TECHNICAL SPECIFICATION

# ISO/IEC TS 29167-15

First edition
2017-09

# Information technology — Automatic identification and data capture techniques —

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

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

*Partie 15: Services de sécurité par suite cryptographique XOR pour communications d'interface radio*

Reference number
ISO/IEC TS 29167-15:2017(E)

## COPYRIGHT PROTECTED DOCUMENT



ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

# 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 voluntary nature of standards, the m teeaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

A list of all parts in the ISO/IEC 29167 series can be found on the ISO website.

# Introduction

This document defines a coding suite based on an exclusive or (XOR) operation for the ISO/IEC 18000 air interfaces standards for radio frequency identification (RFID) devices.

XOR is a type of logical disjunction on two operands that results in a value of true if exactly one of the operands has a value of true. The primary advantage of XOR operation is that it is simple to implement and that the XOR operation is computationally inexpensive for hiding information in cases where either no particular or light security is required. The simple implementation of XOR does not require a cipher and therefore limits the security protection and attacks like eaves dropping are much easier.

The security service tag authentication is a mandatory security service. All other services in this coding suite are optional. Every manufacturer has the liberty to chose which of these services will be implemented on a tag.

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 may be obtained from:

Patent holder: China IWNCOMM Co., Ltd.

Address: A201, QinFengGe, Xi'an Software Park,
No. 68, Keji 2nd Road,
Xi'an Hi-Tech Industrial Development Zone
Xi'an, Shaanxi, P. R. China 710075

The latest information on IP that may be applicable to this document can be found at www.iso.org/patents.

# Information technology — Automatic identification and data capture techniques —

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

## 1 Scope

This document defines a coding suite based on an exclusive or (XOR) operation for the ISO/IEC 18000 air interfaces standards for radio frequency identification (RFID) systems. In particular, it specifies the use of XOR as a basic way to hide plain data in the identity authentication and secure communication procedures. The coding suite is defined in alignment with existing air interfaces.

This document defines various authentication methods and methods of use for the XOR. A tag and an interrogator may support one, a subset, or all of the specified options, clearly stating what is supported.

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 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*

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

## 3 Terms, definitions, symbols and abbreviated terms

### 3.1 Terms and definitions

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

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

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at http://www.iso.org/obp

**3.1.1**
**command**
<message> command that interrogator sends to tag with "Message" as parameter

**3.1.2**
**message**
part of the command that is defined by the CS

**3.1.3**
**reply**
<response> reply that tag returns to the interrogator with "Response" as parameter

**3.1.4**
**response**
part of the reply (stored or sent) that is defined by the CS

## 3.2 Symbols and abbreviated terms

### 3.2.1 Symbols

| | |
|---|---|
| $\oplus$ | exclusive or |
| # | number |
| $xxxx_h$ | hexadecimal notation |
| \|\| | concatenation |
| $O_n$ | fixed value |
| + | a + b means a addition b mod $2^n$, the length of a and b is n. |
| – | a – b means binary subtraction operation. Given two binary numbers a and b, the operation a – b outputs the result of subtracting b from a.<br>NOTE The easiest way to subtract the second binary number from the first one is to make the second number negative and then add it with the first number. |
| mod | modulo operation |

### 3.2.2 Abbreviated terms

| | |
|---|---|
| CRC | cyclic redundancy check |
| CS | coding suite |
| CSI | coding suite identifier |
| EBV | extensive bit vector (see ISO/IEC 18000-63) |
| ID | identifier |
| MAC | message authentication code |
| PSK | pre-shared key |
| RFID | radio frequency identification |
| RFU | reserved for future use |
| RN | random number |
| SK | session key |

TRAIS tag and reader air interface security

TRAIS-X tag and reader air interface security based on XOR

XOR exclusive or

# 4 Conformance

## 4.1 Claiming conformance

To claim conformance with this document, an interrogator or tag shall comply with all relevant clauses of this document, except those marked as “optional”.

## 4.2 Interrogator conformance and obligations

To conform to this document, an interrogator shall

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

To conform to this document, an interrogator may

— implement any subset of the optional commands defined in this document.

To conform to this document, the interrogator shall not

— implement any command that conflicts with this document, or

— require the use of an optional, proprietary or custom command to meet the requirements of this document.

## 4.3 Tag conformance and obligations

To conform to this document, a tag shall

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

To conform to this document, a tag may

— implement any subset of the optional commands defined in this document.

To conform to this document, a tag shall not

— implement any command that conflicts with this document, or

— require the use of an optional, proprietary or custom command to meet the requirements of this document.

# 5 Cipher introduction

The logical operation exclusive disjunction, also called eXclusive OR (XOR) is a type of logical disjunction on two operands that results in a value of true if exactly one of the operands has a value of true and often used for bitwise operations or algebra computing. For example:

Bitwise operation:

— $1 \oplus 1 = 0$

— $1 \oplus 0 = 1$

— $0 \oplus 1 = 1$

— $0 \oplus 0 = 0$

— $a \oplus b = a + b \pmod 2$

The XOR operator is extremely common as a component in complex ciphers. By itself, using a constant repeating key, a simple XOR crypto can trivially be broken using frequency analysis. If the content of any message can be guessed or otherwise known then the key can be revealed (the XOR crypto is vulnerable to a known-plaintext attack, since plaintext $\oplus$ ciphertext = key). Its primary advantage is that it is simple to implement and that the XOR operation is computationally inexpensive. A simple repeating XOR crypto is therefore sometimes used for hiding information in cases where either no particular or light security is required. For detailed cipher descriptions, see Annex C. For some security considerations of this coding suite, see Annex G.

# 6 Parameter definitions

**Table 1 — Definition of parameters**

| Parameter | Description |
|---|---|
| Command Code [7:0] | The values of security commands (See 3.1.1 for the definition of Command) |
| RFU[7:0] | The reserved values for future use |
| Coding Suite ID [7:0] | CSI: coding suite identifier |
| Length[Variable] | The length of message with extensive bit vector format |
| Payload[Variable] | Message data (See 3.1.2 for the definition of Message) |
| CRC-16[15:0] | The cyclic redundancy check value |
| Message | See 3.1.2 |
| Reply | See 3.1.3 |
| Response | See 3.1.4 |
| RN[63:0] | 64-bit random number |
| Header[1:0] | The value of header |
| AuthType[1:0] | This shows the authentication type in the authentication procedure. The values are as follows:<br>— 00: mutual authentication<br>— 01: interrogator authentication<br>— 10: tag authentication<br>— 11: RFU |
| AuthStep[2:0] | This shows the step number in the authentication procedure. The values are as follows:<br>— 000: RFU<br>— 001: Step 1 of Authenticate command<br>— 010: Step 2 of Authenticate command<br>— 011-111: RFU |

**Table 1** *(continued)*

| Parameter | Description |
|---|---|
| Key ID[4:0] | The key identifier that the tag and interrogator used in the authentication procedure. |
| AuthData[Variable] | This shows the data computed in the authentication procedure. The values are as follows:<br>— SORNi = (RNi′ + $O_n$) ⊕ PSK′<br>— SORNt = (RNt′ + $O_n$) ⊕ PSK′<br>— SRNi = RNi ⊕ PSK<br>— SRNt = RNt ⊕ PSK<br>— NULL<br>where<br>— RNi′ : RNi′ means bit-wise ROTATE RNi left for n bits, where RNi is a 64-bit random number generated by an interrogator, n is the number of binary value 1 of RNi<br>— RNt : RNt′ means bit-wise ROTATE RNt left for n bits, where RNt is a 64-bit random number generated by a tag, n is the number of binary value 1 of RNt<br>— $O_n$ : 5555 5555 5555 5555h<br>— PSK′: PSK′ means bit-wise ROTATE PSK left for n bits, PSK is a value of pre-shared key (64-bit), n is the number of binary value 1 of RNi or RNt |
| MAC[127:0] | The value of message authentication code |

## 7 State diagram

Figure 1 shows the state machine of XOR coding suite. The state diagram for this coding suite consists of four states. For state transition tables, Annex A shall be consulted.

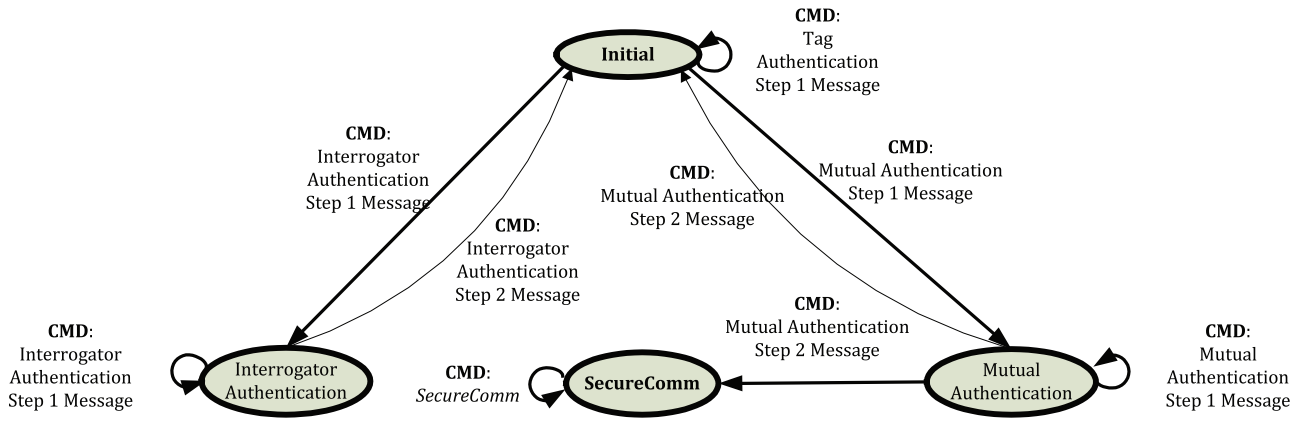


**Figure 1 — State diagram**

## 8 Initialization and resetting

This document shall implement an **Initial** state.

After power-up and after a reset of the coding suite the tag moves into the **Initial** state.

Implementations of this suite shall assure that all memory used for intermediate results is cleared after each operation (message-response pair) and after reset.

# 9 Authentication

## 9.1 General

This document describes additions to the ISO/IEC 18000 series of standards protocol to support the tag and reader air interface security (TRAIS) based on XOR (TRAIS-X). Specially, it defines

— the use of XOR crypto for mutual, interrogator and tag authentication procedures;

— the use of XOR crypto for secure communication;

— the encoding in the related commands and the processing of those messages.

Figures 2 and 3 shows protocol flows of mutual and interrogator, and tag authentication procedures, respectively.

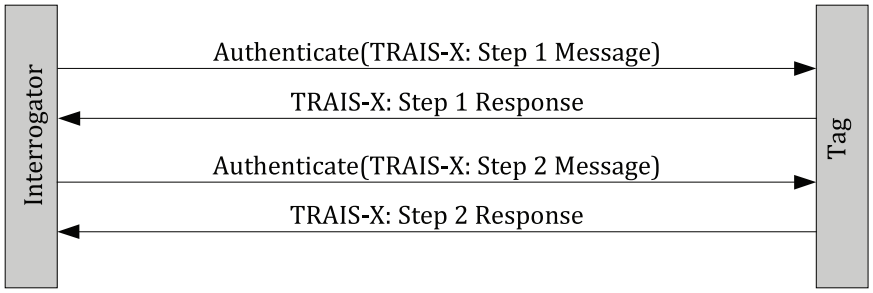


**Figure 2 — TRAIS-X mutual and interrogator authentication protocol flows**

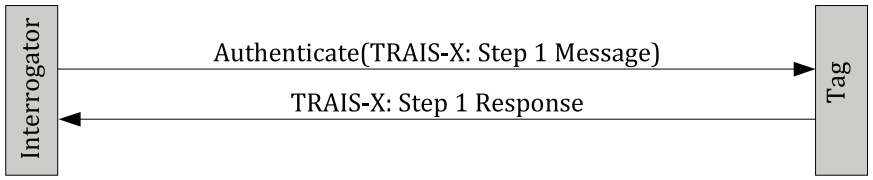


**Figure 3 — TRAIS-X tag authentication**

The formats of authenticate and response commands are shown in Table E.1 and Table E.2, respectively.

## 9.2 Authentication procedure

### 9.2.1 Protocol requirements

The authentication protocol requires that a tag and interrogator should have a PSK before they start the authentication procedure. How to generate and set a high quality PSK is out of the scope of this document. A key update function is supported and described in Clause 11. For error codes and error handling, the process in Annex B shall be followed.

### 9.2.2 Procedure

#### 9.2.2.1 Mutual authentication

The mutual authentication procedure is as follows.

a) The interrogator

   1) generates a random number RNi,

2) computes SRNi = (RNi + $O_n$) ⊕ PSK, where $O_n$ = 5555 5555 5555 5555h, and

3) sends the authenticate command to the tag (see Table E.3).

b) On receipt of the authenticate command, the tag

1) computes RNi = (SRNi ⊕ PSK) – $O_n$,

2) computes SORNi = (PSK′ + $O_n$) ⊕ RNi′, where PSK′ and RNi′ means bit-wise ROTATE PSK and RNi left for n bits, respectively (n is the number of binary value 1 of RNi),

3) generates a random number RNt, then computes SRNt = (RNt + $O_n$) ⊕ PSK, and

4) returns the response to the interrogator (see Table E.4).

c) On receipt of the response from the tag, the interrogator

1) computes the value of (SORNi ⊕ RNi′),

2) computes the value of (PSK′ + $O_n$),

3) compares the value of (PSK′ + $O_n$) with (SORNi ⊕ RNi′). If this validation fails, the authentication procedure is failed. Otherwise, the tag is legal. Proceeds to the next step,

4) computes RNt = (SRNt ⊕ PSK) – $O_n$, and

5) computes SORNt = (PSK′ + $O_n$) ⊕ RNt′, where RNt′ means bit-wise ROTATE RNt left for n bits (n is the number of binary value 1 of RNt), and

6) sends the authenticate command to the tag (see Table E.5).

d) On receipt of the authenticate command, the tag

1) computes the value of (SORNt ⊕ RNt′),

2) computes the value of (PSK′ + $O_n$),

3) compares the value of (PSK′ + $O_n$) with (SORNt ⊕ RNt′). If this validation fails, the authentication procedure is failed. Otherwise, the interrogator is legal. Proceeds to the next step, and

4) returns the response to the interrogator(see Table E.6).

#### 9.2.2.2 Interrogator authentication

The interrogator authentication procedure is as follows.

a) The interrogator sends the authenticate command to the tag (see Table E.7).

b) On receipt of the authenticate command, the tag

1) generates a random number RNt,

2) computes SRNt = (RNt + $O_n$) ⊕ PSK, where $O_n$=5555 5555 5555 5555h, and

3) returns the response to the interrogator (see Table E.8).

c) On receipt of the response from the tag, the interrogator

1) computes RNt = (SRNt ⊕ PSK) – $O_n$,

2) computes SORNt = (PSK′ + $O_n$) ⊕ RNt′, where PSK′ and RNt′ means bit-wise ROTATE PSK and RNt left for n bits, respectively (n is the number of binary value 1 of RNt), and

3) sends the authenticate command to the tag (see Table E.9).

d) On receipt of the authenticate command, the tag

1) computes the value of (SORNt ⊕ RNt′),

2) compares the value of (PSK′ + $O_n$ ) with (SORNt ⊕ RNt′). If this validation fails, the authentication procedure is failed. Otherwise, the interrogator is legal. Proceeds to the next step, and

3) returns the response to the interrogator(see Table E.10).

#### 9.2.2.3 Tag authentication

The tag authentication procedure is as follows.

a) The interrogator

1) generates a random number RNi,

2) computes SRNi = (RNi + $O_n$) ⊕ PSK, where $O_n$=5555 5555 5555 5555h, and

3) sends the authenticate command to the tag (see Table E.11).

b) On receipt of the authenticate command, the tag

1) computes RNi = (SRNi ⊕ PSK) – $O_n$,

2) computes SORNi = (PSK′ + $O_n$) ⊕ RNi′, where PSK′ and RNi′ means bit-wise ROTATE PSK and RNi left for n bits, respectively (n is the number of binary value 1 of RNi), and

3) returns the response to the interrogator (see Table E.12).

c) On receipt of the response from the tag, the interrogator

1) computes the value of (SORNi ⊕ RNi′), and

2) compares the value of (PSK′ + $O_n$) with (SORNi ⊕ RNi′). If this validation fails, the authentication procedure is failed. Otherwise, the tag is legal.

## 10 Secure communication (optional)

Interrogators and tags may implement the Secure Communication (SecureComm) command. Figure 4 shows a representative procedure for an interrogator sending or receiving data using secure communications. For an interrogator or a tag that supports secure communication, the session key (SK) used to encrypt the message shall be generated by both a tag and interrogator as following after the successful authentication procedure: SK = RNt ⊕ RNi ⊕ PSK. The SecureComm command provides an encrypted payload message generated by SK ⊕ Command as shown in Table E.13 which encapsulates another command intended for the tag to execute. The encrypted payload message shall be decrypted by SK ⊕ Message after the tag received the SecureComm command. The SecureComm reply as shown in Table E.14 provides an encrypted payload response generated by SK ⊕ Reply which encapsulates the reply from the tag regarding the execution of the encapsulated command. The encrypted payload response shall be decrypted by SK ⊕ Response after the interrogator received the SecureComm reply.

To ensure that the key is distinct for every encrypted payload message, if the length of the encrypted payload is greater than the length of SK, SK = SK || $SK_1$ || $SK_2$||...|| $SK_X$, where, $SK_1$ means bit-wise ROTATE SK left for n bits, $SK_2$ means bit-wise ROTATE $SK_1$ left for n bits and $SK_X$ means bit-wise ROTATE $SK_{x-1}$ left for n bits (n is the number of binary value 1 of SK). The length of SK shall be the same as the length of the encrypted payload.

The formats of SecureComm and response commands are shown in Table E.13 and Table E.14, respectively.

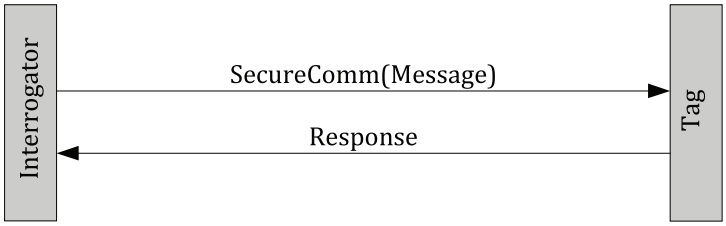


**Figure 4 — Secure communication**

## 11 Key update (optional)

Figure 5 shows a representative procedure for an interrogator update a key with tag, it provides a means for an authenticated interrogator to modify a key according to the security concern. The formats of KeyUpdate and response commands are shown in Table E.15 and Table E.16, respectively.

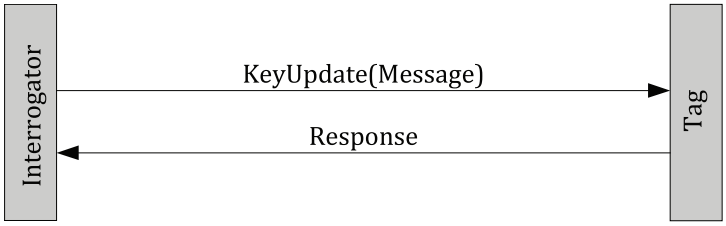


**Figure 5 — Key update**

Upon receiving a valid KeyUpdate command, a tag shall overwrite its old key with the new key. If the tag does not write the new key successfully, then it shall revert completely to the prior stored key.

# Annex A
(normative)

# State transition tables

## A.1 Initial state transition table

**Table A.1 — Initial state transition table**

| Command | Conditions | Next state |
|---|---|---|
| Tag Authentication Step 1 Message | All | Initial |
| Interrogator Authentication Step 1 Message | Success | Interrogator Authentication |
| Mutual Authentication Step 1 Message | Success | Mutual Authentication |

## A.2 Interrogator authentication state transition table

**Table A.2 — Interrogator authentication state transition table**

| Command | Conditions | Next state |
|---|---|---|
| Interrogator Authentication Step 1 Message | All | Interrogator Authentication |
| Interrogator Authentication Step 2 Message | Fail | Initial |

## A.3 Mutual authentication state transition table

**Table A.3 — Mutual authentication state transition table**

| Command | Conditions | Next state |
|---|---|---|
| Mutual Authentication Step 1 Message | All | Mutual Authentication |
| Mutual Authentication Step 2 Message | Fail | Initial |
| Mutual Authentication Step 2 Message | Success | SecureComm |

## A.4 SecureComm state transition table

**Table A.4 — SecureComm state transition table**

| Command | Conditions | Next state |
|---|---|---|
| SecureComm | All | SecureComm |

# Annex B
(normative)

# Error codes and error handling

## B.1 Error code format

**Table B.1 — Error code format**

| Error code | Error subcode |
|---|---|
| 8-bit | 8-bit |

## B.2 Error type and error subcode

**Table B.2 — Error type and error subcode**

| Error type code | Description | Error subcode | Description |
|---|---|---|---|
| $01_h$ | Authentication failed | $01_h$ | The authentication was failed. |
| $02_h$ | Secure communication failed | $01_h$ | The secure communication between interrogator and tag was failed. |

# Annex C
(informative)

# Cipher Description

## C.1 General

The logical operation of eXclusive OR (XOR) outputs true whenever both inputs differ (one is true, the other is false). Table C.1 of A XOR B shows that it outputs true whenever the inputs differ.

**Table C.1 — XOR true table**

| Input | | Output |
|---|---|---|
| A | B | |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## C.2 Symbols and programs

It is usually symbolized by the infix operators XOR and ⊕. In several programming languages, such as C, C++, C#, Java, Perl, MATLAB and Python, a caret (^) is used to denote the bitwise XOR operator. This is not used outside of programming contexts because it is too easily confused with other uses of the caret. To encrypt a single character you can use char x = x ^ key; if you have a key of one byte. To encrypt a string of characters with a longer key, you can use something akin to the following code. The program encrypts each character in the string using the ^ bit operator to XOR, the string value with the key value for each character.

```
#include <iostream.h>

int main()

{

  char string[11]="A nice cat";

  char key[11]="ABCDEFGHIJ";

  for(int x=0; x<10; x++)

  {

   string[x]=string[x]^key[x];

   cout<<string[x];

  }

  return 0;

}
```

# Annex D
(informative)

# Test vectors

## D.1 Mutual authentication

The following example uses a 64-bit key and a 64-bit random number.

| | | |
|---|---|---|
| PSK | d4f625e4 | 122688af |
| RNi | 1ba58677 | 7e45a0e7 |
| RNt | 680e9b5f | 5d7508a1 |
| | | |
| SRNi | a40cfe28 | c1bc7d93 |
| SORNi | 204bb61f | 654744d0 |
| SRNt | 6995d550 | a0ecd559 |
| SORNt | 7764b3d1 | 3e5493a1 |

## D.2 Interrogator authentication

The following example uses a 64-bit key and a 64-bit random number.

| | | |
|---|---|---|
| PSK | d4f625e4 | 122688af |
| RNt | 680e9b5f | 5d7508a1 |
| SRNt | 6995d550 | a0ecd559 |
| SORNt | 7764b3d1 | 3e5493a1 |

## D.3 Tag authentication

The following example uses a 64-bit key and a 64-bit random number.

| | | |
|---|---|---|
| PSK | d4f625e4 | 122688af |
| RNi | 1ba58677 | 7e45a0e7 |
| SRNi | a40cfe28 | c1bc7d93 |
| SORNi | 204bb61f | 654744d0 |

# Annex E
(normative)

# Protocol specific

## E.1 ISO/IEC 18000-63

This subclause defines protocol specific information for ISO/IEC 18000-63.

For ISO/IEC 18000-63, key properties for encryption keys shall be manufacturer-defined.

## E.2 Command

### E.2.1 Authenticate command

Interrogators and tags shall implement the Authenticate command as shown in Table E.1. The fast response reply to an Authenticate command is shown in Table E.2. An interrogator shall use Authenticate commands to perform mutual authentication. The CSI specified in the Authenticate command selects a particular coding suite from among those supported by the tag. The number and encoding of the Authenticate commands required to implement the authentication depends on the chosen coding suite.

**Table E.1 — Authenticate command**

| | Command | RFU | CSI | Length | Message | RN | CRC-16 |
|---|---|---|---|---|---|---|---|
| **# of bits** | 8 | 8 | 8 | EBV | Variable | 16 | 16 |
| **description** | Command code | 00h | CSI | length of message | message (depends on CSI) | handle | CRC-16 |

**Table E.2 — Fast response reply to an Authenticate command**

| | Header | Length | Response | RN | CRC-16 |
|---|---|---|---|---|---|
| **# of bits** | 1 | EBV | Variable | 16 | 16 |
| **description** | 0 | length of response | response (depends on CSI) | handle | CRC-16 |

### E.2.2 Mutual authentication

#### E.2.2.1 Step 1

The message of Authenticate command and the response of mutual authentication step 1 are as shown in Table E.3 and Table E.4, respectively.

**Table E.3 — Message of Authenticate command of mutual authentication step 1**

| | Message | | | |
|---|---|---|---|---|
| | **AuthType** | **AuthStep** | **Key ID** | **AuthData** |
| **# of bits** | 2 | 3 | 5 | variable |
| **description** | 00 | 001 | key identifier | SRNi |

**Table E.4 — Response of Authenticate command of mutual authentication step 1**

| | Response |
|---|---|
| | AuthData |
| **# of bits** | variable |
| **description** | SORNi \|\| SRNt |

### E.2.2.2 Step 2

The message of Authenticate command and the response of mutual authentication step 2 are as shown in Table E.5 and Table E.6, respectively.

**Table E.5 — Message of Authenticate command of mutual authentication step 2**

| | Message | | | |
|---|---|---|---|---|
| | AuthType | AuthStep | Key ID | AuthData |
| **# of bits** | 2 | 3 | 5 | variable |
| **description** | 00 | 010 | key identifier | SORNt |

**Table E.6 — Response of Authenticate command of mutual authentication step 2**

| | Response |
|---|---|
| | AuthData |
| **# of bits** | 0 |
| **description** | NULL |

## E.2.3 Interrogator authentication

### E.2.3.1 Step 1

The message of Authenticate command and the response of interrogator authentication step 1 are as shown in Table E.7 and Table E.8, respectively.

**Table E.7 — Message of Authenticate command of interrogator authentication step 1**

| | Message | | | |
|---|---|---|---|---|
| | AuthType | AuthStep | Key ID | AuthData |
| **# of bits** | 2 | 3 | 5 | Variable |
| **description** | 01 | 001 | key identifier | NULL |

**Table E.8 — Response of Authenticate command of interrogator authentication step 1**

| | Response |
|---|---|
| | AuthData |
| **# of bits** | variable |
| **description** | SRNt |

### E.2.3.2 Step 2

The message of Authenticate command and the response of interrogator authentication step 2 are as shown in Table E.9 and Table E.10, respectively.

**Table E.9 — Message of Authenticate command of interrogator authentication step 2**

| | Message | | | |
|---|---|---|---|---|
| | **AuthType** | **AuthStep** | **Key ID** | **AuthData** |
| **# of bits** | 2 | 3 | 5 | variable |
| **description** | 1 | 010 | key identifier | SORNt |

**Table E.10 — Response of Authenticate command of interrogator authentication step 2**

| | Response |
|---|---|
| | **AuthData** |
| **# of bits** | 0 |
| **description** | NULL |

### E.2.4 Tag authentication

The message of Authenticate command and the response of tag authentication step 1 are as shown in Table E.11 and Table E.12, respectively.

**Table E.11 — Message of Authenticate command of tag authentication step 1**

| | Message | | | |
|---|---|---|---|---|
| | **AuthType** | **AuthStep** | **Key ID** | **AuthData** |
| **# of bits** | 3 | 3 | 5 | variable |
| **description** | 10 | 001 | key identifier 0 | SRNi |

**Table E.12 — Response of Authenticate command of tag authentication step 1**

| | Response |
|---|---|
| | **AuthData** |
| **# of bits** | variable |
| **description** | SORNi |

### E.2.5 Secure communication

After a tag has cryptographically authenticated an interrogator, it may accept subsequent commands encapsulated in the SecureComm command shown in Table E.13. The message field in a SecureComm may encapsulate an encrypted tag-supported command. Before encapsulating a command an interrogator shall remove the preamble, handle and CRC from the command. An interrogator does not include a MAC in the SecureComm.

**Table E.13 — SecureComm command**

| | Command | CSI | Length | Message | RN | CRC-16 |
|---|---|---|---|---|---|---|
| **# of bits** | 8 | 8 | EBV | Variable | 16 | 16 |
| **description** | command code | CSI | length of message | encrypted message (depends on CSI) | handle | CRC-16 |

The response reply to a SecureComm command is shown in Table E.14.

**Table E.14 — Response reply to a SecureComm command**

| | Header | Length | Response | RN | CRC-16 |
|---|---|---|---|---|---|
| **# of bits** | 1 | EBV | Variable | 16 | 16 |
| **description** | 0 | length of response | encrypted response (depends on CSI) | handle | CRC-16 |

SecureComm allows an interrogator to read data from and write data to a tag while preventing an eavesdropper from decrypting the communications. It allows configuring a tag's secure features by writing to a memory location that configures these features. The authenticate communication is not supported.

### E.2.6 Key update

Interrogators and tags may implement the KeyUpdate command; if they do, they shall implement it as shown in Table E.15. KeyUpdate allows an interrogator to overwrite a key stored in a tag. A KeyUpdate command shall always be encapsulated in a SecureComm.

**Table E.15 — KeyUpdate command**

| | Command | RFU | Key ID | Length | Message |
|---|---|---|---|---|---|
| **# of bits** | 8 | 8 | 5 | EBV | Variable |
| **description** | command code | | key identifier | length of Key | Key |

The response reply to an KeyUpdate command is shown in Table E.16.

**Table E.16 — Response reply to an KeyUpdate command**

| | Header | RN | CRC-16 |
|---|---|---|---|
| **# of bits** | 1 | 16 | 16 |
| **description** | 0 | handle | CRC-16 |

# Annex F
(normative)

# Authentication procedure pseudo-code

## F.1 Mutual authentication: Interrogator

```
begin
    Input On,PSK
    RNr ←GenerateRandom()
    Register1←RNr+On
    Register2←Register1^PSK
    SendToTag(Register2)
    Register2, Register3←ReceiveFromTag()
    Register1←Register1- On
    Counter←GetBit1Counts(Register1)
    Register4←PSK
    while Counter>0 do
        begin
            Register1←Register1<<<1
            Register4←Register4<<<1
            Counter←Counter-1
        end
    Register4←Register4+On
    Register1←Register4^Register1
    If(Register2!=Register1)
        begin
            Failure!
        end
    else
        begin
            Register1←Register3^PSK
        Register1←Register1- On
        Counter←GetBit1Counts(Register1)
        while Counter>0 do
            begin
```

```
                Register1←Register1<<<1
                PSK ←PSK<<<1
                Counter←Counter-1
            end
        Register2←PSK+On
        Register1←Register2^Register1
        SendToTag(Register1)
        Success!
        end
end
```

## F.2 Mutual authentication: Tag

```
begin
    Input On,PSK
    Register1←ReceiveFromReader()
    Register1←Register1^PSK
    Register1←Register1-On
    Counter←GetBit1Counts(Register1)
    Register2←PSK
    while Counter>0 do
        begin
            Register1←Register1<<<1
            Register2←Register2<<<1
            Counter←Counter-1
        end
    Register2←Register2+On
    Register1←Register2^ Register1
    RNt ←GenerateRandom()
    Register2←RNt+On
    Register3←Register2^PSK
    SendToReader(Register1,Register3)
    Register1←ReceiveFromReader()
    Register2←Register2-On
    Counter←GetBit1Counts(Register2)
    while Counter>0 do
        begin
```

```
            Register2←Register2<<<1
            PSK←PSK<<<1
            Counter←Counter-1
        end
    Register3←PSK+On
    Register2←Register3^Register2
    If(Register1!=Register2)
        begin
            Failure!
        end
    else
        begin
            Success!
        end
end
```

# Annex G
(informative)

# Security considerations

The complexity of the proposed scheme in this coding suite is very low. With reasonable effort and using state of the art computers, it is possible to develop software to crack the XOR related mechanisms. For example, an adversary could provide challenges to a tag and collect the responses and would be able to reveal the PSK with some effort and with some attempts of crypto analysis, it could be broken in the sense of allowing practical key recovery by passive attacks, as well if challenges or "securely" transmitted plain texts show any statistically detectable deviation from being uncorrelated and uniformly random distributed.

For above security issues, this document should call attention to users that the use of this mechanism in the high security requirement environment is very strongly discouraged. Users should carefully weigh before using the proposed scheme, as it is sole responsibility of the standard user and nobody else takes responsibility for any possible unintended negative consequences. Manufacturers should provide an obvious mark on their product which implemented the protocol specified in this document about the potential security risk while using it.

# Bibliography

[1] ISO/IEC 18000-2, *Information technology — Radio frequency identification for item management — Part 2: Parameters for air interface communications below 135 kHz*

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

[3] ISO/IEC 18000-4, *Information technology — Radio frequency identification for item management — Part 4: Parameters for air interface communications at 2.45 GHz*

[4] ISO/IEC 18000-7, *Information technology — Radio frequency identification for item management — Part 7: Parameters for active air interface communications at 433 MHz*

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

[6] Barrett R.B., & Stenner A.J. The myth of the exclusive 'or'. Mind. 1971, **80** (317) pp. 116–121

[7] http://www.cprogramming.com/tutorial/xor.html

**ICS 35.040.50**

Price based on 22 pages