockings for MAC64. Since the pre-output and feedback functions are small, it is quite feasible to increase the throughput in this way. By increasing the speed by a factor 32, the cipher will output 32 bits/clock, or 16 bits/clock when using authentication.

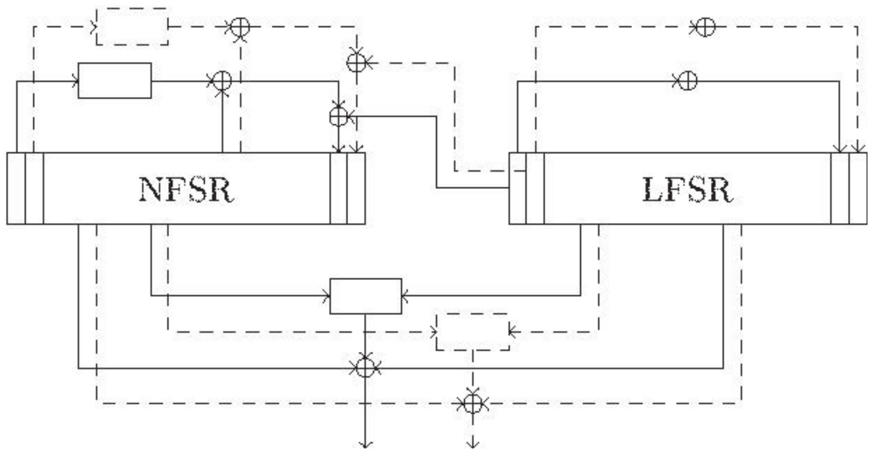


Figure C.2 — 2X Implementation of Pre-output Generator

Figure C.3 depicts an overview of the building blocks of the MAC generator, which is constructed using two main building blocks, namely an accumulator and a shift register. The size of each building block is 32 bits for a MAC32 and 64 bits for a MAC64.

Assume that we have a message of length L defined MSB to LSB by the bits $m_0,\dots,m_{L-1}$. Set $m_L = 1$. Note that $m_L = 1$ is the padding, which is crucial for the security of the message authentication as it ensures that **m** and **m** || 0 have different MAC's.

The content of the accumulator at time i is denoted MSB to LSB by $a_i^0, \dots, a_i^{31}$ for MAC32 and $a_i^0, \dots, a_i^{63}$ for MAC64. The content of the shift register is denoted MSB to LSB by $r_i, \dots, r_{i+31}$ for MAC32 and $r_i, \dots, r_{i+63}$ for MAC64. The accumulator is initialized through $a_0^j = y_{256+j}$, $0 \le j \le 31$ for MAC32 and $a_0^j = y_{256+j}$, $0 \le j \le 63$ for MAC64. The shift register is initialized through $r_i = y_{288+i}$, $0 \le i \le 31$ for MAC32 and $r_i = y_{320+i}$, $0 \le i \le 63$ for MAC64. The shift register is updated as $r_{i+32} = y_{320+2i+1}$ for MAC32 and $r_{i+64} = y_{384+2i+1}$ for MAC64. The accumulator is updated as $a_{i+1}^j = a_i^j + m_i r_{i+j}$ for $0 \le i \le L$ and $0 \le j \le 31$ for MAC32 and $0 \le j \le 63$ for MAC64.

The final content of the accumulator is denoted the MAC and can be used for authentication integrity for communication.

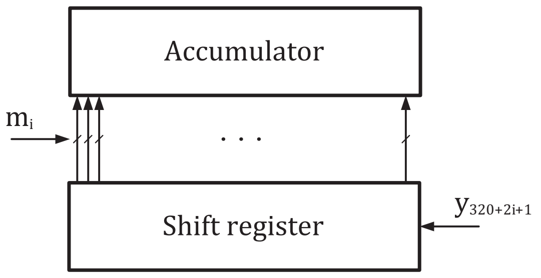


Figure C.3 — MAC32 Generator

The cipher is ready for use as a keystream generator once the cipher initialization has been completed. The output of the keystream generator at time i is denoted ks_i where $ks_i = y_{320+2i}$ when using MAC32 and $ks_i = y_{384+2i}$ when using MAC64. All the shift registers used by the cipher are regularly clocked so the cipher will output one bit every second clock when using authentication. This regular clocking is an advantage, both in terms of performance and resistance to side-channel attacks, compared to using irregular clocking or decimation.

The cipher is also used to perform encryption and decryption operations. Assume that we want to encrypt a plaintext message of length L defined MSB to LSB by the bits $m_0,\ldots,m_{L-1}$. The corresponding ciphertext symbols c_i are obtained from XOR-addition of plaintext symbols m_i and keystream symbols ks_i as shown in Figure C.4. Assume that we want to decrypt a ciphertext message of length L defined MSB to LSB by the bits $c_0,\ldots,c_{L-1}$. The corresponding plaintext symbols m_i are obtained from XOR-addition of ciphertext symbols c_i and keystream symbols ks_i as shown in Figure C.5.

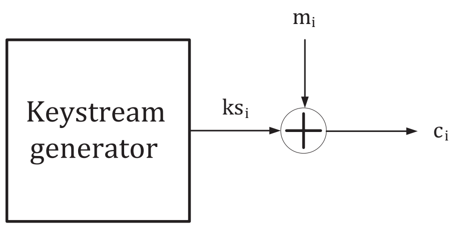


Figure C.4 — Encryption Operation

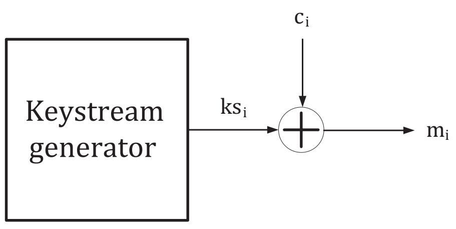


Figure C.5 — Decryption Operation

Annex D
(informative)

Test vectors

D.1 General

The test vectors allow for better understanding of the different usages of Grain-128A. Detailed test vectors are provided below for both MAC32 and MAC64 implementations.

D.2 Detailed test vectors with MAC32

Test vector set 1: Tag Authentication followed by MAC

This vector set illustrates how Grain-128A internal registers are set up and how several commands are chained. Refer to Table D.1 for the vectors used for test vector set 1.

First command is a Tag Authentication:

— The key [1] is loaded in the NFSR.[4]

— T_Random and I_Random are concatenated and loaded in the LFSR. The LFSR is padded with s_{96} = Tag being authenticated, s_{97} = Interrogator being authenticated, $s_i = 1, 98 \leq i \leq 126, s_{127} = 0$. Line [5] shows the full content of the LFSR after the set up.

— Grain-128A is then clocked 256 times. At the end of the operation the NFSR and LSFR respectively contain the values in [6] and [7].

— In line [8], Grain-128A generates the pre-output.

— The first 32 bits of the pre-output are used to set up the MAC accumulator [9]. Next 32 bits of the pre-output are used to set up the shift register of the MAC [10].

— The following bits of the pre-output (from bit 64 onwards) are split in 2 parts. The even bits compose the key stream [11], whereas the odd bits compose the MAC stream [12].

— Finally, the TKeyStream [14] is extracted from the 64 bits of the keystream.

— As the command is a Tag Authentication, IKeyStream is not generated.

— It has to be noted that, for this command, the Mac Stream is ignored.

— Remark: At this stage, the MAC registers, Accumulator and Shift register have been set up but not updated.

Second command is an Authenticated Communication and computes a MAC:

— At this stage, Grain-128A engine is not reinitialized. Instead, the context is inherited from the previous command, in this case the Tag Authentication. The MAC accumulator and shift register contain their set up value [19, 20]. The LSFR and NFSR are shown in lines [17,18].

— A pre-output of twice the length+1 of the message (82 bits) is generated [21].

— The even bits are extracted to compose the key stream [22]. The odds bits compose the MAC stream [23].

— The MAC stream is reinjected in the MAC engine to update the Accumulator and the Shift register.

— The MAC [25] consists in the final content of the Accumulator.

— It has to be noted, that this time the keystream is ignored.

Test vector set 2: Interrogator Authentication followed by MAC

This vector set is very similar to the first test vector set, performing an Interrogator Authentication instead of a Tag Authentication. It especially illustrates the effect of the IV padding during initialization of the cipher as defined in Clause 9. Refer to Table D.1 for the vectors used for test vector set 2.

First command is an Interrogator Authentication:

— The main difference from test vector set 1 appears in line [5]. The LFSR is padded with s_{96} = Tag being authenticated, s_{97} = Interrogator being authenticated, $s_i = 1, 98 \leq i \leq 126, s_{127} = 0$.

— The LFSR being set up differently, all the following values differ from test vector set 1.

— Finally, IKeyStream [13] is extracted from the keystream [11].

Second command is an Authenticated Communication and computes a MAC:

— Processing is in a similar way as for test vector set 1.

Table D.1 — Test Vector Set 1 and Set 2 for Grain-128A using MAC32

			Test Vector Set 1 using MAC32	**Test Vector Set 2 using MAC32**
		First command : Authentication	**Tag Authentication**	**Interrogator Authentication**
Inputs	1	Key[0:127]	0000000000000000 0000000000000000	0000000000000000 0000000000000000
	2	I_RANDOM[0:47]	800000000000	800000000000
	3	T_RANDOM[0:47]	000000000000	000000000000
Intermediate data	4	NFSR[0:127] after set up	0000000000000000 0000000000000000	0000000000000000 0000000000000000
	5	LFSR[0:127] after set up	8000000000000000 00000000**B**FFFFFFE	8000000000000000 00000000**7**FFFFFFE
	6	NFSR[0:127] after 256 steps	902A737F9A7B3038 6B94D1DA00390F77	2B66A445596E3DE6 BC7134C4BAAD023B
	7	LFSR[0:127] after 256 steps	A062786C5B23BECD AC72CC6A53FC3C79	C579D7468E2EE844 711301DEE67A484A
	8	pre-output stream[0..191]	62D65B2AB49F2458 CC3C07EC06170A8B 64740D484AB48852	EC6C2FB001BE0C16 A488E73086F0CD48 687210FD1E9B93D4
	9	Accumulator[0:31]	62D65B2A	EC6C2FB0
	10	Shift Register[0:31]	B49F2458	01BE0C16
	11	Keystream[0..63]	A61E113B44223CA1	CAD49CA2650E3B98
	12	Macstream[0..63]	A63A2701AE38860C	20B42CB88C4F655E
Outputs	13	IKeyStream[0..63]	N\A	CAD49CA2650E3B98
	14	TKeyStream[0..63]	A61E113B44223CA1	N\A

Table D.1 *(continued)*

		Second command: Authenticated Communication	**MAC only**	**MAC only**
Inputs	15	Message[0..39]	12345678AB	12345678AB
	16	Message length in bits	40	40
Intermediate data	17	NFSR[0..127] at start of the Command	948A926A91D7FA0C 31813D114D83FDA6	2085C14D99DBB859 F2813BB4065F37E6
	18	LFSR[0..127] at start of the Command	C762EB637F7F4B1E 0C782F1E8E6BD7D6	0162E4EA0316C8EC 7A78AA1BAFD74A60
	19	Shift Register[0..31] at start of the command	B49F2458	01BE0C16
	20	Accumulator[0..31] at start of the command	62D65B2A	EC6C2FB0
	21	Pre-Output[0..81]	090DD9F168BD2993 FF9B80	2FA73D3EFA3C5643 C629C0
	22	Key stream[0..40]	22AC6E69FB80	7D67F6119680
	23	Mac Stream[0..40]	13DD8715F500	3376C6E9A180
Outputs	24	Encrypted Message	N\A	N\A
	25	MAC[0..31]	4335B1F6	C7C85384

Data values for the table are written in hexadecimal. When the data length in bits is not a multiple of 8, for instance [21], [22] and[23], the value is padded with 0's at the end.

Test vector set 3: Mutual Authentication followed by MAC

This vector set is very similar to test vector sets 1 and 2, but performs a Mutual Authentication followed by a MAC computation. It illustrates again the effect of the IV padding during initialization of the cipher as defined in Clause 9. It also shows the necessity to generate more pre-output bits to generate both IKeyStream and TKeyStream. Refer to Table D.2 for the vectors used for test vector set 3.

First command is a Mutual Authentication and the differences with test vector sets 1 and 2 appear as:

— Line[5], the LFSR is padded with s_{96} = Tag being authenticated, s_{97} = Interrogator being authenticated, $s_i = 1, 98 \le i \le 126, s_{127} = 0$, impacting all the following values.

— Line [8], Grain-128A generates the pre-output.

— The first 32 bits of the pre-output are used to set up the MAC accumulator [9]. Next 32 bits of the pre-output are used to set up the shift register of the MAC [10].

— The following bits of the pre-output (from bit 64) are split in 2 parts. The even bits compose the key stream [11], whereas the odd bits compose the MAC stream [12].

— The IKeyStream [13] are composed of the 64 first bits of the keystream.

— The TKeyStream [14] are composed of the 64 next bits of the keystream

Second command is an Authenticated Communication and computes a MAC similar to the test vectors 1 and 2.

Test vector set 4: Mutual Authentication followed by MAC

This vector set is nearly identical to test vector set 3 and illustrates the forcing of $s_0 = 1$ during initialization of the cipher as defined in Clause 9. Refer to Table D.2 for the vectors used for test vector set 4.

First command is a Mutual Authentication and the differences with test vector set 3 appear as:

— Line[2], I_RANDOM is completely null.

— Line[5], I_RANDOM and T_RANDOM are concatenated. The LFSR is padded with s_{96} = Tag being authenticated, s_{97} = Interrogator being authenticated, $s_i = 1$, $98 \le i \le 126$, $s_{127} = 0$, impacting all the following values. Finally, s_0 is forced to 1.

Second command is an Authenticated Communication and computes a MAC identical to test vector set 3.

Table D.2 — Test Vector Set 3 and Set 4 for Grain-128A using MAC32

			Test Vector Set 3 using MAC32	**Test Vector Set 4 using MAC32**
		First command : Authentication	**Mutual Authentication**	**Mutual Authentication**
Inputs	1	Key[0:127]	0000000000000000 0000000000000000	0000000000000000 0000000000000000
	2	I_RANDOM[0:47]	800000000000	000000000000
	3	T_RANDOM[0:47]	000000000000	000000000000
Intermediate data	4	NFSR[0:127] after set up	0000000000000000 0000000000000000	0000000000000000 0000000000000000
	5	LFSR[0:127] after set up	8000000000000000 00000000**F**FFFFFFE	8000000000000000 00000000**F**FFFFFFE
	6	NFSR[0:127] after 256 steps	9D2C0C5281D33CB9 444720688B0A3A7A	9D2C0C5281D33CB9 444720688B0A3A7A
	7	LFSR[0:127] after 256 steps	A3F545F997EBC748 83A7E1384513C974	A3F545F997EBC748 83A7E1384513C974
	8	pre-output stream[0..319]	564B362219BD90E3 01F259CF52BF5DA9 DEB1845BE6993ABD 2D3C77C4ACB90E42 2640FBD6E8AE642A	564B362219BD90E3 01F259CF52BF5DA9 DEB1845BE6993ABD 2D3C77C4ACB90E42 2640FBD6E8AE642A
	9	Accumulator[0:31]	564B3622	564B3622
	10	Shift Register[0:31]	19BD90E3	19BD90E3
	11	Keystream[0..127]	0D2B1F2EBC83DA7E 6658EE3150F9EF47	0D2B1F2EBC83DA7E 6658EE3150F9EF47
	12	Macstream[0..127]	1CDBC7F1E52DA547 36FA252828DE82A0	1CDBC7F1E52DA547 36FA252828DE82A0
Outputs	13	IKeyStream[0..63]	0D2B1F2EBC83DA7E	0D2B1F2EBC83DA7E
	14	TKeyStream[0..63]	6658EE3150F9EF47	6658EE3150F9EF47
		Second command: Authenticated Communication	**MAC only**	**MAC only**
Inputs	15	Message[0..39]	12345678AB	12345678AB
	16	Message length in bits	40	40

Table D.2 *(continued)*

			Test Vector Set 3 using MAC32	**Test Vector Set 4 using MAC32**
		First command : Authentication	**Mutual Authentication**	**Mutual Authentication**
Intermediate data	17	NFSR[0..127] at start of the Command	2F0E190C3F28FF25 87E726F0CB3FA13B	2F0E190C3F28FF25 87E726F0CB3FA13B
	18	LFSR[0..127] at start of the Command	A429A0136DE6D407 5E4DE180E34E5209	A429A0136DE6D407 5E4DE180E34E5209
	19	Shift Register[0..31] at start of the command	19BD90E3	19BD90E3
	20	Accumulator[0..31] at start of the command	564B3622	564B3622
	21	Pre-Output[0..81]	9942C4B00AB-37C64E77FC0	9942C4B00AB-37C64E77FC0
	22	Key stream[0..40]	A18C3D64D780	A18C3D64D780
	23	Mac Stream[0..40]	58A405EABF80	58A405EABF80
Outputs	24	Encrypted Message	N\A	N\A
	25	MAC[0..31]	D594AD7D	D594AD7D

Data values for the table are written in hexadecimal. When the data length in bits is not a multiple of 8, for instance [21], [22] and[23], the value is padded with 0's at the end.

Test vector set 5: Mutual Authentication followed by encryption

Test vector set 5 is similar to test vector set 3. It starts by a Mutual Authentication, but continues with the MAC computation and encryption of a message. Refer to Table D.3 for the vectors used for test vector set 5.

First command is a Mutual Authentication and is the same as for test vector set 3.

Second command is a Secure Authenticated Communication and the MAC and encrypted command differ from test vector set 3 in the usage of the key stream:

— A pre-output of twice the length of the message (80 bits) is generated [21].

— The even bits are extracted to compose the key stream [22]. The odds bits compose the MAC stream [23].

— The message is encrypted performing a bit-wise XOR of the message and the keystream [22] to obtain the value [24].

— The MAC stream is reinjected in the MAC engine to update the Accumulator and the Shift register.

— The message bit that enters in the MAC engine is the bit of the encrypted message. It implies that the final value is not the same as for test vector sets 3 and 4.

— The MAC [25] consists in the final content of the Accumulator.

Test vector set 6: Mutual Authentication followed by encryption (non trivial values)

Test vector 6 is similar to test vector 5, but uses non-trivial values. It it useful to verify the bit order of the data. Refer to Table D.3 for the vectors used for test vector set 6.

Table D.3 — Test Vector Set 5 and Set 6 for Grain-128A using MAC32

			Test Vector Set 5 using MAC32	**Test Vector Set 6 using MAC32**
		First command : Authentication	**Mutual Authentication**	**Mutual Authentication**
Inputs	1	Key[0:127]	0000000000000000 0000000000000000	0123456789ABCDEF FEDCBA9876543210
	2	I_RANDOM[0:47]	800000000000	112233445566
	3	T_RANDOM[0:47]	000000000000	778899AABBCC
Intermediate data	4	NFSR[0:127] after set up	0000000000000000 0000000000000000	0123456789ABCDEF FEDCBA9876543210
	5	LFSR[0:127] after set up	8000000000000000 00000000**F**FFFFFFE	9122334455667788 99AABBCC**F**FFFFFFE
	6	NFSR[0:127] after 256 steps	9D2C0C5281D33CB9 444720688B0A3A7A	EBD538C90CF87DC1 CFEBF485DE38D75E
	7	LFSR[0:127] after 256 steps	A3F545F997EBC748 83A7E1384513C974	7631DCA9EF303CC2 E4B932C9C126315D
	8	pre-output stream[0:319]	564B362219BD90E3 01F259CF52BF5DA9 DEB1845BE6993ABD 2D3C77C4ACB90E42 2640FBD6E8AE642A	4BD5F24D4464B119 1AF86A6A62B042D2 31E66DF620FFA6D4 D1D230BA94C15E0D 05E6E284C7D7D653
	9	Accumulator[0:31]	564B3622	4BD5F24D
	10	Shift Register[0:31]	19BD90E3	4464B119
	11	Keystream[0..127]	0D2B1F2EBC83DA7E 6658EE3150F9EF47	3E775C194D6D4FD8 894F88320DD89991
	12	Macstream[0..127]	1CDBC7F1E52DA547 36FA252828DE82A0	4C88848C5ABE0F2E DC4469E33A82BFED
Outputs	13	IKeyStream[0..63]	0D2B1F2EBC83DA7E	3E775C194D6D4FD8
	14	TkeyStream[[0..63]	6658EE3150F9EF47	894F88320DD89991
		Second command: Secure Authenticated Communication	**MAC and Encryption**	**MAC and Encryption**
Inputs	15	Message[0..39]	12345678AB	12345678AB
	16	Message length in bits	40	40

Table D.3 *(continued)*

			Test Vector Set 5 using MAC32	**Test Vector Set 6 using MAC32**
		First command : Authentication	**Mutual Authentication**	**Mutual Authentication**
Intermediate data	17	NFSR[0..127] at start of the Command	2F0E190C3F28FF25 87E726F0CB3FA13B	624650EF2334AD45 0EFC8BDB7ED6A7A9
	18	LFSR[0..127] at start of the Command	A429A0136DE6D407 5E4DE180E34E5209	650D94DA709D0037 2DE7906201718BD3
	19	Shift Register[0..31] at start of the command	19BD90E3	4464B119
	20	Accumulator[0..31] at start of the command	564B3622	4BD5F24D
	21	Pre-Output[0..81]	9942C4B00AB-37C64E77FC0	772BCE1E9F-5166EA3DBC0
	22	Key stream[0..40]	A18C3D64D780	57B3B05F6F80
	23	Mac Stream[0..40]	58A405EABF80	F1A67DA87580
Outputs	24	Encrypted Message[0..39]	B3B86B1C7C	4587E627C4
	25	MAC[0..31]	66789267	D495799A

Data values for the table are written in hexadecimal. When the data length in bits is not a multiple of 8, for instance [21], [22], and [23], the value is padded with 0’s at the end.

D.3 Detailed test vectors with MAC64

Test vector sets 1 and 2 of this section have the same inputs as test vector sets 1 and 2 of the previous section. Nevertheless, the MAC is configured to 64 bits for this section instead of 32 bits as in the previous section. It influences all the results including the IKeyStream and the TKeyStream [2,3]. The encryption of the message [8] is also affected, together, of course with the MAC value [9].

Table D.4 — Test Vector Set 1 and Set 2 for Grain-128A using MAC64

			Test Vector Set 1 using MAC64	**Test Vector Set 2 using MAC64**
		First command : Authentication	**Tag Authentication**	**Interrogator Authentication**
Inputs	1	Key[0:127]	0000000000000000 0000000000000000	0000000000000000 0000000000000000
	2	I_RANDOM[0:47]	800000000000	800000000000
	3	T_RANDOM[0:47]	000000000000	000000000000
Outputs	4	IKeyStream[0..63]	N\A	650E3B987D67F611
	5	TKeyStream[0..63]	44223CA122AC6E69	N\A
Inputs	6	Message[0..39]	12345678AB	12345678AB
	7	Message length in bits	40	40
Outputs	8	Encrypted Message[0..39]	N\A	N\A
	9	MAC[0..63]	84E0EA3EDD6C0825	A66CEE82D876E368

Annex E
(normative)

Protocol specific

E.1 Applicable air interface protocols

The Grain-128A CS provides security services for HF and UHF air interface protocols as described in E.2 and E.3, respectively. ISO/IEC 29167-1 defines the Crypto Suite Identifier (CSI) for Grain-128A to be 000011_b and it is expanded to the 8-bit value 03_h for use by all air interface protocols in this Annex.

E.2 Security Services for ISO/IEC 18000-3 Mode 1

RFU - To be completed in the future when 18000-3 information is available.

E.3 Security Services for ISO/IEC 18000-63

E.3.1 General

Interrogators and Tags implementing the Grain-128A CS shall provide the security services shown in Table E.1 using the protocol commands shown in Table E.3. During authentication, Tags shall report the features implemented in support of the Grain-128A CS as shown in Table E.2. All security services that are provided shall be implemented as defined in Clauses 10 and 11.

Table E.1 — Security Services Provided

Security Service	Method	Mandatory or Optional
Authentication		
	00_b : Tag authentication (TA)	Mandatory
	01_b : Interrogator authentication (IA)	Optional
	10_b : Mutual authentication (MA)	Mandatory
	11_b : Vendor defined	Optional
Communication		
Authenticated Tag from TA	Authenticated communication (Tag => Interrogator)	Mandatory
	Secure authenticated communication (Tag => Interrogator)	Optional
Authenticated Interrogator from IA (when IA provided)	Authenticated communication (Interrogator => Tag)	Mandatory
	Secure authenticated communication (Interrogator => Tag)	Optional

Table E.1 *(continued)*

Security Service	Method	Mandatory or Optional
Authenticated Interrogator and Tag from MA	Authenticated communication (Interrogator <=> Tag)	Mandatory
	Secure authenticated communication (Interrogator <=> Tag)	Optional
Key Update	Using *KeyUpdate* command	Optional

Table E.2 — Tag Implemented Features (CSFeatures[7:0])

Name	Description	Mandatory or Vendor defined
CSFeatures[7]	Vendor defined	Vendor defined
CSFeatures[6]	0 = Encrypted read of hidden memory not supported, 1 = Encrypted read of hidden memory supported	Vendor defined
CSFeatures[5]	0 = Key update not supported, 1 = Key update supported	Vendor defined
CSFeatures[4]	0 = Secure authenticated communication not supported, 1 = Secure authenticated communication supported	Vendor defined
CSFeatures[3]	0 = MAC64 not supported, 1 = MAC64 supported	Vendor defined
CSFeatures[2]	0 = MAC32 not supported, 1 = MAC32 supported	Vendor defined
CSFeatures[1]	0 = IA not supported, 1 = IA supported	Vendor defined
CSFeatures[0]	1 = TA supported	Mandatory

NOTE It is mandatory to support at least one MAC feature (MAC32 and/or MAC64).

Table E.3 — Protocol Command Implementations

Command	Feature	Mandatory or Vendor defined
Challenge	Tag and Interrogator implement the command for all authentication types provided by the Tag	Mandatory
	Tag execution time shall be ≤ 5ms	Mandatory
	Tag ignores all other commands from an Interrogator during execution	Mandatory
	Tag and Interrogator support sending ResponseBuffer during reply to an *ACK* command and a *ReadBuffer* command	Mandatory
	Tag supports security timeout for a crypto error	Vendor defined

Table E.3 *(continued)*

Command	Feature	Mandatory or Vendor defined
Authenticate	Tag and Interrogator implement the command for all authentication types provided by the Tag	Mandatory
	Tag execution time shall be ≤ 5ms	Mandatory
	Tag and Interrogator support sending the response during reply to the *Authenticate* command and a *ReadBuffer* command	Mandatory
	Tag supports security timeout for a crypto error	Vendor defined
	During authentication, a Tag does not reply to a command having an invalid handle or invalid CRC, the crypto engine is reset, and the next Tag state is open	Mandatory
	During authentication, a crypto error resets the crypto engine, the Tag replies with the error code defined in E.3.3, and the next Tag state is arbitrate	Mandatory
	During authentication, the next Tag state is open for a successful Tag Authentication and the next Tag state is secured for a successful Interrogator Authentication or Mutual Authentication	Mandatory
AuthComm	Tag and Interrogator implement the command	Mandatory
	Tag supports security timeout for a crypto error	Vendor defined
	During authenticated communication in the open state or secured state, a Tag does not reply to a command having an invalid handle or invalid CRC, the crypto engine is reset, and the next Tag state is open	Mandatory
	During authenticated communication in the open state or secured state, a crypto error resets the crypto engine, the Tag replies with the error code defined in E.3.3, and the next Tag state is arbitrate	Mandatory
SecureComm	Tag and Interrogator implement the command	Vendor defined
	Tag supports sending the response during reply to the *SecureComm* command and a *ReadBuffer* command	Mandatory
	Tag supports security timeout for a crypto error	Vendor defined
	During secure authenticated communication in the secured state, a Tag does not reply to a command having an invalid handle or invalid CRC, the crypto engine is reset, and the next Tag state is open	Mandatory
	During secured authenticated communication in the secured state, a crypto error resets the crypto engine, the Tag replies with the error code defined in E.3.3, and the next Tag state is arbitrate	Mandatory
KeyUpdate	Tag and Interrogator implement the command	Vendor defined
	Supported only with encapsulation	Mandatory

The *Challenge and/or Authenticate* command shall be used for all authentication methods provided by a Tag. Tags shall implement both Tag authentication and mutual authentication and may implement Interrogator authentication.

Authentication integrity shall be maintained for an Interrogator authentication and a Mutual authentication. It is an option to maintain the integrity of a Tag authentication. Integrity shall be performed using authenticated communication and/or secure authenticated communication. Authenticated communication shall be implemented via the use of *AuthComm* commands and secure authenticated communication shall be implemented via the use of *SecureComm* commands.

Integrity of an authenticated Interrogator shall be required when all of the following conditions occur:

— The Interrogator is authenticated via IA or MA.

— The Tag is in the Secured state.

— The Interrogator command is to access the singulated Tag and have the Tag remain in the Secured state.

Integrity of an authenticated Tag shall be required when all of the following conditions occur:

— The Tag is authenticated via MA.

— The Tag is in the Secured state.

— The Interrogator command is to access the singulated Tag and have the Tag remain in the Secured state.

Integrity of an authenticated Tag may be required at the discretion of the Interrogator when all of the following conditions occur:

— The Tag is authenticated via TA.

— The Tag is in the Open or Secured state.

— The Interrogator command is to access the singulated Tag.

The Tag shall indicate if it supports encrypted read of hidden memory. If supported, then the Tag shall use the crypto key specified by KeyID during authentication to determine if read access to hidden memory is permitted. If permitted, then the Interrogator may read the hidden memory but the memory contents shall be encrypted in the Tag reply.

E.3.2 Protocol Commands for Security Services

E.3.2.1 Challenge Command

The *Challenge* command allows an Interrogator to instruct multiple Tags to simultaneously yet independently precompute cryptographic values for use in a subsequent Tag, Interrogator, or mutual authentication, and to store the computed value(s) for subsequent use. Because all cryptographic algorithms require significant computation time, such parallel precomputation can significantly accelerate the authentication of a population of Tags. The *Challenge* command provides a payload message as shown in Table E.4 for the Tag CS to execute during authentication. The *Challenge* reply provides a payload message as shown in Table E.5 for the Interrogator CS to execute during authentication. The CryptoAuthCmd(Payload) and CryptoAuthResp(Payload) are defined in Section 10.

Table E.4 — Challenge Command

Challenge OpCode	**RFU**	**IncRepLen**	**Immed**	**CSI**	**Length**	**Message = CryptoAuthCmd(Payload)**	**CRC**

Table E.5 — Reply to Challenge Command

Length (optional)	Message = CryptoAuthResp(Payload)

E.3.2.2 Authenticate Command

The *Authenticate* command allows an Interrogator to instruct a singulated Tag to compute cryptographic values for use in Tag, Interrogator, or mutual authentication. The *Authenticate* command provides a payload message as shown in Table E.6 for the Tag CS to execute during authentication. The *Authenticate* reply provides a payload message as shown in Table E.7 for the Interrogator CS to execute during authentication.

The CryptoAuthCmd(Payload) and CryptoAuthResp(Payload) are defined in Clause 10. The *Authenticate* command provides the only cryptographic means to transition a Tag into the Secured state.

Table E.6 — Authenticate Command

Authenticate OpCode	RFU	SenRep	IncRepLen	CSI	Length	Message = CryptoAuthCmd(Payload)	RN	CRC

Table E.7 — Reply to Authenticate Command

Barker Code	Done	Header	Length (optional)	Response = CryptoAuthResp(Payload)	RN	CRC

E.3.2.3 AuthComm Command

An *AuthComm* command provides a means for ensuring the integrity of communication with an authenticated Interrogator and/or Tag. The *AuthComm* command provides a payload message as shown in Table E.8 which encapsulates another command intended for the Tag to execute. The payload message for a command from an authenticated Interrogator includes a MAC to maintain integrity once IA or MA is achieved. The *AuthComm* reply as shown in Table E.9 encapsulates the reply from the Tag regarding the execution of the encapsulated command. The payload message for a reply from an authenticated Tag includes a MAC to maintain integrity once TA or MA is achieved. The MAC shall not be included in the payload messages for a non-authenticated Interrogator or a non-authenticated Tag. The use of MAC's in CryptoCommCmd(Payload) and CryptoCommResp(Payload) are defined in 11.2.

Table E.8 — AuthComm Command

AuthComm OpCode	RFU	IncRepLen	Message = CryptoCommCmd(Payload)	RN	CRC

Table E.9 — Reply to AuthComm Command

Barker Code	Done	Header	Length (optional)	Response = CryptoCommResp(Payload)	RN	CRC

E.3.2.4 SecureComm Command

A *SecureComm* command provides a means to prevent eavesdropping while ensuring the integrity of communication with an authenticated Interrogator and Tag once MA is achieved. The *SecureComm* command provides an encrypted payload message as shown in Table E.10 which encapsulates another command intended for the Tag to execute. The *SecureComm* reply as shown in Table E.11 provides an encrypted payload message which encapsulates the reply from the Tag regarding the execution of the encapsulated command.

Table E.10 — SecureComm Command

SecureComm OpCode	RFU	SenRep	IncRepLen	Length	Message = CryptoSecCommCmd(Payload)	RN	CRC

Table E.11 — Reply to SecureComm Command

Barker Code	Done	Header	Length (optional)	Response = CryptoSecCommResp(Payload)	RN	CRC

E.3.2.5 KeyUpdate Command

A *KeyUpdate* command provides a means for an authenticated Interrogator to modify the value of a crypto Key. A *KeyUpdate* command shall always be encapsulated within an *AuthComm* or *SecureComm* command. If the *KeyUpdate* is encapsulated in a *SecureComm* then the CryptoKeyCmd(Payload) shall not be encrypted before it is included in the CryptoSecCommCmd(Payload). If the *KeyUpdate* is encapsulated in a *AuthComm* then it shall be encrypted before it is included as the CryptoKeyCmd(Payload).

Table E.12 — KeyUpdate Command

KeyUpdate OpCode	RFU	SenRep	IncRepLen	KeyID	Length	Message = CryptoKeyCmd(Payload)	RN	CRC

E.3.3 Reporting Crypto Suite Errors

Crypto suite error conditions defined in Annex B that may be reported to the Interrogator shall use the following error code.

Table E.13 — Crypto Suite Error Code

Error Type in Annex B	ISO/IEC 18000-63 Error Code	ISO/IEC 18000-63 Error-Code Name	Error Description
Type 1	00000101_b	Crypto suite error	This is a non-catastrophic error specified by the cryptographic suite that shall be reported to the Interrogator using a defined 18000-63 error code in a error Tag reply
Type 2	--	--	This is a catastrophic error that the Tag shall report to the Interrogator via a parameter in a non-error Tag reply
Type 3	--	--	This is a catastrophic error and the Tag shall not reply to the Interrogator

E.3.4 Example of Tag Authentication using *Challenge*

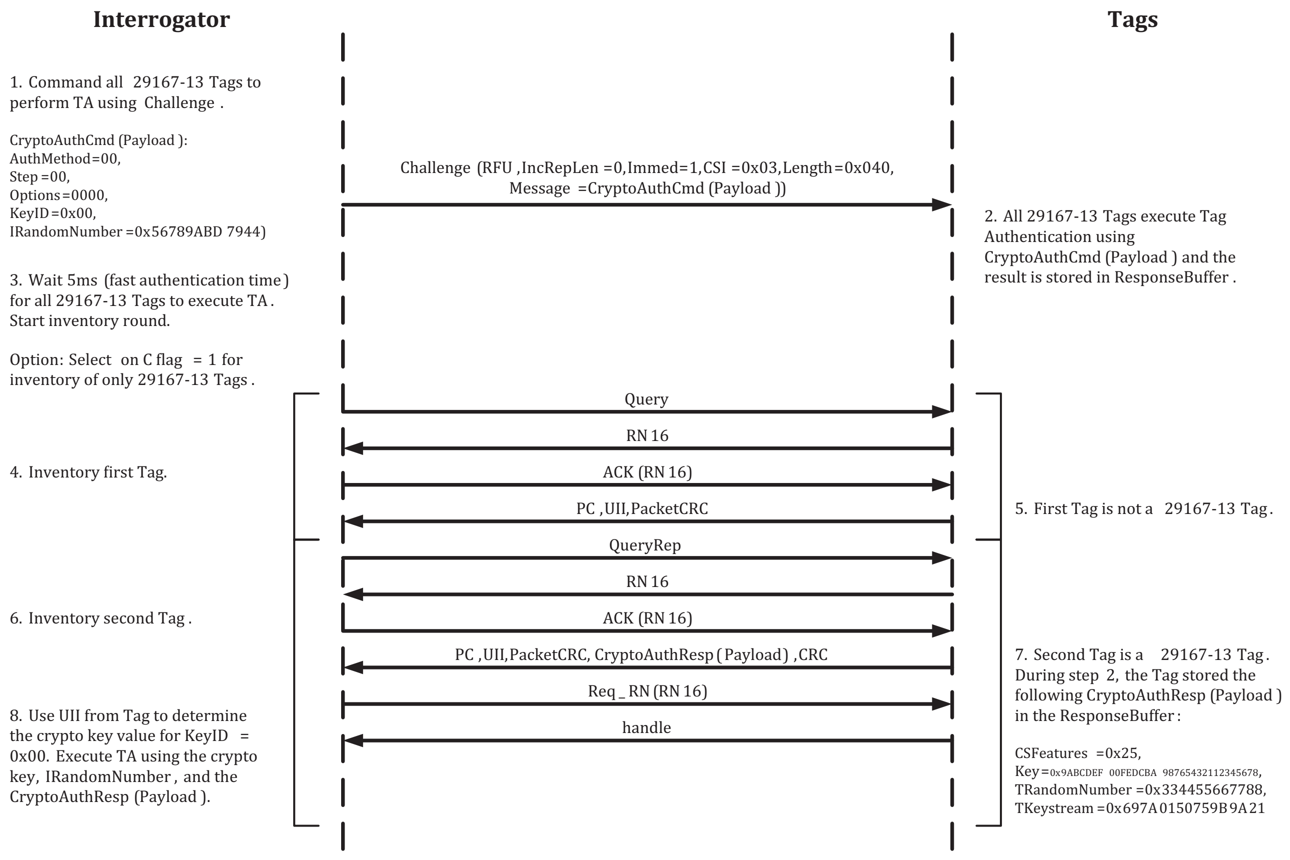
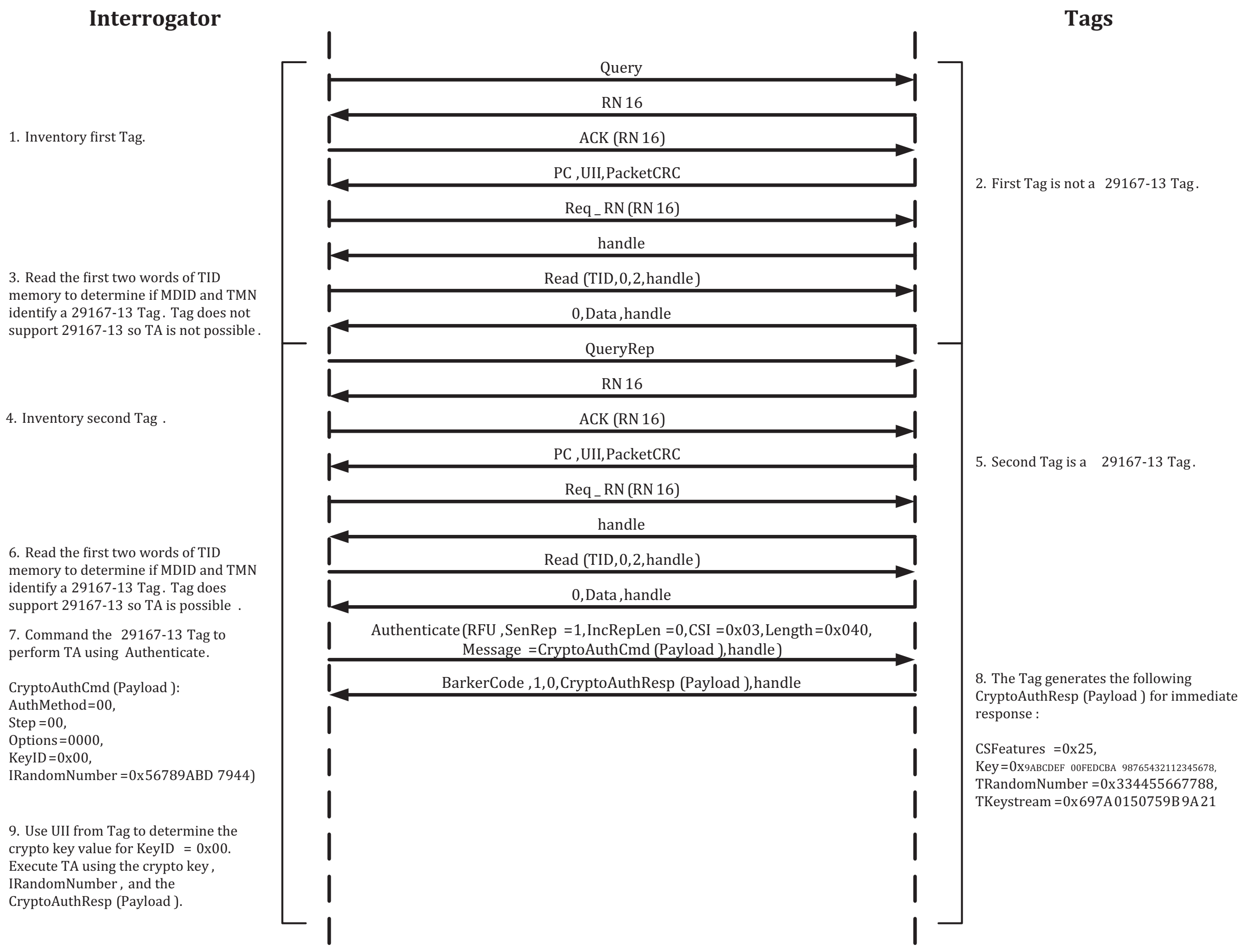

Interrogator
Tags
1. Command all 29167-13 Tags to perform TA using Challenge .
CryptoAuthCmd (Payload):
AuthMethod=00,
Step =00,
Options=0000,
KeyID=0x00,
IRandomNumber =0x56789ABD 7944)
Challenge (RFU ,IncRepLen =0,Immed=1,CSI =0x03,Length=0x040,
Message =CryptoAuthCmd (Payload))
2. All 29167-13 Tags execute Tag Authentication using CryptoAuthCmd (Payload) and the result is stored in ResponseBuffer .
3. Wait 5ms (fast authentication time) for all 29167-13 Tags to execute TA . Start inventory round.
Option: Select on C flag = 1 for inventory of only 29167-13 Tags .
Query
RN 16
4. Inventory first Tag.
ACK (RN 16)
PC ,UII,PacketCRC
5. First Tag is not a 29167-13 Tag .
QueryRep
RN 16
6. Inventory second Tag .
ACK (RN 16)
PC ,UII,PacketCRC, CryptoAuthResp (Payload) ,CRC
7. Second Tag is a 29167-13 Tag . During step 2, the Tag stored the following CryptoAuthResp (Payload) in the ResponseBuffer :
Req _ RN (RN 16)
handle
8. Use UII from Tag to determine the crypto key value for KeyID = 0x00. Execute TA using the crypto key, IRandomNumber , and the CryptoAuthResp (Payload).
CSFeatures =0x25,
Key=0x9ABCDEF 00FEDCBA 98765432112345678,
TRandomNumber =0x334455667788,
TKeystream =0x697A0150759B9A21

Figure E.1 — 18000-63 TA using *Challenge*

E.3.5 Example of Tag Authentication using *Authenticate*



1. Inventory first Tag.

2. First Tag is not a 29167-13 Tag.

3. Read the first two words of TID memory to determine if MDID and TMN identify a 29167-13 Tag. Tag does not support 29167-13 so TA is not possible.

4. Inventory second Tag .

5. Second Tag is a 29167-13 Tag.

6. Read the first two words of TID memory to determine if MDID and TMN identify a 29167-13 Tag. Tag does support 29167-13 so TA is possible .

7. Command the 29167-13 Tag to perform TA using Authenticate.

CryptoAuthCmd (Payload):
AuthMethod=00,
Step =00,
Options=0000,
KeyID=0x00,
IRandomNumber =0x56789ABD 7944)

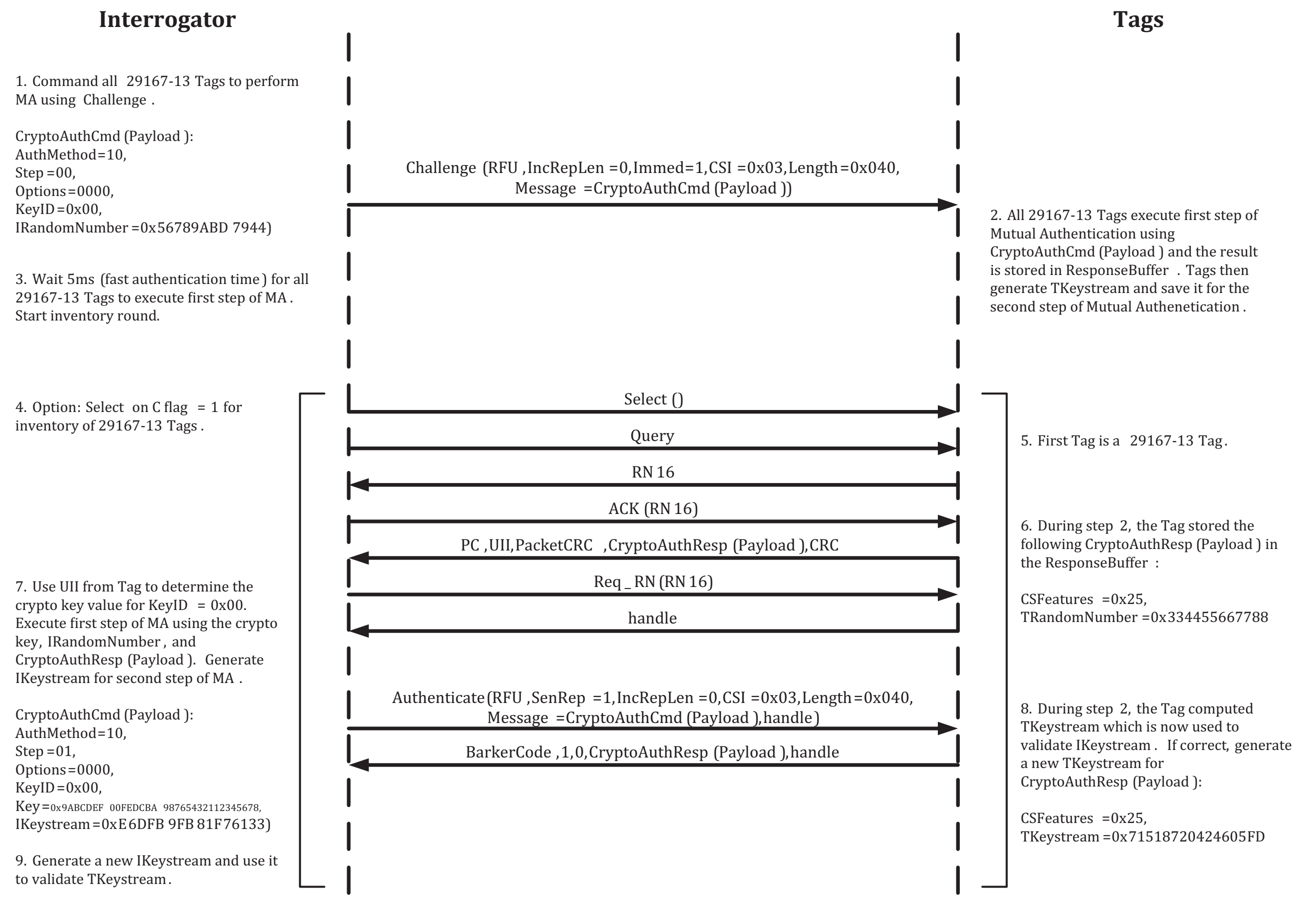
8. The Tag generates the following CryptoAuthResp (Payload) for immediate response :

CSFeatures =0x25,
Key=0x9ABCDEF 00FEDCBA 98765432112345678,
TRandomNumber =0x334455667788,
TKeystream =0x697A0150759B9A21

9. Use UII from Tag to determine the crypto key value for KeyID = 0x00. Execute TA using the crypto key , IRandomNumber , and the CryptoAuthResp (Payload).

Figure E.2 — 18000-63 TA using *Authenticate*

E.3.6 Example of Mutual Authentication using *Challenge*


Interrogator
Tags
Challenge (RFU ,IncRepLen =0,Immed=1,CSI =0x03,Length=0x040,
Message =CryptoAuthCmd (Payload))
Select ()
Query
RN 16
ACK (RN 16)
PC ,UII,PacketCRC ,CryptoAuthResp (Payload),CRC
Req _ RN (RN 16)
handle
Authenticate(RFU ,SenRep =1,IncRepLen =0,CSI =0x03,Length=0x040,
Message =CryptoAuthCmd (Payload),handle)
BarkerCode ,1,0,CryptoAuthResp (Payload),handle

1. Command all 29167-13 Tags to perform MA using Challenge .

CryptoAuthCmd (Payload):
AuthMethod=10,
Step =00,
Options=0000,
KeyID=0x00,
IRandomNumber =0x56789ABD 7944)

2. All 29167-13 Tags execute first step of Mutual Authentication using CryptoAuthCmd (Payload) and the result is stored in ResponseBuffer . Tags then generate TKeystream and save it for the second step of Mutual Authenetication .

3. Wait 5ms (fast authentication time) for all 29167-13 Tags to execute first step of MA . Start inventory round.

4. Option: Select on C flag = 1 for inventory of 29167-13 Tags .

5. First Tag is a 29167-13 Tag .

6. During step 2, the Tag stored the following CryptoAuthResp (Payload) in the ResponseBuffer :

CSFeatures =0x25,
TRandomNumber =0x334455667788

7. Use UII from Tag to determine the crypto key value for KeyID = 0x00. Execute first step of MA using the crypto key, IRandomNumber , and CryptoAuthResp (Payload). Generate IKeystream for second step of MA .

CryptoAuthCmd (Payload):
AuthMethod=10,
Step =01,
Options=0000,
KeyID=0x00,
Key=0x9ABCDEF 00FEDCBA 98765432112345678,
IKeystream=0xE 6DFB 9FB 81F76133)

8. During step 2, the Tag computed TKeystream which is now used to validate IKeystream . If correct, generate a new TKeystream for CryptoAuthResp (Payload):

CSFeatures =0x25,
TKeystream =0x71518720424605FD

9. Generate a new IKeystream and use it to validate TKeystream .

Figure E.3 — 18000-63 MA using *Challenge*

Bibliography

[1] New European Schemes for Signatures, Integrity, and Encryption (NESSIE), URL: https://www.cosic.esat.kuleuven.be/nessie

[2] eSTREAM: The ECRYPT Stream Cipher Project, URL: http://www.ecrypt.eu.org/stream

[3] Hell M., Johansson T., Meier W. `Grain — A stream cipher for constrained environments.', International Journal of Wireless and Mobile Computing. Special Issue on Security of Computer Network and Mobile Systems. 2006, **2** (1) pp. 86–93

[4] Berbain C., Gilbert H., Maximov A. 2006), Cryptanalysis of Grain, in M. Robshaw, ed., `Fast Software Encryption 2006', Vol. 4047 of Lecture Notes in Computer Science, Springer-Verlag, pp. 15-29

[5] Hell M., Johansson T., Maximov A., Meier W. 2006), 'A Stream Cipher Proposal: Grain-128', International Symposium on Information Theory – ISIT, 2006, IEEE

[6] Agren 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. 2011, **5** (1) pp. 48–59

[7] ISO/IEC 18000-3, *Information technology — Radio frequency identification for item management — Part 3: Parameters for air interface communications at 13,56 MHz*

[8] ISO/IEC 18000-63, *Information technology — Radio frequency identification for item management — Part 63: Parameters for air interface communications at 860 MHz to 960 MHz Type C*

ICS 35.040

Price based on 39 pages