



The Global Language of Business

EPC™ Radio-Frequency Identity Protocols Generation-2 UHF RFID Standard

Specification for RFID Air Interface Protocol for
Communications at 860 MHz – 960 MHz

Release 2.1, Ratified, Jul 2018



Document Summary

| Document Item | Current Value |
|----------------------|---|
| Document Name | EPC™ Radio-Frequency Identity Protocols Generation-2 UHF RFID Standard |
| Document Date | Jul 2018 |
| Document Version | 2.1 |
| Document Issue | |
| Document Status | Ratified |
| Document Description | Specification for RFID Air Interface Protocol for Communications at 860 MHz – 960 MHz |

Log of Changes

| Release | Date of Change | Changed By | Summary of Change |
|---------------|----------------|---|--|
| Version 1.0.4 | Sept 8, 2004 | | Modified Chicago protocol V1.0.3 as per August 17, 2004 "combo" CRC change template. |
| Version 1.0.5 | Sept 14, 2004 | | Modified Gen2 protocol V1.0.4 as per September 10, 2004 CRC review. |
| Version 1.0.6 | Sept 17, 2004 | | Modified Gen2 protocol V1.0.5 as per September 17, 2004 HAG review. |
| Version 1.0.7 | Sept 24, 2004 | | Modified Gen2 protocol V1.0.6 as per September 21, 2004 CRC review to fix errata. Changed OID to EPC. |
| Version 1.0.8 | Dec 11, 2004 | | Modified Gen2 protocol V1.0.7 as per the V1.0.7 errata. |
| Version 1.0.9 | Jan 26, 2005 | | Modified Gen2 protocol V1.0.8 as per the V1.0.8 errata and AFI enhancement requests. |
| Version 1.1.0 | Dec 1, 2005 | | Harmonized Gen2 protocol V1.0.9 with the ISO 18000-6 Type C amendment. |
| Version 1.2.0 | May 11, 2008 | | Modified Gen2 protocol V1.1.0 to satisfy the ILT JRG requirements V1.2.3. |
| 2.0.0 | Oct 2013 | | Modified Gen2 protocol V1.2.0 to satisfy EAS JRG requirements V0.8, TA JRG requirements V0.7, and CE JRG requirements V1.5.4. |
| 2.0.1 | Apr 2015 | | Modified Gen2 protocol V2.0.0 to fix errata |
| 2.1 | Jul 2018 | D.Buckley (branding), D.Mullen (content) | Update to new GS1 branding and WR 17-223: Modified Gen2 to harmonize with ISO 18000-63 clarifications (<i>briefly, and incorrectly, published as version 3.0</i>). |

Disclaimer

GS1®, under its IP Policy, seeks to avoid uncertainty regarding intellectual property claims by requiring the participants in the Work Group that developed this **EPC™ Radio-Frequency Identity Protocols Generation-2 UHF RFID Standard** to agree to grant to GS1 members a royalty-free licence or a RAND licence to Necessary Claims, as that term is defined in the GS1 IP Policy. Furthermore, attention is drawn to the possibility that an implementation of one or more features of this Specification may be the subject of a patent or other intellectual property right that does not involve a Necessary Claim. Any such patent or other intellectual property right is not subject to the licencing obligations of GS1. Moreover, the agreement to grant licences provided under the GS1 IP Policy does not include IP rights and any claims of third parties who were not participants in the Work Group.



Accordingly, GS1 recommends that any organisation developing an implementation designed to be in conformance with this Specification should determine whether there are any patents that may encompass a specific implementation that the organisation is developing in compliance with the Specification and whether a licence under a patent or other intellectual property right is needed. Such a determination of a need for licencing should be made in view of the details of the specific system designed by the organisation in consultation with their own patent counsel.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR PARTICULAR PURPOSE, OR ANY WARRANTY OTHER WISE ARISING OUT OF THIS SPECIFICATION. GS1 disclaims all liability for any damages arising from use or misuse of this Standard, whether special, indirect, consequential, or compensatory damages, and including liability for infringement of any intellectual property rights, relating to use of information in or reliance upon this document.

GS1 retains the right to make changes to this document at any time, without notice. GS1 makes no warranty for the use of this document and assumes no responsibility for any errors which may appear in the document, nor does it make a commitment to update the information contained herein.

GS1 and the GS1 logo are registered trademarks of GS1 AISBL.



Table of Contents

- Introduction..... 7**
- 1. Scope..... 8**
- 2. Conformance 8**
 - 2.1 Claiming conformance 8
 - 2.2 General conformance requirements 8
 - 2.2.1 Interrogators..... 8
 - 2.2.2 Tags..... 8
 - 2.3 Command structure and extensibility 9
 - 2.3.1 Mandatory commands 9
 - 2.3.2 Optional commands 9
 - 2.3.3 Proprietary commands..... 9
 - 2.3.4 Custom commands 9
 - 2.4 Reserved for Future Use (RFU)..... 9
 - 2.5 Cryptographic Suite Indicators..... 9
- 3. Normative references 10**
- 4. Terms and definitions 10**
 - 4.1 Additional terms and definitions 10
- 5. Symbols, abbreviated terms, and notation 14**
 - 5.1 Symbols..... 15
 - 5.2 Abbreviated terms..... 15
 - 5.3 Notation..... 16
- 6. Protocol requirements 17**
 - 6.1 Protocol overview..... 17
 - 6.1.1 Physical layer 17
 - 6.1.2 Tag-identification layer 17
 - 6.2 Protocol parameters 17
 - 6.2.1 Signaling – Physical and media access control parameters 17
 - 6.2.2 Logical – Operating procedure parameters 20
 - 6.3 Description of operating procedure..... 21
 - 6.3.1 Physical interface..... 21
 - 6.3.2 Logical interface 41
- 7. Intellectual property rights policy and disclaimer..... 111**
- Annex A (normative) Extensible bit vectors (EBV) 112**
- Annex B (normative) State-transition tables..... 113**
 - B.1 Present state: Ready113
 - B.2 Present state: Arbitrate.....115
 - B.3 Present state: Reply117
 - B.4 Present state: Acknowledged.....119
 - B.5 Present state: Open 121

B.6 Present state: Secured124
 B.7 Present state: Killed127

Annex C (normative) Command-Response Tables..... 128

C.1 Command response: Power-up128
 C.2 Command response: Query128
 C.3 Command response: QueryRep.....129
 C.4 Command response: QueryAdjust129
 C.5 Command response: ACK.....130
 C.6 Command response: NAK.....130
 C.7 Command response: Req_RN130
 C.8 Command response: Select131
 C.9 Command response: Read131
 C.10 Command response: Write131
 C.11 Command response: Kill132
 C.12 Command response: Lock132
 C.13 Command response: Access133
 C.14 Command response: BlockWrite.....133
 C.15 Command response: BlockErase133
 C.16 Command response: BlockPermalock.....134
 C.17 Command response: Challenge.....134
 C.18 Command response: Authenticate.....134
 C.19 Command response: AuthComm135
 C.20 Command response: SecureComm.....135
 C.21 Command response: ReadBuffer136
 C.22 Command response: KeyUpdate136
 C.23 Command response: Untraceable.....136
 C.24 Command response: FileSetup.....137
 C.25 Command response: FileOpen137
 C.26 Command response: FilePrivilege.....137
 C.27 Command response: TagPrivilege.....138
 C.28 Command response: FileList138
 C.29 Command response: T₂ timeout.....138
 C.30 Command response: Faulty command139

Annex D (informative) Example slot-count (Q) selection algorithm..... 141

D.1 Example algorithm an Interrogator might use to choose Q141

Annex E (informative) Example Tag inventory and access 142

E.1 Example inventory and access of a single Tag.....142

Annex F (informative) Calculation of 5-bit and 16-bit cyclic redundancy checks 143

F.1 Example CRC-5 encoder/decoder143
 F.2 Example CRC-16 encoder/decoder.....143
 F.3 Example CRC-16 calculations.....144

Annex G (Normative) Multiple- and dense-Interrogator channelized signaling146

G.1 Overview of dense-Interrogator channelized signaling (informative)146



Annex H (informative) Interrogator-to-Tag link modulation 149
H.1 Baseband waveforms, modulated RF, and detected waveforms 149

Annex I (Normative) Error codes 150
I.1 Tag error codes and their usage 150

Annex J (normative) Slot counter 151
J.1 Slot-counter operation 151

Annex K (informative) Example data-flow exchange 152
K.1 Overview of the data-flow exchange 152
K.2 Tag memory contents and lock-field values 152
K.3 Data-flow exchange and command sequence 153

Annex L (informative) Optional Tag Features 154
L.1 Optional Tag memory banks, memory-bank sizes, and files 154
L.2 Optional Tag commands 154
L.3 Optional Tag passwords, security, and keys 155
L.4 Optional Tag replies 155
L.5 Optional Tag PC and XPC bit designations and values 155
L.6 Optional Tag error-code reporting format 155
L.7 Optional Tag backscatter modulation format 155
L.8 Optional Tag functionality 155

Annex M (informative) Cryptographic-Suite Checklist 156

Annex N (normative) Application Conformance 157

Introduction

This protocol defines the physical and logical requirements for a passive-backscatter, Interrogator-talks-first (ITF), radio-frequency identification (RFID) system operating in the 860 MHz – 960 MHz frequency range. The system comprises Interrogators, also known as Readers, and Tags, also known as Labels or Transponders.

An Interrogator transmits information to a Tag by modulating an RF signal in the 860 MHz – 960 MHz frequency range. The Tag receives both information and operating energy from this RF signal. Tags are passive, meaning that they receive all of their operating energy from the Interrogator's RF signal.

An Interrogator receives information from a Tag by transmitting a continuous-wave (CW) RF signal to the Tag; the Tag responds by modulating the reflection coefficient of its antenna, thereby backscattering an information signal to the Interrogator. The system is ITF, meaning that a Tag modulates its antenna reflection coefficient with an information signal only after being directed to do so by an Interrogator.

Interrogators and Tags are not required to talk simultaneously; rather, communications are half-duplex, meaning that Interrogators talk and Tags listen, or vice versa.

1. Scope

This protocol specifies:

- Physical interactions (the signaling layer of the communication link) between Interrogators and Tags, and
- Logical operating procedures and commands between Interrogators and Tags.

2. Conformance

2.1 Claiming conformance

A device shall not claim conformance with this protocol unless the device complies with

- a. a) all clauses in this protocol (except those marked as optional), and
- b. b) the conformance document associated with this protocol, and,
- c. c) all local radio regulations.

To be certified as alteration-EAS, Tag-alteration, and/or consumer-electronics conformant, Tags and Interrogators shall additionally support the optional clauses or portions of optional clauses specified in [Annex N](#).

Conformance may also require a license from the owner of any intellectual property utilized by the device.

2.2 General conformance requirements

2.2.1 Interrogators

To conform to this protocol, an Interrogator shall:

- Meet the requirements of this protocol,
- Implement the mandatory commands defined in this protocol,
- Modulate/transmit and receive/demodulate a sufficient set of the electrical signals defined in the signaling layer of this protocol to communicate with conformant Tags, and
- Conform to all local radio regulations.

To conform to this protocol, an Interrogator may:

- Implement any subset of the optional commands defined in this protocol, and
- Implement any proprietary and/or custom commands in conformance with this protocol.

To conform to this protocol, an Interrogator shall not:

- Implement any command that conflicts with this protocol, or
- Require using an optional, proprietary, or custom command to meet the requirements of this protocol.

2.2.2 Tags

To conform to this protocol, a Tag shall:

- Meet the requirements of this protocol,
- Implement the mandatory commands defined in this protocol,
- Modulate a backscatter signal only after receiving the requisite command from an Interrogator, and
- Conform to all local radio regulations.

To conform to this protocol, a Tag may:

- Implement any subset of the optional commands defined in this protocol, and

- Implement any proprietary and/or custom commands as defined in [2.3.3](#) and [2.3.4](#), respectively.

To conform to this protocol, a Tag shall not:

- Implement any command that conflicts with this protocol,
- Require using an optional, proprietary, or custom command to meet the requirements of this protocol, or
- Modulate a backscatter signal unless commanded to do so by an Interrogator using the signaling layer defined in this protocol.

2.3 Command structure and extensibility

This protocol allows four command types: (1) mandatory, (2) optional, (3) proprietary, and (4) custom. Subclause [6.3.2.12](#) and [Table 6-28](#) define the structure of the command codes used by Interrogators and Tags for each of the four types, as well as the availability of future extensions. All commands defined by this protocol are either mandatory or optional. Proprietary or custom commands are manufacturer-defined.

2.3.1 Mandatory commands

Conforming Tags shall support all mandatory commands. Conforming Interrogators shall support all mandatory commands.

2.3.2 Optional commands

Conforming Tags may or may not support optional commands. Conforming Interrogators may or may not support optional commands. If a Tag or an Interrogator implements an optional command then it shall implement it in the manner specified in this protocol.

2.3.3 Proprietary commands

Proprietary commands may be enabled in conformance with this protocol, but are not specified herein. All proprietary commands shall be capable of being permanently disabled. Proprietary commands are intended for manufacturing purposes and shall not be used in field-deployed RFID systems.

2.3.4 Custom commands

Custom commands may be enabled in conformance with this protocol, but are not specified herein. An Interrogator shall issue a custom command only after (1) singulating a Tag, and (2) reading (or having prior knowledge of) the Tag manufacturer's identification in the Tag's TID memory. An Interrogator shall use a custom command only in accordance with the specifications of the Tag manufacturer identified in the TID. A custom command shall not solely duplicate the functionality of any mandatory or optional command defined in this protocol by a different method.

2.4 Reserved for Future Use (RFU)

This protocol denotes some Tag memory addresses, Interrogator command codes, and bit fields within Interrogator commands as RFU. GS1 EPCglobal is reserving these RFU values for future extensibility. Under some circumstances GS1 EPCglobal may permit another standards body or organization to use one or more of these RFU values for standardization purposes. In such circumstances the permitted body shall keep GS1 EPCglobal apprised, in a timely manner, of its use or potential use of these RFU values. Third parties, including but not limited to solution providers and end users, shall not use these RFU values for proprietary purposes.

2.5 Cryptographic Suite Indicators

A Tag may support one or more cryptographic suites. The *Challenge* and *Authenticate* commands include a CSI field that specifies a single cryptographic suite. CSI is an 8-bit field with bit values defined below.

- Four most-significant bits: Cryptographic suite assigning authority, as follows:

- 0000₂ – 0011₂: ISO/IEC 29167
- 0100₂ – 1100₂: RFU
- 1101₂: Tag manufacturer
- 1110₂: GS1
- 1111₂: RFU
- Four least-significant bits: One of 16 cryptographic suites that the assigning authority may assign.

Example: CSI=00000000₂ is the first and CSI=00000001₂ is the second suite that ISO/IEC 29167 may assign.

3. Normative references

The following referenced documents are indispensable to the application of this protocol. For dated references, only the edition cited applies. For undated references, the latest edition (including any amendments) applies.

GS1 EPCglobal™: GS1 EPC™ Tag Data Standard

ISO/IEC 15961: *Information technology — Radio frequency identification (RFID) for item management — Data protocol: application interface*

ISO/IEC 15962: *Information technology — Radio frequency identification (RFID) for item management — Data protocol: data encoding rules and logical memory functions*

ISO/IEC 15963: *Information technology — Radio frequency identification for item management — Unique identification for RF tags*

ISO/IEC 18000-1: *Information technology — Radio frequency identification for item management — Part 1: Reference architecture and definition of parameters to be standardized*

ISO/IEC 18000-63: *Information technology automatic identification and data capture techniques — Radio frequency identification for item management air interface — Part 63: Parameters for air interface communications at 860–960 MHz*

ISO/IEC 19762: *Information technology AIDC techniques – Harmonized vocabulary – Part 3: radio-frequency identification (RFID)*

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

4. Terms and definitions

The principal terms and definitions used in this protocol are described in ISO/IEC 19762.

4.1 Additional terms and definitions

Terms and definitions specific to this protocol that supersede any normative references are as follows:

Air interface

The complete communication link between an Interrogator and a Tag including the physical layer, collision-arbitration algorithm, command and response structure, and data-coding methodology.

Asymmetric key pair

A private key and its corresponding public key, used in conjunction with an asymmetric cryptographic suite.

Authentication

The process of determining whether an entity or data is/are who or what, respectively, it claims to be. The types of entity authentication referred-to in this protocol are Tag authentication, Interrogator authentication, and Tag-Interrogator mutual authentication. For data

authentication see authenticated communications.

Authenticated communications

Communications in which message integrity is protected.

Command set

The set of commands used to inventory and interact with a Tag population.

Continuous wave

Typically a sinusoid at a given frequency, but more generally any Interrogator waveform suitable for powering a passive Tag without amplitude and/or phase modulation of sufficient magnitude to be interpreted by a Tag as transmitted data.

Cover coding

A method by which an Interrogator obscures information that it is transmitting to a Tag. To cover-code data or a password, an Interrogator first requests a random number from the Tag. The Interrogator then performs a bit-wise EXOR of the data or password with this random number, and transmits the cover-coded string to the Tag. The Tag uncovers the data or password by performing a bit-wise EXOR of the received cover-coded string with the original random number.

Crypto superuser

A key with an asserted CryptoSuperuser privilege.

Data element

A low-level, indivisible data construct. See file and record.

Dense-Interrogator environment

An operating environment (defined below) within which most or all of the available channels are occupied by active Interrogators (for example, 25 active Interrogators operating in 25 available channels).

Dense-Interrogator mode

A set of Interrogator-to-Tag and Tag-to-Interrogator signaling parameters used in dense-Interrogator environments.

Extended Tag identifier (XTID)

A memory construct that defines a Tag's capabilities and may include a Tag serial number, further specified in the *GS1 EPC Tag Data Standard*.

Extended temperature range

–40 °C to +65 °C (see nominal temperature range).

File type

An 8-bit string that specifies a file's designated type.

File

A set of one or more records accessed as a unit (see record and data element).

File superuser

An access password or key with a 0011₂ **secured**-state file privilege value

Full-duplex communications

A communications channel that carries data in both directions at once. See half-duplex communications.

GS1 EPCglobal™ Application

An application whose usage denotes an acceptance of GS1 EPCglobal™ standards and policies (see non-GS1 EPCglobal™ Application).

Half-duplex communications

A communications channel that carries data in one direction at a time rather than in both directions at once. See full-duplex communications.

Insecure communications

Communications in which neither message integrity nor message confidentiality are protected.

Interrogator authentication

A means for a Tag to determine, via cryptographic means, that an Interrogator's identity is as claimed.

Inventoried flag

A flag that indicates whether a Tag may respond to an Interrogator. Tags maintain a separate **inventoried** flag for each of four sessions; each flag has symmetric *A* and *B* values. Within any given session, Interrogators typically inventory Tags from *A* to *B* followed by a re-inventory of Tags from *B* back to *A* (or vice versa).

Inventory round

The period initiated by a *Query* command and terminated by either a subsequent *Query* command (which also starts a new inventory round), a *Select* command, or a *Challenge* command.

Key

A value used to influence the output of a cryptographic algorithm or cipher.

KeyID

A numerical designator for a single key.

Message authentication code (MAC)

A code, computed over bits in a message, that an Interrogator or a Tag may use to verify the integrity of the message.

Multiple-Interrogator environment

An operating environment (defined below) within which a modest number of the available channels are occupied by active Interrogators (for example, 5 active Interrogators operating in 25 available channels).

Mutual authentication

A means for a Tag and an Interrogator to each determine, via cryptographic means, that the others' identity is as claimed.

Nominal temperature range

–25 °C to +40 °C (see extended temperature range).

Non-GS1 EPCglobal™ Application

An application whose usage does not denote an acceptance of GS1 EPCglobal™ standards and policies (see GS1 EPCglobal™ Application).

Nonremovable Tag

A Tag that a consumer cannot physically detach from an item without special equipment or without compromising the item's intended functionality. See removable Tag.

Operating environment

A region within which an Interrogator's RF transmissions are attenuated by less than 90dB. In free space, the operating environment is a sphere whose radius is approximately 1000m, with the Interrogator located at the center. In a building or other enclosure, the size and shape of the operating environment depends on factors such as the material properties and shape of the building, and may be less than 1000m in certain directions and greater than 1000m in other directions.

Operating procedure

Collectively, the set of functions and commands used by an Interrogator to inventory and interact with Tags (also known as the *Tag-identification layer*).

PacketCRC

A 16-bit cyclic-redundancy check (CRC) code that a Tag calculates over its PC, optional XPC word or words, and EPC and backscatters during inventory.

PacketPC

Protocol-control information that a Tag with an asserted **XI** dynamically calculates. See StoredPC.

Passive Tag (or passive Label)

A Tag (or Label) whose transceiver is powered by the RF field.

Password

A secret value sent by an Interrogator to a Tag to enable restricted Tag operations. Passwords are not keys. (Note: The only passwords defined by this protocol are the kill and access passwords).

Permalock or permalocked

A memory location whose lock status is unchangeable (i.e. the memory location is permanently locked or permanently unlocked).

Persistent memory or persistent flag

A memory or flag value whose state is maintained during a brief loss of Tag power.

Physical layer

The data coding and modulation waveforms used in Interrogator-to-Tag and Tag-to-Interrogator signaling.

Private key

The undisclosed or non-distributed key in an asymmetric, or public-private key pair, cipher. A private key is typically used for decryption or digital-signature generation. See public key.

Protocol

Collectively, a physical layer and a Tag-identification layer specification.

Public key

The disclosed or distributed key in an asymmetric, or public-private key pair, cipher. A public key is typically used for encryption or signature verification. See private key.

Q

A parameter that an Interrogator uses to regulate the probability of Tag response. An Interrogator instructs Tags in an inventory round to load a Q -bit random (or pseudo-random) number into their slot counter; the Interrogator may also command Tags to decrement their slot counter. Tags reply when the value in their slot counter (i.e. their slot – see below) is zero. Q is an integer in the range (0,15); the corresponding Tag-response probabilities range from $2^0 = 1$ to $2^{-15} = 0.000031$.

Random-slotted collision arbitration

A collision-arbitration algorithm where Tags load a random (or pseudo-random) number into a slot counter, decrement this slot counter based on Interrogator commands, and reply to the Interrogator when their slot counter reaches zero.

Record

A set of one or more data elements accessed as a unit. See data element and file.

Removable Tag

A Tag that a consumer can physically detach from an item without special equipment and without compromising the item's intended functionality.

Secure communications

Communications in which message confidentiality is protected.

Security

A degree of protection against threats identified in a security policy. A system is secure if it is protected to the degree specified in the security policy. See security policy.

Security policy

A definition, either explicit or implicit, of the threats a system is intended to address. See security.

Session

An inventory process comprising an Interrogator and an associated Tag population. An Interrogator chooses one of four sessions and inventories Tags within that session. The Interrogator and associated Tag population operate in one and only one session for the duration of an inventory round (defined above). For each session, Tags maintain a corresponding **inventoried** flag. Sessions allow Tags to keep track of their inventoried status

separately for each of four possible time-interleaved inventory processes, using an independent **inventoried** flag for each process.

Session key

A temporary key generated by one or both of Tag and Interrogator and typically used for authenticated and/or secure communications.

Single-Interrogator environment

An operating environment (defined above) within which there is a single active Interrogator at any given time.

Singulation

Identifying an individual Tag in a multiple-Tag environment.

Slot

Slot corresponds to the point in an inventory round at which a Tag may respond. Slot is the value output by a Tag's slot counter; Tags reply when their slot (i.e. the value in their slot counter) is zero. See *Q*.

StoredCRC

A 16-bit cyclic-redundancy check (CRC) computed over the StoredPC and the EPC specified by the length (*L*) bits in the StoredPC, and stored in EPC memory.

StoredPC

Protocol-control information stored in EPC memory. See PacketPC.

Symmetric key

A shared key used in conjunction with a symmetric cipher.

Tag authentication

A means for an Interrogator to determine, via cryptographic means, that a Tag's identity is as claimed.

Tag-identification layer

Collectively, the set of functions and commands used by an Interrogator to inventory and interact with Tags (also known as the *operating procedure*).

Tari

Reference time interval for a data-0 in Interrogator-to-Tag signaling. The mnemonic "Tari" derives from the ISO/IEC 18000-6 (part A) specification, in which Tari is an abbreviation for Type A Reference Interval.

Traceable

Not restricting the identifying information a Tag exposes and/or the Tag's operating range. See untraceable.

Untraceable privilege

A privilege given to the access password or to a key that grants an Interrogator using the access password or key the right to access untraceably hidden memory and/or to issue an *Untraceable* command.

Untraceably hidden memory

Memory that an untraceable tag hides from Interrogators with a deasserted untraceable privilege.

Untraceable

Restricting the identifying information a Tag exposes and/or the Tag's operating range. See traceable.

5. Symbols, abbreviated terms, and notation

The principal symbols and abbreviated terms used in this protocol are detailed in ISO/IEC 19762. Symbols, abbreviated terms, and notation specific to this protocol are as follows:

5.1 Symbols

| | |
|--|---|
| BLF | Backscatter-link frequency ($BLF = 1/T_{pri} = DR/TR_{cal}$) |
| C | Computed-response indicator |
| CSI | Cryptographic suite identifier |
| DR | Divide ratio |
| F | File-services indicator (whether a Tag supports the <i>FileOpen</i> command) |
| FrT | Frequency tolerance |
| H | Hazmat indicator |
| K | Killable indicator |
| M | Number of subcarrier cycles per symbol |
| M_h | RF signal envelope ripple (overshoot) |
| M_{hh} | FHSS signal envelope ripple (overshoot) |
| M_l | RF signal envelope ripple (undershoot) |
| M_{hl} | FHSS signal envelope ripple (undershoot) |
| M_s | RF signal level when OFF |
| M_{hs} | FHSS signal level during a hop |
| NR | Nonremovable indicator |
| Q | Slot-count parameter |
| R=>T | Interrogator-to-Tag |
| RTcal | Interrogator-to-Tag calibration symbol |
| S | Security-services indicator (whether a Tag supports the <i>Challenge</i> and/or <i>Authenticate</i> commands) |
| SLI | SL indicator |
| T | Numbering system identifier |
| T₁ | Time from Interrogator transmission to Tag response for an <i>immediate</i> Tag reply |
| T₂ | Time from Tag response to Interrogator transmission |
| T₃ | Time an Interrogator waits, after T ₁ , before it issues another command |
| T₄ | Minimum time between Interrogator commands |
| T₅ | Time from Interrogator transmission to Tag response for a <i>delayed</i> Tag reply |
| T₆ | Time from Interrogator transmission to first Tag response for an <i>in-process</i> Tag reply |
| T₇ | Time between Tag responses for an <i>in-process</i> Tag reply |
| T_f or T_{f,10-90%} | RF signal envelope fall time |
| T_{hf} | FHSS signal envelope fall time |
| T_{hr} | FHSS signal envelope rise time |
| T_{hs} | Time for an FHSS signal to settle to within a specified percentage of its final value |
| T_{pri} | Backscatter-link pulse-repetition interval ($T_{pri} = 1/BLF = TR_{cal}/DR$) |
| T_r or T_{r,10-90%} | RF signal envelope rise time |
| T_s | Time for an RF signal to settle to within a specified percentage of its final value |
| T=>R | Tag-to-Interrogator |
| TN | Tag-notification indicator |
| TRcal | Tag-to-Interrogator calibration symbol |
| U | Untraceability indicator |
| UMI | User-memory indicator |
| X | XTID indicator (whether a Tag implements an XTID) |
| XEB | XPC_W2 indicator |
| XI | XPC_W1 indicator |
| XPC | Extended protocol control |
| xxxx₂ | binary notation |
| xxxx_h | hexadecimal notation |

5.2 Abbreviated terms

| | |
|------------|-------------------------------|
| AFI | Application family identifier |
| AM | Amplitude modulation |
| ASK | Amplitude shift keying |
| CRC | Cyclic redundancy check |

| | |
|----------------|---|
| CW | Continuous wave |
| dBch | Decibels referenced to the integrated power in the reference channel |
| DSB | Double sideband |
| DSB-ASK | Double-sideband amplitude-shift keying |
| EPC | Electronic product code |
| ETSI | European Telecommunications Standards Institute |
| FCC | Federal Communications Commission |
| FDM | Frequency-Division Multiplexing |
| FHSS | Frequency-hopping spread spectrum |
| Handle | 16-bit Tag identifier |
| MAC | Message authentication code |
| PC | Protocol control |
| PIE | Pulse-interval encoding |
| Pivot | Decision threshold differentiating an R=>T data-0 symbol from a data-1 symbol |
| ppm | Parts-per-million |
| PSK | Phase shift keying or phase shift keyed |
| PR-ASK | Phase-reversal amplitude shift keying |
| RF | Radio frequency |
| RFID | Radio-frequency identification |
| RFU | Reserved for future use |
| RN16 | 16-bit random or pseudo-random number |
| RNG | Random or pseudo-random number generator |
| ITF | Interrogator talks first (reader talks first) |
| SSB | Single sideband |
| SSB-ASK | Single-sideband amplitude-shift keying |
| TDM | Time-division multiplexing or time-division multiplexed (as appropriate) |
| TID | Tag-identification or Tag identifier, depending on context |
| Word | 16 bits |
| XPC_W1 | XPC word 1 |
| XPC_W2 | XPC word 2 |
| XTID | Extended Tag identifier |

5.3 Notation

This protocol uses the following notational conventions:

- States and flags are denoted in bold. Some command parameters are also flags; a command parameter used as a flag will be bold. Example: **ready**.
- Command parameters are underlined. Some flags are also command parameters; a flag used as a command parameter will be underlined. Example: Pointer.
- Commands are denoted in italics. Variables are also denoted in italics. Where there might be confusion between commands and variables, this protocol will make an explicit statement. Example: *Query*.
- For logical negation, labels are preceded by '~'. Example: If **flag** is true, then **~flag** is false.
- The symbol, R=>T, refers to commands or signaling from an Interrogator to a Tag (Reader-to-Tag).
- The symbol, T=>R, refers to commands or signaling from a Tag to an Interrogator (Tag-to-Reader).

6. Protocol requirements

6.1 Protocol overview

6.1.1 Physical layer

An Interrogator sends information to one or more Tags by modulating an RF carrier using double-sideband amplitude shift keying (DSB-ASK), single-sideband amplitude shift keying (SSB-ASK), or phase-reversal amplitude shift keying (PR-ASK) using a pulse-interval encoding (PIE) format. Tags receive their operating energy from this same modulated RF carrier.

An Interrogator receives information from a Tag by transmitting an unmodulated RF carrier and listening for a backscattered reply. Tags communicate information by backscatter modulating the amplitude and/or phase of the RF carrier. The encoding format, selected in response to Interrogator commands, is either FMO or Miller-modulated subcarrier. The communications link between Interrogators and Tags is half-duplex, meaning that Tags shall not be required to demodulate Interrogator commands while backscattering. A Tag shall not respond to a mandatory or optional command using full-duplex communications.

6.1.2 Tag-identification layer

An Interrogator manages Tag populations using three basic operations:

- a. **Select.** Choosing a Tag population. An Interrogator may use a *Select* command to select one or more Tags based on a value or values in Tag memory, and may use a *Challenge* command to challenge one or more Tags based on Tag support for the desired cryptographic suite and authentication type. An Interrogator may subsequently inventory and access the chosen Tag(s).
- b. **Inventory.** Identifying individual Tags. An Interrogator begins an inventory round by transmitting a *Query* command in one of four sessions. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the Tag's EPC. Inventory comprises multiple commands. An inventory round operates in one and only one session at a time.
- c. **Access.** Communicating with an identified Tag. The Interrogator may perform a core operation such as reading, writing, locking, or killing the Tag; a security-related operation such as authenticating the Tag; or a file-related operation such as opening a particular file in the Tag's User memory. Access comprises multiple commands. An Interrogator may only access a uniquely identified Tag.

6.2 Protocol parameters

6.2.1 Signaling – Physical and media access control parameters

[Table 6-1](#) and [Table 6-2](#) provide an overview of parameters for R=>T and T=>R communications according to this protocol. For those parameters that do not apply to or are not used in this protocol the notation "N/A" indicates that the parameter is "Not Applicable".

Table 6-1: Interrogator-to-Tag (R=>T) communications

| Ref | Parameter Name | Description |
|--------|--|---|
| Int:1 | Operating Frequency Range | 860 – 960 MHz, as required by local regulations |
| Int:1a | Default Operating Frequency | Determined by local radio regulations and by the radio-frequency environment at the time of the communication |
| Int:1b | Operating Channels (spread-spectrum systems) | In accordance with local regulations; if the channelization is unregulated, then as specified |
| Int:1c | Operating Frequency Accuracy | As specified |
| Int:1d | Frequency Hop Rate (frequency-hopping [FHSS] systems) | In accordance with local regulations |

| Ref | Parameter Name | Description |
|---------|---|--|
| Int:1e | Frequency Hop Sequence (frequency-hopping [FHSS] systems) | In accordance with local regulations |
| Int:2 | Occupied Channel Bandwidth | In accordance with local regulations |
| Int:2a | Minimum Receiver Bandwidth | In accordance with local regulations |
| Int:3 | Interrogator Transmit Maximum EIRP | In accordance with local regulations |
| Int:4 | Interrogator Transmit Spurious Emissions | As specified; local regulation may impose tighter emission limits |
| Int:4a | Interrogator Transmit Spurious Emissions, In-Band (spread-spectrum systems) | As specified; local regulation may impose tighter emission limits |
| Int:4b | Interrogator Transmit Spurious Emissions, Out-of-Band | As specified; local regulation may impose tighter emission limits |
| Int:5 | Interrogator Transmitter Spectrum Mask | As specified; local regulation may impose tighter emission limits |
| Int:6 | Timing | As specified |
| Int:6a | Transmit-to-Receive Turn-Around Time | $\text{MAX}(\text{RT}_{\text{cal}}, 10\text{T}_{\text{pri}})$ nominal |
| Int:6b | Receive-to-Transmit Turn-Around Time | 3T_{pri} minimum; 20T_{pri} maximum when Tag is in reply & acknowledged states; no limit otherwise |
| Int:6c | Dwell Time or Interrogator Transmit Power-On Ramp | 1500 μs , maximum settling time |
| Int:6d | Decay Time or Interrogator Transmit Power-Down Ramp | 500 μs , maximum |
| Int:7 | Modulation | DSB-ASK, SSB-ASK, or PR-ASK |
| Int:7a | Spreading Sequence (direct-sequence [DSSS] systems) | N/A |
| Int:7b | Chip Rate (spread-spectrum systems) | N/A |
| Int:7c | Chip Rate Accuracy (spread-spectrum systems) | N/A |
| Int:7d | Modulation Depth | 90% nominal |
| Int:7e | Duty Cycle | 48% – 82.3% (time the waveform is high) |
| Int:7f | FM Deviation | N/A |
| Int:8 | Data Coding | PIE |
| Int:9 | Bit Rate | 26.7 kbps to 128 kbps (assuming equiprobable data) |
| Int:9a | Bit Rate Accuracy | +/- 1%, minimum |
| Int:10 | Interrogator Transmit Modulation Accuracy | As specified |
| Int:11 | Preamble | Required |
| Int:11a | Preamble Length | As specified |
| Int:11b | Preamble Waveform(s) | As specified |
| Int:11c | Bit Sync Sequence | None |
| Int:11d | Frame Sync Sequence | Required |
| Int:12 | Scrambling (spread-spectrum systems) | N/A |
| Int:13 | Bit Transmission Order | MSB is transmitted first |
| Int:14 | Wake-up Process | As specified |

| Ref | Parameter Name | Description |
|--------|----------------|---------------|
| Int:15 | Polarization | Not specified |

Table 6-2: Tag-to-Interrogator (T=>R) communications

| Ref. | Parameter Name | Description |
|--------|---|--|
| Tag:1 | Operating Frequency Range | 860 – 960 MHz, inclusive |
| Tag:1a | Default Operating Frequency | Tags respond to Interrogator signals that satisfy Int:1a |
| Tag:1b | Operating Channels (spread-spectrum systems) | Tags respond to Interrogator signals that satisfy Int:1b |
| Tag:1c | Operating Frequency Accuracy | As specified |
| Tag:1d | Frequency Hop Rate (frequency-hopping [FHSS] systems) | Tags respond to Interrogator signals that satisfy Int:1d |
| Tag:1e | Frequency Hop Sequence (frequency-hopping [FHSS] systems) | Tags respond to Interrogator signals that satisfy Int:1e |
| Tag:2 | Occupied Channel Bandwidth | In accordance with local regulations |
| Tag:3 | Transmit Maximum EIRP | In accordance with local regulations |
| Tag:4 | Transmit Spurious Emissions | In accordance with local regulations |
| Tag:4a | Transmit Spurious Emissions, In-Band (spread-spectrum systems) | In accordance with local regulations |
| Tag:4b | Transmit Spurious Emissions, Out-of-Band | In accordance with local regulations |
| Tag:5 | Transmit Spectrum Mask | In accordance with local regulations |
| Tag:6a | Transmit-to-Receive Turn-Around Time | $3T_{pri}$ minimum, $32T_{pri}$ maximum in reply & acknowledged states; no limit otherwise |
| Tag:6b | Receive-to-Transmit Turn-Around Time | $MAX(RT_{cal}, 10T_{pri})$ nominal |
| Tag:6c | Dwell Time or Transmit Power-On Ramp | Receive commands 1500 μ s after power-up |
| Tag:6d | Decay Time or Transmit Power-Down Ramp | N/A |
| Tag:7 | Modulation | ASK and/or PSK modulation (selected by Tag) |
| Tag:7a | Spreading Sequence (direct sequence [DSSS] systems) | N/A |
| Tag:7b | Chip Rate (spread-spectrum systems) | N/A |
| Tag:7c | Chip Rate Accuracy (spread-spectrum systems) | N/A |
| Tag:7d | On-Off Ratio | Not specified |
| Tag:7e | Subcarrier Frequency | 40 kHz to 640 kHz |
| Tag:7f | Subcarrier Frequency Accuracy | As specified |
| Tag:7g | Subcarrier Modulation | Miller, at the data rate |
| Tag:7h | Duty Cycle | FM0: 50%, nominal Subcarrier: 50%, nominal |
| Tag:7i | FM Deviation | N/A |
| Tag:8 | Data Coding | Baseband FM0 or Miller-modulated subcarrier (selected by the Interrogator) |
| Tag:9 | Bit Rate | FM0: 40 kbps to 640 kbps Subcarrier modulated: 5 kbps to 320 kbps |

| Ref. | Parameter Name | Description |
|---------|---|---|
| Tag:9a | Bit Rate Accuracy | Same as Subcarrier Frequency Accuracy; see Tag:7f |
| Tag:10 | Tag Transmit Modulation Accuracy (frequency-hopping [FHSS] systems) | N/A |
| Tag:11 | Preamble | Required |
| Tag:11a | Preamble Length | As specified |
| Tag:11b | Preamble Waveform | As specified |
| Tag:11c | Bit-Sync Sequence | None |
| Tag:11d | Frame-Sync Sequence | None |
| Tag:12 | Scrambling (spread-spectrum systems) | N/A |
| Tag:13 | Bit Transmission Order | MSB is transmitted first |
| Tag:14 | Reserved | Deliberately left blank |
| Tag:15 | Polarization | Tag dependent; not specified by this protocol |
| Tag:16 | Minimum Tag Receiver Bandwidth | Tag dependent; not specified by this protocol. |

6.2.2 Logical – Operating procedure parameters

[Table 6-3](#) and [Table 6-4](#) identify and describe parameters used by an Interrogator during the selection, inventory, and access of Tags according to this protocol. For those parameters that do not apply to or are not used in this protocol the notation “N/A” indicates that the parameter is “Not Applicable”.

Table 6-3: Tag inventory and access parameters

| Ref. | Parameter Name | Description |
|------|---------------------------|---|
| P:1 | Who Talks First | Interrogator |
| P:2 | Tag Addressing Capability | As specified |
| P:3 | Tag EPC | Contained in Tag memory |
| P:3a | EPC Length | As specified |
| P:3b | EPC Format | T=0 ₂ : As specified in the GS1 EPC Tag Data Standard, T=1 ₂ : As specified in ISO/IEC 15961 |
| P:4 | Read size | Multiples of 16 bits |
| P:5 | Write Size | Multiples of 16 bits |
| P:6 | Read Transaction Time | Varied with R=>T & T=>R link rate and number of bits being read |
| P:7 | Write Transaction Time | 20 ms (maximum) after end of <i>Write</i> command |
| P:8 | Error Detection | Interrogator-to-Tag: <i>Select</i> and <i>Challenge</i> commands: 16-bit CRC <i>Query</i> command: 5-bit CRC Other Inventory commands: Command length Access commands: 16-bit CRC Tag-to-Interrogator: PC/XPC, EPC: 16-bit CRC RN16: None or 16-bit CRC (varies by command) <u>handle</u> : 16-bit CRC All other: 16-bit CRC |
| P:9 | Error Correction | None |
| P:10 | Memory Size | Tag dependent, extensible (size is neither limited nor specified by this protocol) |

| Ref. | Parameter Name | Description |
|------|-------------------------------------|--------------|
| P:11 | Command Structure and Extensibility | As specified |

Table 6-4: Collision management parameters

| Ref. | Parameter Name | Description |
|------|---------------------------------------|--|
| A:1 | Type (Probabilistic or Deterministic) | Probabilistic |
| A:2 | Linearity | Linear up to 2^{15} Tags in the Interrogator's RF field; above that number, $N \log N$ for Tags with unique EPCs |
| A:3 | Tag Inventory Capacity | Unlimited for Tags with unique EPCs |

6.3 Description of operating procedure

The operating procedure defines the physical and logical requirements for an Interrogator-talks-first, random-slotted collision arbitration, RFID system operating in the 860 – 960 MHz frequency range.

6.3.1 Physical interface

The physical interface between an Interrogator and a Tag may be viewed as the signaling layer in a layered network communication system. The signaling interface defines frequencies, modulation, data coding, RF envelope, data rates, and other parameters required for RF communications.

6.3.1.1 Operational frequencies

Tags shall receive power from and communicate with Interrogators within the frequency range from 860 – 960 MHz, inclusive. An Interrogator's choice of operational frequency will be determined by local radio regulations and by the local radio-frequency environment. Interrogators certified for operation in dense-Interrogator environments shall support, but are not required to always use, the optional dense-Interrogator mode described in [Annex G](#).

6.3.1.2 Interrogator-to-Tag (R=>T) communications

An Interrogator communicates with one or more Tags by modulating an RF carrier using DSB-ASK, SSB-ASK, or PR-ASK with PIE encoding. Interrogators shall use a fixed modulation format and data rate for the duration of an inventory round, where "inventory round" is defined in [4.1](#). The Interrogator sets the data rate by means of the preamble that initiates the inventory round.

The high values in [Figure 6-1](#), [Figure 6-2](#), [Figure 6-3](#), [Figure 6-4](#), and [Figure 6-5](#) correspond to emitted CW (i.e. an Interrogator delivering power to the Tag or Tags) whereas the low values correspond to attenuated CW.

6.3.1.2.1 Interrogator frequency accuracy

Interrogators certified for operation in single- or multiple-Interrogator environments shall have a frequency accuracy that meets local regulations.

Interrogators certified for operation in dense-Interrogator environments shall have a frequency accuracy of ± 10 ppm over the nominal temperature range (-25°C to $+40^{\circ}\text{C}$) and ± 20 ppm over the extended temperature range (-40°C to $+65^{\circ}\text{C}$). Interrogators rated by the manufacturer to have a temperature range wider than nominal but different from extended shall have a frequency accuracy of ± 10 ppm over the nominal temperature range and ± 20 ppm to the extent of their rated range. If local regulations specify tighter frequency accuracy then the Interrogator shall meet the local regulations.

6.3.1.2.2 Modulation

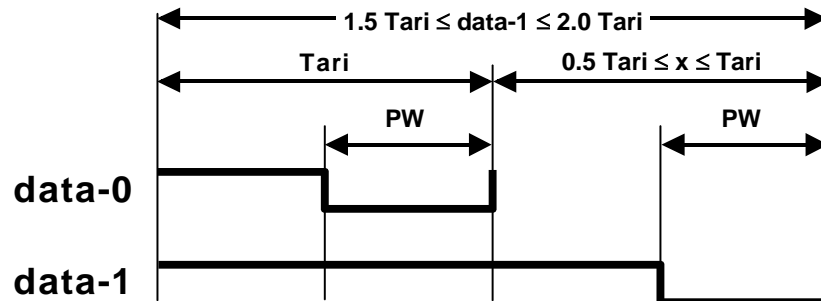
Interrogators shall communicate using DSB-ASK, SSB-ASK, or PR-ASK modulation, detailed in [Annex H](#). Tags shall demodulate all three modulation types.

6.3.1.2.3 Data encoding

The R=>T link shall use PIE, shown in [Figure 6-1](#). T_{ari} is the reference time interval for

Interrogator-to-Tag signaling, and is the duration of a data-0. High values represent transmitted CW; low values represent attenuated CW. Pulse modulation depth, rise time, fall time, and PW shall be as specified in [Table 6-5](#), and shall be the same for a data-0 and a data-1. Interrogators shall use a fixed modulation depth, rise time, fall time, PW, Tari, data-0 length, and data-1 length for the duration of an inventory round. The RF envelope shall be as specified in [Figure 6-2](#).

Figure 6-1: PIE symbols



6.3.1.2.4 Tari values

Interrogators shall communicate using Tari values in the range of $6.25\mu\text{s}$ to $25\mu\text{s}$. Interrogator compliance shall be evaluated using at least one Tari value between $6.25\mu\text{s}$ and $25\mu\text{s}$ with at least one value of the parameter x . The tolerance on all parameters specified in units of Tari shall be $\pm 1\%$. The choice of Tari value and x shall be in accordance with local radio regulations.

6.3.1.2.5 R=>T RF envelope

The R=>T RF envelope shall comply with [Figure 6-2](#) and [Table 6-5](#). The electric or magnetic field strength A (as appropriate) is the maximum amplitude of the RF envelope, measured in units of V/m or A/m, respectively. Tari is defined in [Figure 6-1](#). The pulsewidth is measured at the 50% point on the pulse. An Interrogator shall not change the R=>T modulation type (i.e. shall not switch between DSB-ASK, SSB-ASK, or PR-ASK) without first powering down its RF waveform (see [6.3.1.2.7](#)).

6.3.1.2.6 Interrogator power-up waveform

The Interrogator power-up RF envelope shall comply with [Figure 6-3](#) and [Table 6-6](#). Once the carrier level has risen above the 10% level, the power-up envelope shall rise monotonically until at least the ripple limit M_i . The RF envelope shall not fall below the 90% point in Figure 6.3 during interval T_s . Interrogators shall not issue commands before the end of the maximum settling-time interval in [Table 6-6](#) (i.e. before the end of T_s). Interrogators shall meet the frequency-accuracy requirement specified in [6.3.1.2.1](#) by the end of interval T_s in [Figure 6-3](#).

Figure 6-2: Interrogator-to-Tag RF envelope

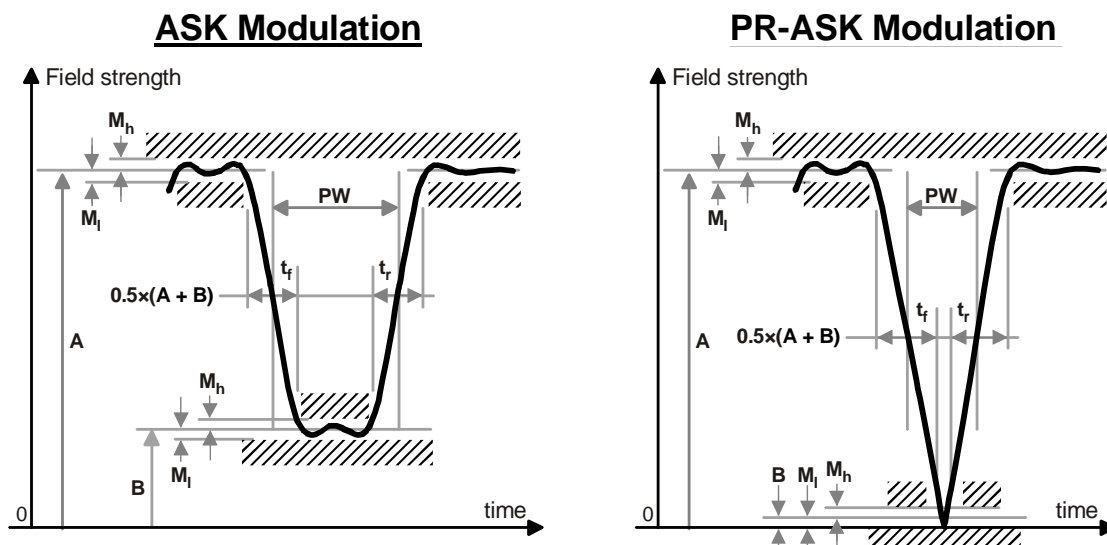
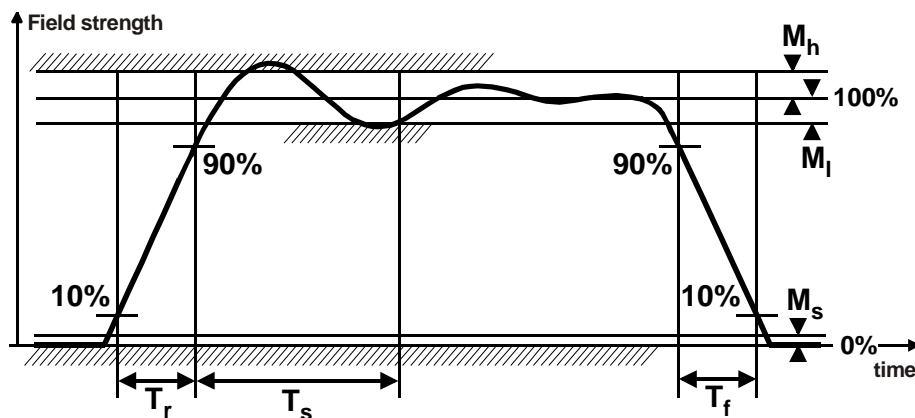


Table 6-5: RF envelope parameters

| Tari | Parameter | Symbol | Minimum | Nominal | Maximum | Units |
|----------------------------------|-----------------------|-----------------|---------------------|---------|-------------|---------------|
| 6.25 μ s to 25 μ s | Modulation Depth | $(A-B)/A$ | 80 | 90 | 100 | % |
| | RF Envelope Ripple | $M_h = M_l$ | 0 | | $0.05(A-B)$ | V/m or A/m |
| | RF Envelope Rise Time | $t_{r,10-90\%}$ | 0 | | $0.33Tari$ | μ s |
| | RF Envelope Fall Time | $t_{f,10-90\%}$ | 0 | | $0.33Tari$ | μ s |
| | RF Pulsewidth | PW | $MAX(0.265Tari, 2)$ | | $0.525Tari$ | μ s |

Figure 6-3: Interrogator power-up and power-down RF envelope

Table 6-6: Interrogator power-up waveform parameters

| Parameter | Definition | Minimum | Nominal | Maximum | Units |
|-----------|-----------------------|---------|---------|---------|--------------|
| T_r | Rise time | 1 | | 500 | μ s |
| T_s | Settling time | | | 1500 | μ s |
| M_s | Signal level when OFF | | | 1 | % full scale |
| M_l | Undershoot | | | 5 | % full scale |
| M_h | Overshoot | | | 5 | % full scale |

Table 6-7: Interrogator power-down waveform parameters

| Parameter | Definition | Minimum | Nominal | Maximum | Units |
|-----------|-----------------------|---------|---------|---------|--------------|
| T_f | Fall time | 1 | | 500 | μ s |
| M_s | Signal level when OFF | | | 1 | % full scale |
| M_l | Undershoot | | | 5 | % full scale |
| M_h | Overshoot | | | 5 | % full scale |

6.3.1.2.7 Interrogator power-down waveform

The Interrogator power-down RF envelope shall comply with [Figure 6-3](#) and [Table 6-7](#). Once the carrier level has fallen below the 90% level, the power-down envelope shall fall monotonically until the power-off limit M_s . Once powered off, an Interrogator shall remain powered off for a least 1ms before powering up again.

6.3.1.2.8 R=>T preamble and frame-sync

An Interrogator shall begin all R=>T signaling with either a preamble or a frame-sync, both of which are shown in [Figure 6-4](#). A preamble shall precede a *Query* command (see [6.3.2.12.2.1](#)) and

denotes the start of an inventory round. All other signaling shall begin with a frame-sync. The tolerance on all parameters specified in units of T_{ari} shall be $\pm 1\%$. PW shall be as specified in [Table 6-5](#). The RF envelope shall be as specified in [Figure 6-2](#).

A preamble shall comprise a fixed-length start delimiter, a data-0 symbol, an R=>T calibration (RTcal) symbol, and a T=>R calibration (TRcal) symbol.

- **RTcal:** An Interrogator shall set RTcal equal to the length of a data-0 symbol plus the length of a data-1 symbol ($RT_{\text{cal}} = 0_{\text{length}} + 1_{\text{length}}$). A Tag shall measure the length of RTcal and compute $pivot = RT_{\text{cal}} / 2$. A Tag shall interpret subsequent Interrogator symbols shorter than $pivot$ to be data-0s, and subsequent Interrogator symbols longer than $pivot$ to be data-1s. A Tag shall interpret symbols longer than 4 RTcal to be invalid. Prior to changing RTcal, an Interrogator shall transmit CW for a minimum of 8 RTcal.
- **TRcal:** An Interrogator shall specify a Tag's backscatter link frequency (its FMO datarate or the frequency of its Miller subcarrier) using the TRcal and divide ratio (DR) in the preamble and payload, respectively, of a *Query* command that initiates an inventory round. Equation (1) specifies the relationship between the backscatter link frequency (BLF), TRcal, and DR. A Tag shall measure the length of TRcal, compute BLF, and adjust its T=>R link rate to be equal to BLF ([Table 6-9](#) shows BLF values and tolerances). The TRcal and RTcal that an Interrogator uses in any inventory round shall meet the constraints in Equation (2):

$$BLF = \frac{DR}{TR_{\text{cal}}} \quad (1)$$

$$1.1 \times RT_{\text{cal}} \leq TR_{\text{cal}} \leq 3 \times RT_{\text{cal}} \quad (2)$$

A frame-sync is identical to a preamble, minus the TRcal symbol. An Interrogator, for the duration of an inventory round, shall use the same length RTcal in a frame-sync as it used in the preamble that initiated the round.

6.3.1.2.9 Frequency-hopping spread-spectrum waveform

When an Interrogator uses frequency-hopping spread spectrum (FHSS) signaling, the Interrogator's RF envelope shall comply with [Figure 6-5](#) and [Table 6-8](#). The RF envelope shall not fall below the 90% point in [Figure 6.5](#) during interval T_{hs} . Interrogators shall not issue commands before the end of the maximum settling-time interval in [Table 6-8](#) (i.e. before the end of T_{hs}). The maximum time between frequency hops and the minimum RF-off time during a hop shall meet local regulatory requirements. Interrogators shall meet the frequency-accuracy requirement specified in [6.3.1.2.1](#) by the end of interval T_{hs} in [Figure 6-5](#).

6.3.1.2.10 Frequency-hopping spread-spectrum channelization

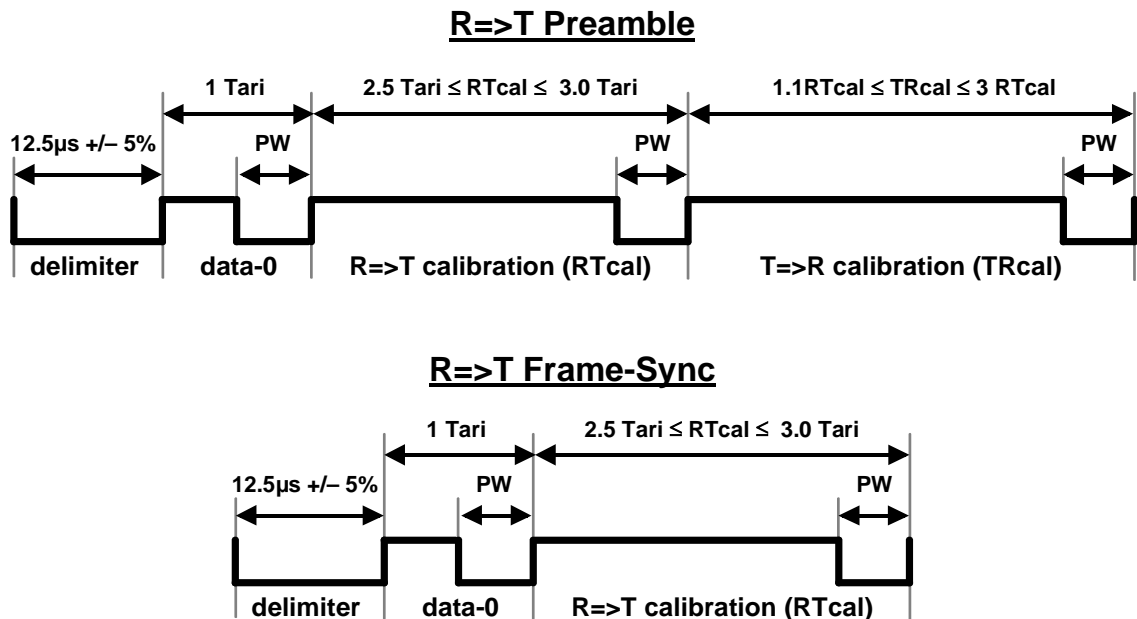
Interrogators certified for operation in single-Interrogator environments shall meet local regulations for spread-spectrum channelization. Interrogators certified for operation in multiple- or dense-Interrogator environments shall meet local regulations for spread-spectrum channelization, unless the channelization is unregulated, in which case Interrogators shall adopt the channel plan at <http://www.gs1.org/epcglobal/implementation> for the chosen regulatory region (see also [Annex G](#), which describes multiple- and dense-Interrogator channelized signaling).

6.3.1.2.11 Transmit mask

Interrogators certified for operation according to this protocol shall meet local regulations for out-of-channel and out-of-band spurious radio-frequency emissions.

Interrogators certified for operation in multiple-Interrogator environments shall meet both local regulations and the Multiple-Interrogator Transmit Mask described below and shown in [Figure 6-6](#).

Figure 6-4: R=>T preamble and frame-sync



- **Multiple-Interrogator Transmit Mask:** For an Interrogator transmitting random data in channel R , and any other channel $S \neq R$, the ratio of the integrated power $P()$ in channel S to that in channel R shall not exceed the specified values:
 - $|R - S| = 1$: $10\log_{10}(P(S) / P(R)) < -20$ dB
 - $|R - S| = 2$: $10\log_{10}(P(S) / P(R)) < -50$ dB
 - $|R - S| = 3$: $10\log_{10}(P(S) / P(R)) < -60$ dB
 - $|R - S| > 3$: $10\log_{10}(P(S) / P(R)) < -65$ dB

Where $P()$ denotes the total integrated power in the specified channel. This mask is shown graphically in [Figure 6-6](#), with dBch defined as dB referenced to the integrated power in the reference channel. The channel width shall be as specified by local regulations, unless the width is unregulated, in which case Interrogators shall adopt the width shown at <http://www.gs1.org/epcglobal/implementation> for the chosen regulatory region. The channel spacing shall be set equal to the channel width (measured channel center to channel center). For any transmit channel R , two exceptions to the mask are permitted, provided that

- neither exception exceeds -50 dBch, and
- neither exception exceeds local regulatory requirements.

An exception occurs when the integrated power in a channel S exceeds the mask. Each channel that exceeds the mask shall be counted as an exception.

Interrogators certified for operation in dense-Interrogator environments shall meet both local regulations and the Dense-Interrogator Transmit Mask described below and shown in [Figure 6-7](#). Interrogators may meet the Dense-Interrogator Transmit Mask during non-dense-Interrogator operation. Regardless of the mask used, Interrogators certified for operation in dense-Interrogator environments shall not be permitted the two exceptions to the transmit mask that are allowed for Interrogators certified for operation in multiple-Interrogator environments.

- **Dense-Interrogator Transmit Mask:** For Interrogator transmissions centered at a frequency f_c , a $2.5/Tari$ bandwidth R_{BW} also centered at f_c , an offset frequency $f_o = 2.5/Tari$, and a $2.5/Tari$ bandwidth S_{BW} centered at $(n \times f_o) + f_c$ (integer n), the ratio of the integrated power $P()$ in S_{BW} to that in R_{BW} with the Interrogator transmitting random data shall not exceed the specified values:

Figure 6-5: FHSS Interrogator RF envelope

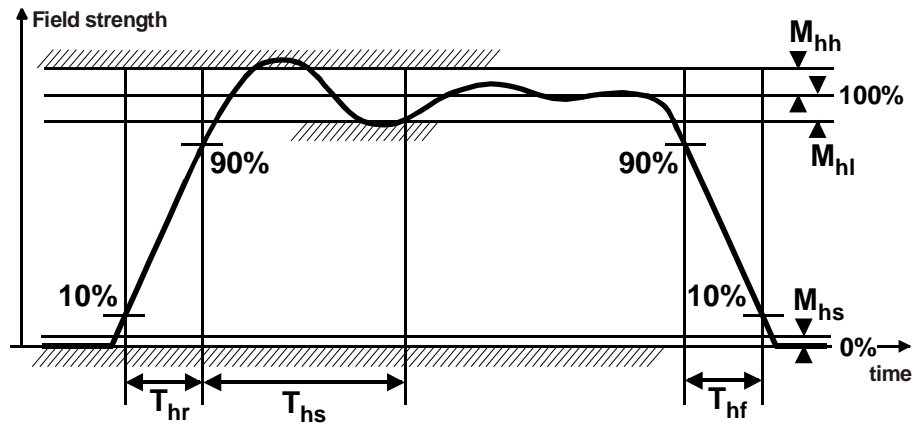


Table 6-8: FHSS waveform parameters

| Parameter | Definition | Minimum | Nominal | Maximum | Units |
|-----------|-------------------------|---------|---------|---------|--------------|
| T_{hr} | Rise time | | | 500 | μs |
| T_{hs} | Settling time | | | 1500 | μs |
| T_{hf} | Fall time | | | 500 | μs |
| M_{hs} | Signal level during hop | | | 1 | % full scale |
| M_{hl} | Undershoot | | | 5 | % full scale |
| M_{hh} | Overshoot | | | 5 | % full scale |

Figure 6-6: Transmit mask for multiple-Interrogator environments

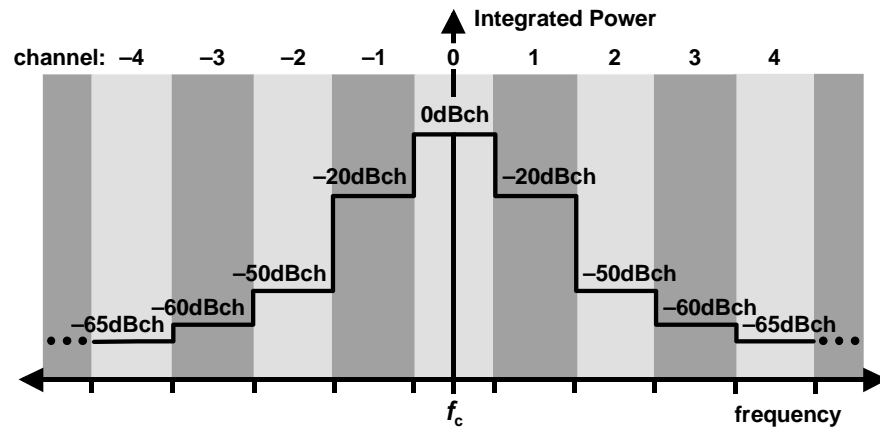
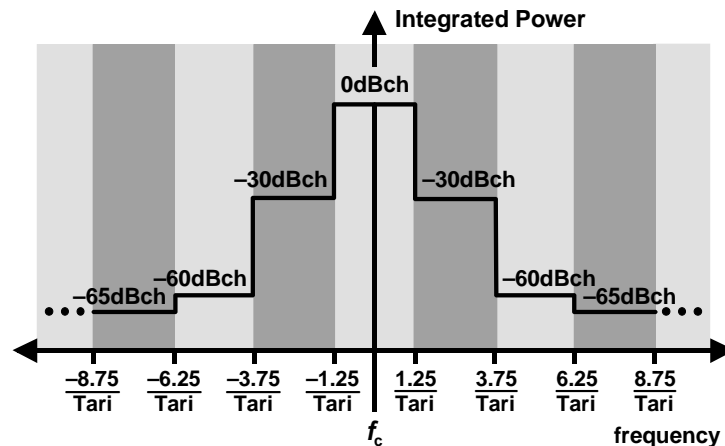


Figure 6-7: Transmit mask for dense-Interrogator environments


- $|n| = 1$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -30$ dB
- $|n| = 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -60$ dB
- $|n| > 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -65$ dB

Where $P()$ denotes the total integrated power in the $2.5/T_{ari}$ reference bandwidth. This mask is shown graphically in [Figure 6-7](#), with dBch defined as dB referenced to the integrated power in the reference channel.

6.3.1.3 Tag-to-Interrogator (T=>R) communications

A Tag communicates with an Interrogator using backscatter modulation, in which the Tag switches the reflection coefficient of its antenna between two states in accordance with the data being sent.

A Tag shall backscatter using a fixed modulation format, data encoding, and data rate for the duration of an inventory round, where “inventory round” is defined in [4.1](#). The Tag selects the modulation format; the Interrogator selects the data encoding and data rate by means of the *Query* command that initiates the round. The low values in [Figure 6-9](#), [Figure 6-10](#), [Figure 6-11](#), [Figure 6-13](#), [Figure 6-14](#), and [Figure 6-15](#) correspond to the antenna-reflectivity state the Tag exhibits during the CW period prior to a T=>R preamble (e.g. ASK Tag absorbing power), whereas the high values correspond to the antenna-reflectivity state the Tag exhibits during the first high pulse of a T=>R preamble (e.g. ASK Tag reflecting power).

6.3.1.3.1 Modulation

Tag backscatter shall use ASK and/or PSK modulation. The Tag manufacturer selects the modulation format. Interrogators shall demodulate both modulation types.

6.3.1.3.2 Data encoding

Tags shall encode the backscattered data as either FMO baseband or Miller modulation of a subcarrier at the data rate. The Interrogator specifies the encoding type.

6.3.1.3.2.1 FMO baseband

[Figure 6-8](#) shows basis functions and a state diagram for generating FMO (bi-phase space) encoding. FMO inverts the baseband phase at every symbol boundary; a data-0 has an additional mid-symbol phase inversion. The state diagram in [Figure 6-8](#) maps a logical data sequence to the FMO basis functions that are transmitted. The state labels, S_1 – S_4 , indicate four possible FMO-encoded symbols, represented by the two phases of each of the FMO basis functions. The state labels also represent the FMO waveform that is transmitted upon entering the state. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_2 to S_3 is disallowed because the resulting transmission would not have a phase inversion on a symbol boundary.

[Figure 6-9](#) shows generated baseband FMO symbols and sequences. The duty cycle of a 00 or 11

sequence, measured at the modulator output, shall be a minimum of 45% and a maximum of 55%, with a nominal value of 50%. FM0 encoding has memory; consequently, the choice of FM0 sequences in [Figure 6-9](#) depends on prior transmissions. FM0 signaling shall always end with a “dummy” data-1 bit at the end of a transmission, as shown in [Figure 6-10](#).

6.3.1.3.2.2 FM0 preamble

T=>R FM0 signaling shall begin with one of the two preambles shown in [Figure 6-11](#). The choice depends on the TRext value specified in the *Query* that initiated the inventory round, unless a Tag is replying to a command that uses a *delayed* or *in-process* reply (see [6.3.1.6](#)), in which case a Tag shall use the extended preamble regardless of TRext (i.e. a Tag replies as if TRext=1 regardless of the TRext value specified in the *Query*—see [6.3.2.12.3](#)). The “v” shown in [Figure 6-11](#) indicates an FM0 violation (i.e. a phase inversion should have occurred but did not).

Figure 6-8: FM0 basis functions and generator state diagram

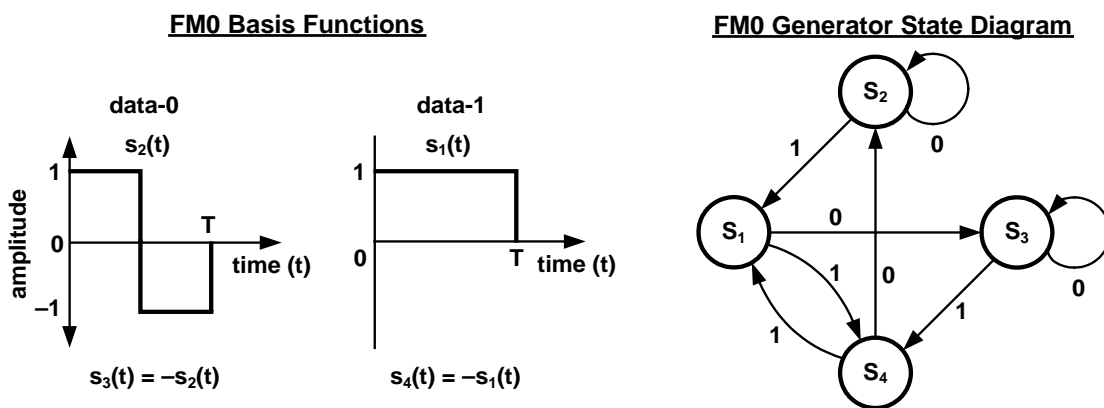


Figure 6-9: FM0 symbols and sequences

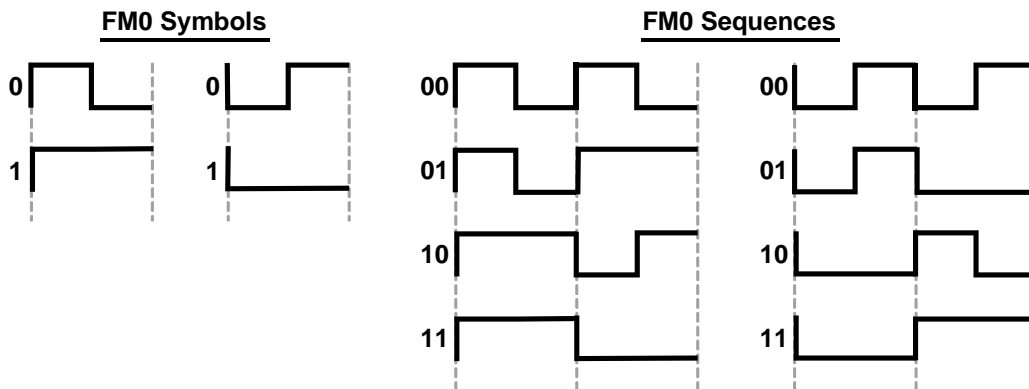


Figure 6-10: Terminating FM0 transmissions

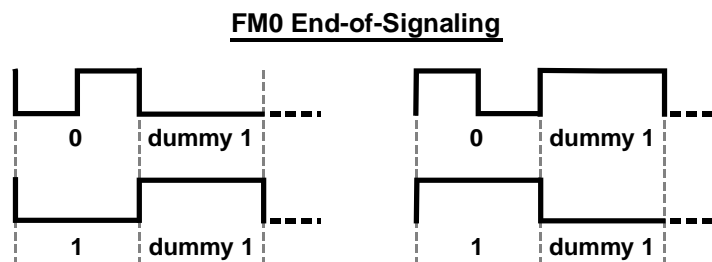
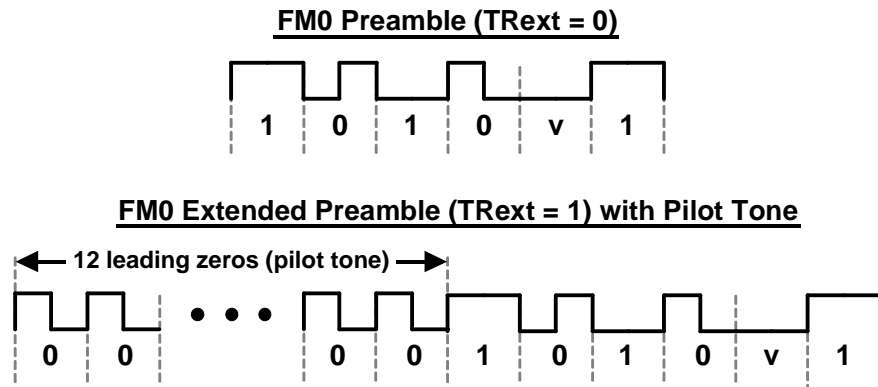
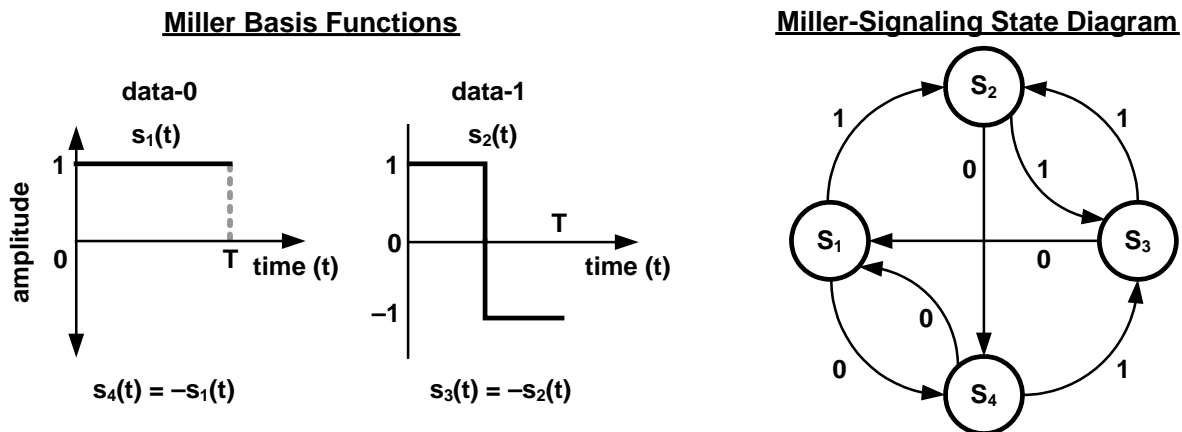


Figure 6-11: FMO T=>R preamble


6.3.1.3.2.3 Miller-modulated subcarrier

[Figure 6-12](#) shows basis functions and a state diagram for generating Miller encoding. Baseband Miller inverts its phase between two data-0s in sequence. Baseband Miller also places a phase inversion in the middle of a data-1 symbol. The state diagram in [Figure 6-12](#) maps a logical data sequence to baseband Miller basis functions. The state labels, S_1 – S_4 , indicate four possible Miller-encoded symbols, represented by the two phases of each of the Miller basis functions. The state labels also represent the baseband Miller waveform that is generated upon entering the state. The transmitted waveform is the baseband waveform multiplied by a square-wave at M times the symbol rate. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_1 to S_3 is disallowed because the resulting transmission would have a phase inversion on a symbol boundary between a data-0 and a data-1.

Figure 6-12: Miller basis functions and generator state diagram


[Figure 6-13](#) shows Miller-modulated subcarrier sequences; the Miller sequence shall contain exactly two, four, or eight subcarrier cycles per bit, depending on the M value specified in the *Query* command that initiated the inventory round (see [Table 6-10](#)). The duty cycle of a 0 or 1 symbol, measured at the modulator output, shall be a minimum of 45% and a maximum of 55%, with a nominal value of 50%. Miller encoding has memory; consequently, the choice of Miller sequences in [Figure 6-13](#) depends on prior transmissions. Miller signaling shall always end with a “dummy” data-1 bit at the end of a transmission, as shown in [Figure 6-14](#).

6.3.1.3.2.4 Miller subcarrier preamble

T=>R subcarrier signaling shall begin with one of the two preambles shown in [Figure 6-15](#). The choice depends on the TRext value specified in the *Query* that initiated the inventory round, unless a Tag is replying to a command that uses a *delayed* or *in-process* reply (see [6.3.1.6](#)), in which case a Tag shall use the extended preamble regardless of TRext (i.e. a Tag replies as if TRext=1 regardless of the TRext value specified in the *Query*—see [6.3.2.12.3](#)).

Figure 6-13: Subcarrier sequences

Miller Subcarrier Sequences

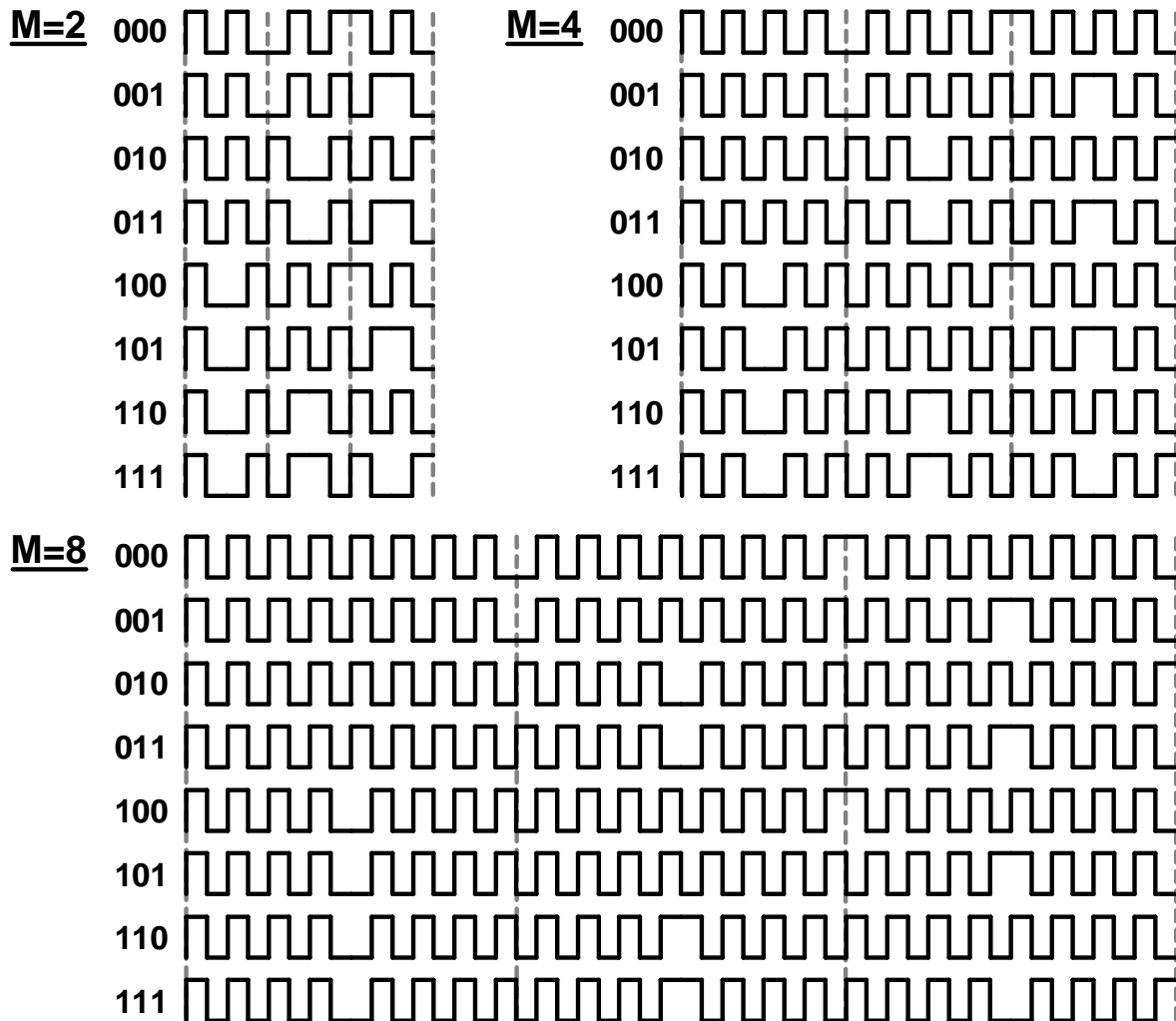


Figure 6-14: Terminating subcarrier transmissions

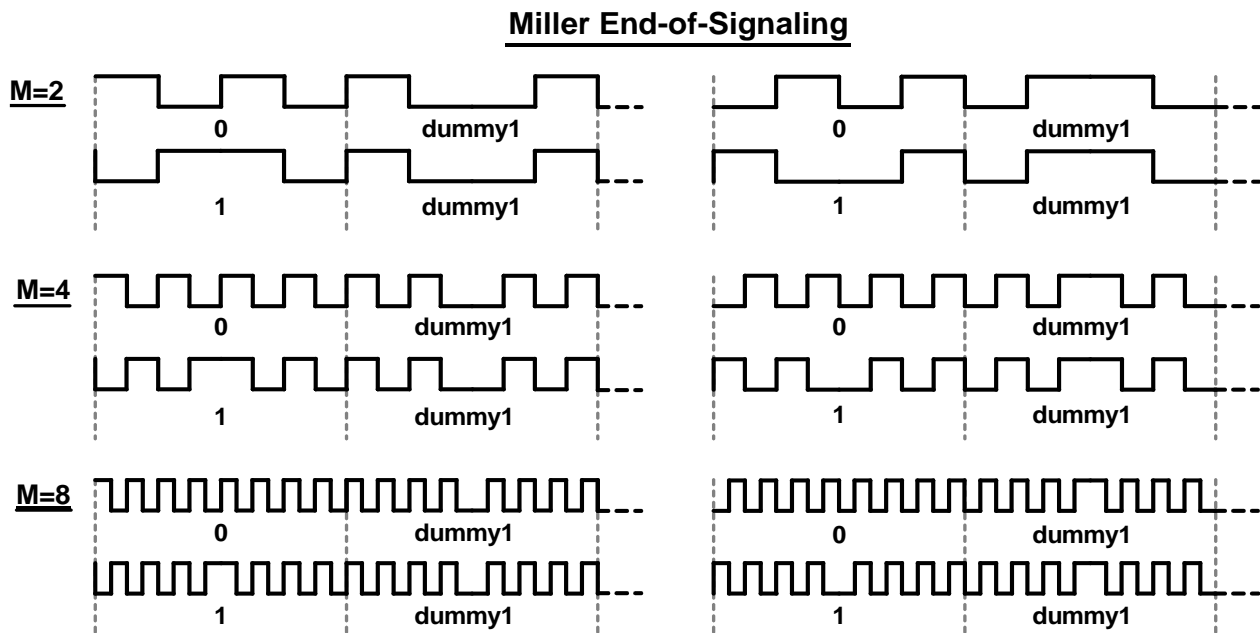
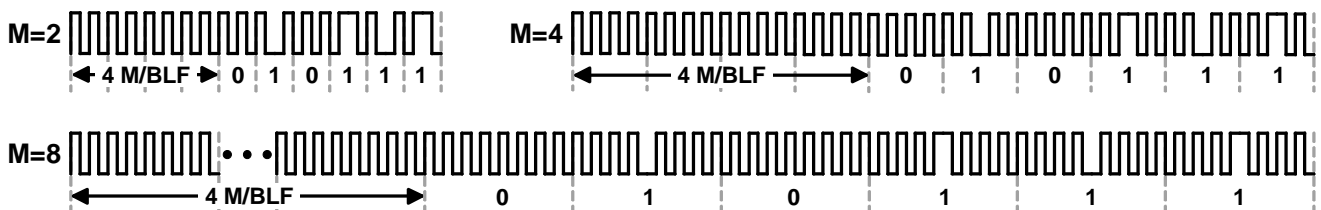
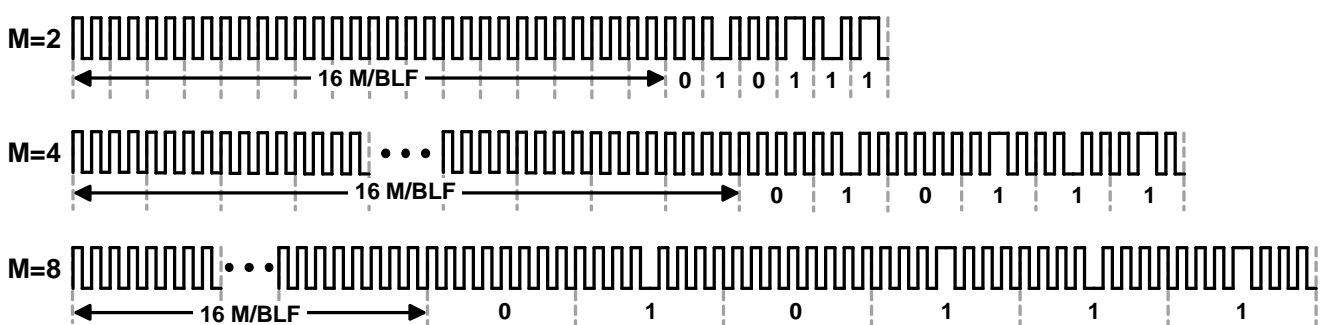


Figure 6-15: Subcarrier T=>R preamble

Miller Preamble (TRext = 0)



Miller Extended Preamble (TRext = 1) with Pilot Tone



6.3.1.3.3 Tag supported Tari values and backscatter link rates

Tags shall support all R=>T Tari values in the range of 6.25μs to 25μs, over all parameters allowed by [6.3.1.2.3](#).

Tags shall support the T=>R link frequencies and tolerances specified in [Table 6-9](#) and the T=>R data rates specified in [Table 6-10](#). The frequency-variation requirement in [Table 6-9](#) includes both frequency drift and short-term frequency variation during Tag response to an Interrogator command. The Query command that initiates an inventory round specifies DR in [Table 6-9](#) and M in [Table 6-10](#); the preamble that precedes the Query specifies TRcal. BLF is computed using Eq. (1).

These four parameters together define the backscatter frequency, modulation type (FMO or Miller), and T=>R data rate for the round (see also [6.3.1.2.8](#)).

6.3.1.3.4 Tag power-up timing

Tags energized by an Interrogator shall be capable of receiving and acting on Interrogator commands within a period not exceeding the maximum settling-time interval specified in [Table 6-6](#) or [Table 6-8](#), as appropriate (i.e. by the end of T_s or T_{hs}, respectively).

6.3.1.3.5 Minimum operating RF field strength and backscatter strength

For a Tag certified to this protocol, the Tag manufacturer shall specify:

1. free-space sensitivity,
2. minimum backscattered modulated power (ASK modulation) or change in radar cross-section or equivalent (phase modulation), and

Table 6-9: Tag-to-Interrogator link frequencies

| DR: Divide Ratio | TRcal ¹ (µs +/- 1%) | BLF: Link Frequency (kHz) | Frequency Tolerance FrT (nominal temp) | Frequency Tolerance FrT (extended temp) | Frequency variation during backscatter |
|------------------|--------------------------------|---------------------------|--|---|--|
| 64/3 | 33.3 | 640 | + / - 15% | + / - 15% | + / - 2.5% |
| | 33.3 < TRcal < 66.7 | 320 < BLF < 640 | + / - 22% | + / - 22% | + / - 2.5% |
| | 66.7 | 320 | + / - 10% | + / - 15% | + / - 2.5% |
| | 66.7 < TRcal < 83.3 | 256 < BLF < 320 | + / - 12% | + / - 15% | + / - 2.5% |
| | 83.3 | 256 | + / - 10% | + / - 10% | + / - 2.5% |
| | 83.3 < TRcal ≤ 133.3 | 160 ≤ BLF < 256 | + / - 10% | + / - 12% | + / - 2.5% |
| | 133.3 < TRcal ≤ 200 | 107 ≤ BLF < 160 | + / - 7% | + / - 7% | + / - 2.5% |
| | 200 < TRcal ≤ 225 | 95 ≤ BLF < 107 | + / - 5% | + / - 5% | + / - 2.5% |
| 8 | 17.2 ≤ TRcal < 25 | 320 < BLF ≤ 465 | + / - 19% | + / - 19% | + / - 2.5% |
| | 25 | 320 | + / - 10% | + / - 15% | + / - 2.5% |
| | 25 < TRcal < 31.25 | 256 < BLF < 320 | + / - 12% | + / - 15% | + / - 2.5% |
| | 31.25 | 256 | + / - 10% | + / - 10% | + / - 2.5% |
| | 31.25 < TRcal < 50 | 160 < BLF < 256 | + / - 10% | + / - 10% | + / - 2.5% |
| | 50 | 160 | + / - 7% | + / - 7% | + / - 2.5% |
| | 50 < TRcal ≤ 75 | 107 ≤ BLF < 160 | + / - 7% | + / - 7% | + / - 2.5% |
| | 75 < TRcal ≤ 200 | 40 ≤ BLF < 107 | + / - 4% | + / - 4% | + / - 2.5% |

Note 1: Allowing two different TRcal values (with two different DR values) to specify the same BLF offers flexibility in specifying Tari and RTcal.

Table 6-10: Tag-to-Interrogator data rates

| M: Number of subcarrier cycles per symbol | Modulation type | Data rate (kbps) |
|---|-------------------|------------------|
| 1 | FMO baseband | BLF |
| 2 | Miller subcarrier | BLF/2 |

| M: Number of subcarrier cycles per symbol | Modulation type | Data rate (kbps) |
|---|-------------------|------------------|
| 4 | Miller subcarrier | BLF/4 |
| 8 | Miller subcarrier | BLF/8 |

- the manufacturer's normal operating conditions, for the Tag mounted on one or more manufacturer-selected materials.

6.3.1.4 Transmission order

The transmission order for all R=>T and T=>R communications shall be most-significant bit (MSB) first.

Within each message, the most-significant word shall be transmitted first.

Within each word, the MSB shall be transmitted first.

6.3.1.5 Cyclic-redundancy check (CRC)

A CRC is a cyclic-redundancy check that a Tag uses to ensure the validity of certain R=>T commands, and an Interrogator uses to ensure the validity of certain backscattered T=>R replies. This protocol uses two CRC types: (i) a CRC-16, and (ii) a CRC-5. [Annex F](#) describes both CRC types.

To generate a CRC-16 a Tag or Interrogator shall first generate the CRC-16 precursor shown in [Table 6-11](#), and then take the ones-complement of the generated precursor to form the CRC-16.

A Tag or Interrogator shall verify the integrity of a received message that uses a CRC-16. The Tag or Interrogator may use one of the methods described in [Annex F](#) to verify the CRC-16.

A Tag calculates and saves into memory a 16-bit StoredCRC. See [6.3.2.1.2.1](#).

During inventory a Tag backscatters a 16-bit PacketCRC that the Tag calculates dynamically.

Tags shall append a CRC-16 to those replies that use a CRC-16. See [6.3.2.12](#) for command-specific replies.

To generate a CRC-5 an Interrogator shall use the definition in [Table 6-12](#).

A Tag shall verify the integrity of a received message that uses a CRC-5. The Tag may use the method described in [Annex F](#) to verify a CRC-5.

Interrogators shall append the appropriate CRC to R=>T transmissions as specified in [Table 6-28](#).

6.3.1.6 Link timing

[Figure 6-18](#) illustrates R=>T and T=>R link timing. The figure (not drawn to scale) defines Interrogator interactions with a Tag population. [Table 6-16](#) shows the timing requirements for [Figure 6-18](#), while [6.3.2.12](#) describes the commands. Tags and Interrogators shall meet all timing requirements shown in [Table 6-16](#). RTcal is defined in [6.3.1.2.8](#); T_{pri} is the T=>R link period (T_{pri} = 1 / BLF). As described in [6.3.1.2.8](#), an Interrogator shall use a fixed R=>T link rate for the duration of an inventory round; prior to changing the R=>T link rate an Interrogator shall transmit CW for a minimum of 8 RTcal. [Figure 6-18](#) illustrates three types of Tag reply timing denoted *immediate*, *delayed*, and *in-process*. These reply timings are defined in [6.3.1.6.1](#), [6.3.1.6.2](#), and [6.3.1.6.3](#), respectively. [Table 6-28](#) specifies the reply type that a Tag uses for each Interrogator command.

Table 6-11: CRC-16 precursor

| CRC-16 precursor (See also Annex F) | | | | |
|--|---------|-----------------------------|-------------------|-------------------|
| CRC Type | Length | Polynomial | Preset | Residue |
| ISO/IEC 13239 | 16 bits | $x^{16} + x^{12} + x^5 + 1$ | FFFF _h | 1D0F _h |

Table 6-12: CRC-5 definition

| CRC-5 Definition (See also Annex F) | | | | |
|--|--------|-----------------|--------------------|--------------------|
| CRC Type | Length | Polynomial | Preset | Residue |
| — | 5 bits | $x^5 + x^3 + 1$ | 01001 ₂ | 00000 ₂ |

6.3.1.6.1 Immediate Tag reply

An *immediate* Tag reply is a reply that meets T_1 specified in [Table 6-16](#).

6.3.1.6.2 Delayed Tag reply

A *delayed* Tag reply is a reply that meets T_5 specified in [Table 6-16](#). After issuing a command that uses *delayed* reply timing an Interrogator shall transmit CW for at least the lesser of T_{REPLY} or $T_{5(max)}$, where T_{REPLY} is the time between the Interrogator's command and the Tag's backscattered reply. An Interrogator may observe several possible outcomes from a command that uses *delayed* reply timing, depending on the success or failure of the Tag's internal operations:

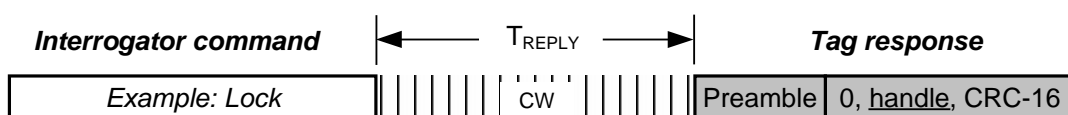
- The Tag successfully executes the command:** After executing the command the Tag shall backscatter the reply shown in [Table 6-13](#) and [Figure 6-16](#), comprising a header (a 0-bit), the Tag's handle, and a CRC-16 calculated over the 0-bit and handle. The reply shall meet the T_5 limits in [Table 6-16](#). If the Interrogator observes this reply within $T_{5(max)}$ then the command completed successfully.
- The Tag encounters an error:** The Tag shall backscatter an error code (see [Annex I](#)) during the CW period rather than the reply shown in [Table 6-13](#). The error code shall meet the T_5 limits in [Table 6-16](#). An error-code reply uses header=1 (see [Annex I](#)) to differentiate it from a successful reply.
- The Tag fails to execute the command:** If the Interrogator does not observe a Tag reply within $T_{5(max)}$ then the Tag did not execute the command successfully. The Interrogator typically issues a subsequent *Req_RN* (containing the Tag's handle) to verify that the Tag is still in the Interrogator's energizing RF field, and may then issue another command or commands.

A Tag shall ignore Interrogator commands while processing a prior command that specified a *delayed* reply. If an Interrogator transmits a command while the Tag is processing then the Tag may continue with its processing or, in environments with limited power availability, may undergo a power-on reset.

A *delayed* Tag reply shall use the extended preamble shown in [Figure 6-11](#) or [Figure 6-15](#), as appropriate (i.e. the Tag shall reply as if $T_{Rext}=1$ regardless of the T_{Rext} value in the *Query* that initiated the inventory round).

Table 6-13: Tag reply after successfully executing a delayed-reply command

| | Header | RN | CRC |
|-------------|--------|---------------|--------|
| # of bits | 1 | 16 | 16 |
| description | 0 | <u>handle</u> | CRC-16 |

Figure 6-16: Successful delayed-reply sequence


6.3.1.6.3 In-process Tag reply

An *in-process* reply allows a Tag to spend longer than $T_{5(max)}$ executing a command, and to notify the Interrogator on a periodic basis that it is still processing the command. An *in-process* reply may include multiple backscatter transmissions from Tag to Interrogator. The first transmission shall

meet the T_6 limits specified in [Table 6-16](#); subsequent transmissions (if any) shall meet T_7 . A Tag shall backscatter a transmission at least once every $T_{7(max)}$ while processing the command. A Tag may backscatter a “processing” notification more frequently than $T_{7(max)}$, and may backscatter an intermediate “processing” notification even if it can complete its processing within $T_{6(max)}$. An Interrogator specifies, in every access command that uses an *in-process* reply (except *AuthComm* – see [6.3.2.12.3.11](#)), whether a Tag, when done processing, backscatters its final response or stores it in the Tag’s ResponseBuffer. A Tag always backscatters (and never stores) the response to an *AuthComm*.

A Tag’s *in-process* reply or replies shall be as shown in [Table 6-14](#). The reply includes a 7-bit Barker code, done, header, optional length (length of the response regardless of whether the Tag backscatters or stores it), response (null if a Tag stores its response), the Tag’s handle, and a CRC-16 calculated from Barker code to handle, inclusive. The Tag replies shall be consistent for first and subsequent Tag transmissions – i.e. if the final reply includes length then all intermediate replies shall include length, and vice versa.

An Interrogator may observe several possible outcomes from a command that uses *in-process* reply timing, depending on the success or failure of the Tag’s internal operations:

- a. **The Tag successfully executes the command:** While processing the command the Tag backscatters a transmission as shown in [Table 6-14](#) at least once every $T_{7(max)}$. Done and header for these intermediate replies shall be zero, response shall be null, and if the replies include length then length=0000_h. When the Tag has finished processing it sends a final reply, also as shown in [Table 6-14](#), including done=1, header=0, response, and optional length. All replies shall meet the T_6 and T_7 limits specified in [Table 6-16](#). If the Interrogator observes a final reply with header=0 then the command completed successfully.
- b. **The Tag encounters an error:** While processing the command the Tag backscatters a transmission as shown in [Table 6-14](#) at least once every $T_{7(max)}$. Done and header for these intermediate replies shall be zero, response shall be null, and if the replies include length then length=0000_h. When a Tag encounters an error it sends a final reply, also as shown in [Table 6-14](#), including done=1, header=1, response, and optional length. All replies shall meet the T_6 and T_7 limits specified in [Table 6-16](#). If the Interrogator observes a final reply with header=1 then the Tag encountered an error (see [Annex I](#)).
- c. **The Tag fails to execute the command:** If the Interrogator does not observe a Tag reply within $T_{6(max)}$ for the first reply or $T_{7(max)}$ for subsequent replies then the Tag did not execute the command successfully. The Interrogator typically issues a subsequent *Req_RN* (containing the Tag’s handle) to verify that the Tag is still in the Interrogator’s energizing RF field, and may then issue another command or commands.

Done indicates whether the Tag is still processing a command. Done=0 means the Tag is still processing; done=1 means the Tag has finished processing. Header indicates whether response is an error code. Header=0 means the computation was successful and response includes a result; header=1 means response is an error code.

The optional length field specifies the length of a Tag’s computed response (in bits), regardless of whether a Tag backscatters or stores this response. Length shall comprise a 15-bit value field followed by an even parity bit (the number of 1’s in the 16-bit length field shall be an even number, with Length=0000_h an allowed value). An Interrogator specifies, in every command that uses an *in-process* reply, whether the Tag omits or includes length in its reply. If the Interrogator does not request length then the Tag omits length from its reply; if the Interrogator requests length then the Tag includes length in its reply. For the latter case, in the event of a stored response, length specifies the length of the stored response (and is therefore typically nonzero) but the response that the Tag actually backscatters will be null.

Response contains the Tag’s computed result or an error code if the Interrogator instructed the Tag to backscatter its response, or null if the Interrogator instructed the Tag to store its response in the ResponseBuffer.

[Table 6-15](#) shows values for the fields in an *in-process* reply depending on (a) whether a Tag sends its response or stores it in its ResponseBuffer, (b) how quickly the Tag executes the command, (c) whether the reply includes length, and (d) whether the Tag was able to successfully execute the command.

A Tag shall ignore Interrogator commands while processing a prior command that specified an *in-process* reply. If an Interrogator transmits a command while the Tag is processing then the Tag may continue with its processing or, in environments with limited power availability, may undergo a power-on reset.

After issuing a command that uses an *in-process* reply an Interrogator shall transmit CW until the Interrogator either (a) observes a reply with done=1 indicating the Tag has finished executing the command, or (b) fails to observe a reply for at least $T_{6(max)}$ or $T_{7(max)}$ (as appropriate) indicating that the Tag failed to execute the command.

An *in-process* Tag reply shall use the extended preamble shown in [Figure 6-11](#) or [Figure 6-15](#), as appropriate (i.e. the Tag shall reply as if TRext=1 regardless of the TRext value in the *Query* that initiated the inventory round).

6.3.1.6.4 ResponseBuffer

A Tag that implements a *Challenge* command, or any access command (other than *AuthComm*) that employs an *in-process* reply, shall implement a **C** flag and a ResponseBuffer with the following properties:

- The **C** flag, located in the Tag's XPC_W1 (see [Table 6-18](#)), indicates whether the Tag has a stored response (result or error code) in its ResponseBuffer. If **C**=1 then the ResponseBuffer contains data; if **C**=0 then the ResponseBuffer is empty.
- A Tag shall set **C**=0 upon either (1) receiving an access command containing SenRep (c.f. 6.3.2.12.3.10) (2) receiving a *Challenge* command, or (3) when specified by a cryptographic suite, and shall set **C**=1 after storing a response (result or error code) in its ResponseBuffer.
- The **C** flag shall be selectable using a *Select* command.

Table 6-14: In-process Tag reply omitting and including length field

| | Barker Code | Done | Header | Response | RN | CRC |
|-------------|-------------|---------------------------|-----------------------------|-----------------------------|--------------------------|--------|
| # of bits | 7 | 1 | 1 | variable | 16 | 16 |
| description | 1110010 | 0: Working 1: Finished | 0: Success 1: Error code | <u>result</u> or error code | <u>handl</u> <u>e</u> | CRC-16 |

| | Barker Code | Done | Header | Length | Response | RN | CRC |
|-------------|-------------|---------------------------|-----------------------------|---|-----------------------------|---------------|--------|
| # of bits | 7 | 1 | 1 | 16 | variable | 16 | 16 |
| description | 1110010 | 0: Working 1: Finished | 0: Success 1: Error code | 15 bits encoding <u>length</u> of <u>result</u> or error code followed by even parity bit | <u>result</u> or error code | <u>handle</u> | CRC-16 |

- If an access command with SenRep=0 or a *Challenge* command specifies IncRepLen=0 then a Tag shall not include a length field with its stored response, so the first word of the stored response shall be at ResponseBuffer location 00_h. If the command specifies IncRepLen=1 then ResponseBuffer bits 00_h – 0E_h shall contain the length of the stored response in bits, ResponseBuffer bit 0F_h shall contain an even parity bit that the Tag computes over bits 00_h – 0E_h, and the first word of the stored response shall be at ResponseBuffer location 10_h. See [Figure 6-17](#).
- The maximum size of a stored response shall be 32 kbits.
- The maximum ResponseBuffer size shall be 32,784 bits (15 length bits, 1 parity bit, 32k response bits). A Tag manufacturer may limit the ResponseBuffer to a size less than this maximum. A Tag shall dynamically adjust its ResponseBuffer, on a command-by-command basis, to the required size.
- An Interrogator may read the ResponseBuffer using a *ReadBuffer* command. See [6.3.2.12.3.15](#).

- The ResponseBuffer shall be read-only to an Interrogator.
- A Tag shall abort command processing and instead store an error code in its ResponseBuffer if and when it determines that response will overflow the ResponseBuffer (see [Annex I](#)).
- A Tag shall retain data in its ResponseBuffer with the persistence of its **C** flag (see [Table 6-20](#)). When **C** is 1 then a Tag shall maintain the data in its ResponseBuffer. When **C** is or becomes 0 then a Tag shall deallocate its ResponseBuffer.

This protocol does not specify a Tag memory location for the ResponseBuffer. Also, because a ResponseBuffer is not writeable by an Interrogator, this protocol does not specify a mechanism for an Interrogator to write to it.

Figure 6-17: ResponseBuffer data storage

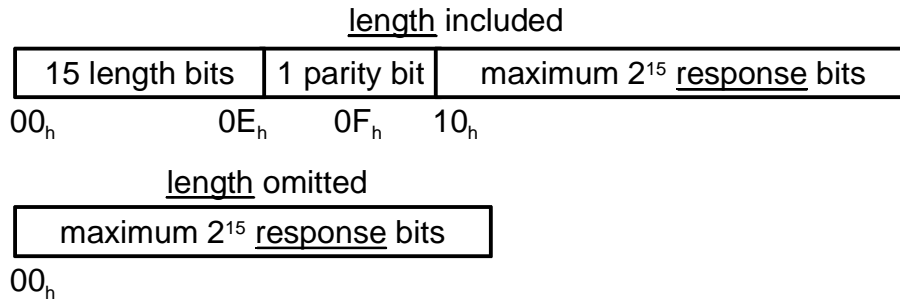


Table 6-15: Possible in-process Tag replies

| Sent or stored reply | Time for Tag to execute command? | Omit or include <u>length</u> ? | Reply | |
|----------------------|----------------------------------|---------------------------------|---|---|
| | | | Tag executed command | Tag failed to execute command |
| Sent | ≤ T _{o(max)} | Omit | Done: 1 Header: 0 Response: <u>result</u> | Done: 1 Header: 1 Response: error code |
| | | Include | Done: 1 Header: 0 Length: <u>length</u> of sent <u>result</u> Response: <u>result</u> | Done: 1 Header: 1 Length: <u>length</u> of sent error code Response: error code |
| | Any | Omit | Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 0 Response: <u>result</u> | Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 1 Response: error code |
| | | | Include | Intermediate reply or replies Done: 0 Header: 0 Length: 0000 _h Response: null Final reply Done: 1 Header: 0 Length: <u>length</u> of sent <u>result</u> Response: <u>result</u> |

| Sent or stored reply | Time for Tag to execute command? | Omit or include <u>length</u> ? | Reply | |
|----------------------|----------------------------------|---------------------------------|--|---|
| | | | Tag executed command | Tag failed to execute command |
| Stored | $\leq T_{o(max)}$ | Omit | Done: 1 Header: 0 Response: null | Done: 1 Header: 1 Response: null |
| | | Include | Done: 1 Header: 0 Length: <u>length</u> of stored <u>result</u> Response: null | Done: 1 Header: 1 Length: <u>length</u> of stored error code Response: null |
| | Any | Omit | Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 0 Response: null | Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 1 Response: null |
| | | Include | Intermediate reply or replies Done: 0 Header: 0 Length: 0000 _h Response: null Final reply Done: 1 Header: 0 Length: <u>length</u> of stored <u>result</u> Response: null | Intermediate reply or replies Done: 0 Header: 0 Length: 0000 _h Response: null Final reply Done: 1 Header: 1 Length: <u>length</u> of stored error code Response: null |

Figure 6-18: Link timing

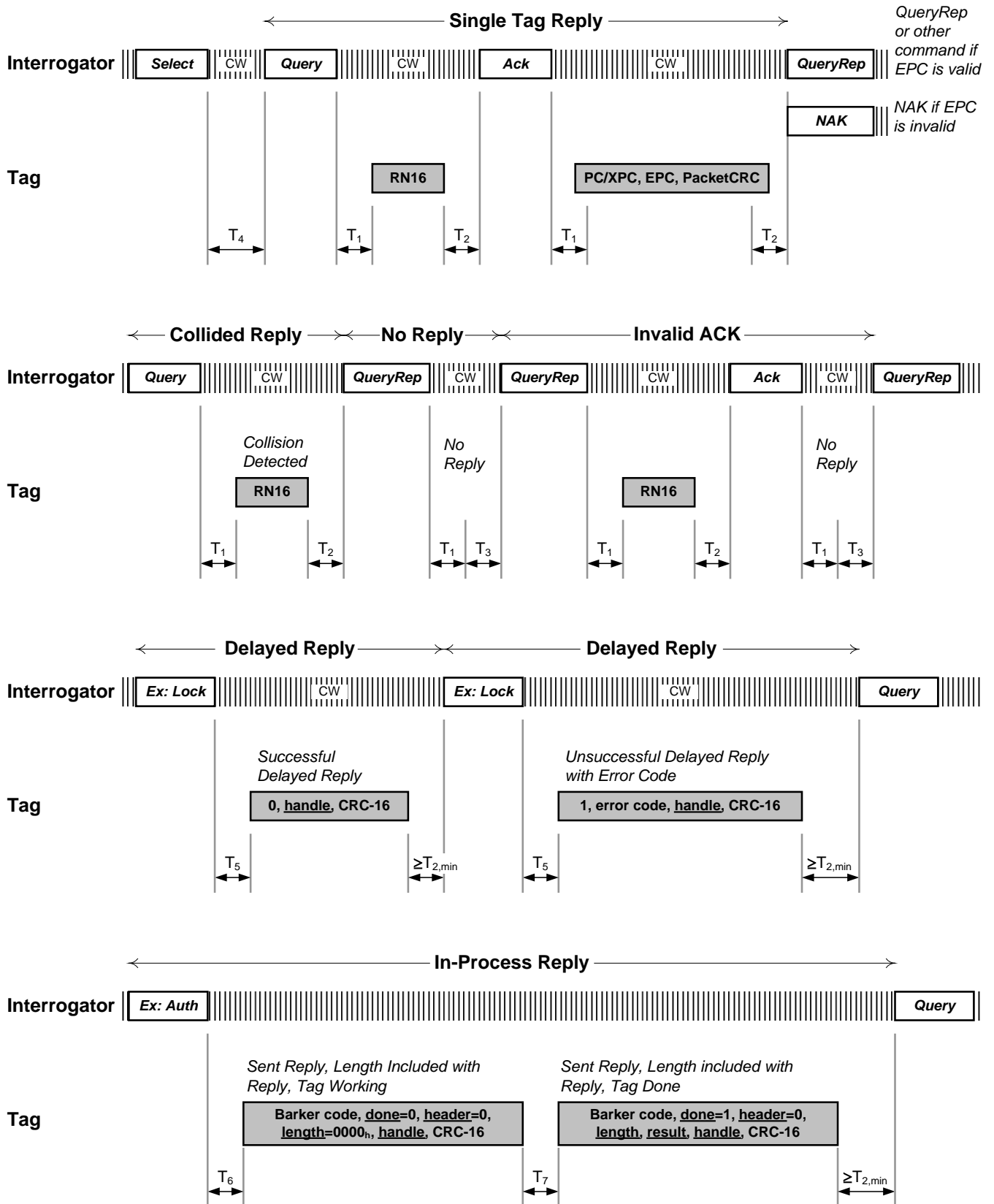


Table 6-16: Link timing parameters

| Parameter | Minimum | Nominal | Maximum | Description |
|----------------|---|--|---|---|
| T ₁ | $\text{MAX}(\text{RT}_{\text{cal}}, 10T_{\text{pri}}) \times (1 - \text{FrT}) - 2\mu\text{s}$ | $\text{MAX}(\text{RT}_{\text{cal}}, 10T_{\text{pri}})$ | $\text{MAX}(\text{RT}_{\text{cal}}, 10T_{\text{pri}}) \times (1 + \text{FrT}) + 2\mu\text{s}$ | Immediate reply time from Interrogator transmission to Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag reply for an immediate Tag reply, measured at the Tag's antenna terminals. |
| T ₂ | 3.0T _{pri} | | 20.0T _{pri} | Interrogator reply time if a Tag is to demodulate the Interrogator signal, measured from the end of the last (dummy) bit of the Tag reply to the first falling edge of the Interrogator transmission |
| T ₃ | 0.0T _{pri} | | | Time an Interrogator waits, after T ₁ , before it issues another command |
| T ₄ | 2.0 RT _{cal} | | | Minimum time between Interrogator commands |
| T ₅ | $\text{MAX}(\text{RT}_{\text{cal}}, 10T_{\text{pri}}) \times (1 - \text{FrT}) - 2\mu\text{s}$ | | 20ms | <i>Delayed</i> reply time from Interrogator transmission to Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag reply for a <i>delayed</i> Tag reply, measured at the Tag's antenna terminals. |
| T ₆ | $\text{MAX}(\text{RT}_{\text{cal}}, 10T_{\text{pri}}) \times (1 - \text{FrT}) - 2\mu\text{s}$ | | 20ms | <i>In-process</i> reply time from Interrogator transmission to the first Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the first Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals |
| T ₇ | $\text{MAX}(250\mu\text{s}, T_{2(\text{max})})$ | | 20ms | <i>In-process</i> reply time between Tag replies. Specifically, the time from the end of the last (dummy) bit of the Tag's prior transmission indicating that the Tag is still working to the first rising edge of the current Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals |

The following items apply to the requirements specified in [Table 6-16](#):

- T_{pri} denotes either the commanded period of an FMO symbol or the commanded period of a single subcarrier cycle, as appropriate.
- The maximum value for T₂ shall apply only to Tags in the **reply** or **acknowledged** states (see [6.3.2.6.3](#) and [6.3.2.6.4](#)). For a Tag in the **reply** or **acknowledged** states, if T₂ expires (i.e. reaches its maximum value):
 - Without the Tag receiving a valid command, the Tag shall transition to the **arbitrate** state (see [6.3.2.6.2](#)),
 - During the reception of a valid command, the Tag shall execute the command,
 - During the reception of an invalid command, the Tag shall transition to **arbitrate** upon determining that the command is invalid.

In all other states the maximum value for T₂ shall be unrestricted. A Tag shall be allowed a tolerance of $20.0T_{\text{pri}} \leq T_{2(\text{max})} \leq 32T_{\text{pri}}$ in determining whether T₂ has expired. "Invalid command" is defined in [6.3.2.12](#).
- An Interrogator may transmit a new command prior to interval T₂ (i.e. during a Tag response). In this case the responding Tag may ignore the new command and, in environments with limited power availability, and may undergo a power-on reset.
- FrT is the frequency tolerance specified in [Table 6-9](#).
- T₁+T₃ shall not be less than T₄.

6.3.2 Logical interface

The logical interface between an Interrogator and a Tag may be viewed as the lowest level in the data link layer of a layered network communication system. The logical interface defines Tag memory, flags, states, selection, inventory, and access.

6.3.2.1 Tag memory

Tag memory shall be logically separated into the four distinct memory banks shown in [Figure 6-19](#), each of which may comprise zero or more memory words. The memory banks are:

- **Reserved memory** shall contain the kill and and/or access passwords, if passwords are implemented on the Tag. The kill password shall be stored at memory addresses 00_h to 1F_h; the access password shall be stored at memory addresses 20_h to 3F_h. See [6.3.2.1.1](#).
- **EPC memory** shall contain a StoredCRC at memory addresses 00_h to 0F_h, a StoredPC at addresses 10_h to 1F_h, a code (such as an EPC, and hereafter referred to as an EPC) that identifies the object to which the Tag is or will be attached beginning at address 20_h, and if the Tag implements Extended Protocol Control (XPC) then either one or two XPC word(s) beginning at address 210_h. See [6.3.2.1.2](#).
- **TID memory** shall contain an 8-bit ISO/IEC 15963 allocation class identifier at memory locations 00_h to 07_h. TID memory shall contain sufficient identifying information above 07_h for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. See [6.3.2.1.3](#).
- **User memory** is optional. If a Tag implements User memory then it may partition the User memory into one or more files. If the Tag implements a single file then that file is File_0. See [6.3.2.1.4](#) and [6.3.2.11.3](#).

The logical addressing of all memory banks and User-memory files shall begin at 00_h. The physical memory map is Tag-manufacturer defined. When a Tag backscatters data from memory the order is left-to-right and bottom-to-top in [Figure 6-19](#). The backscatter shall fall on word boundaries (except for a truncated reply – see [6.3.2.12.1.1](#)).

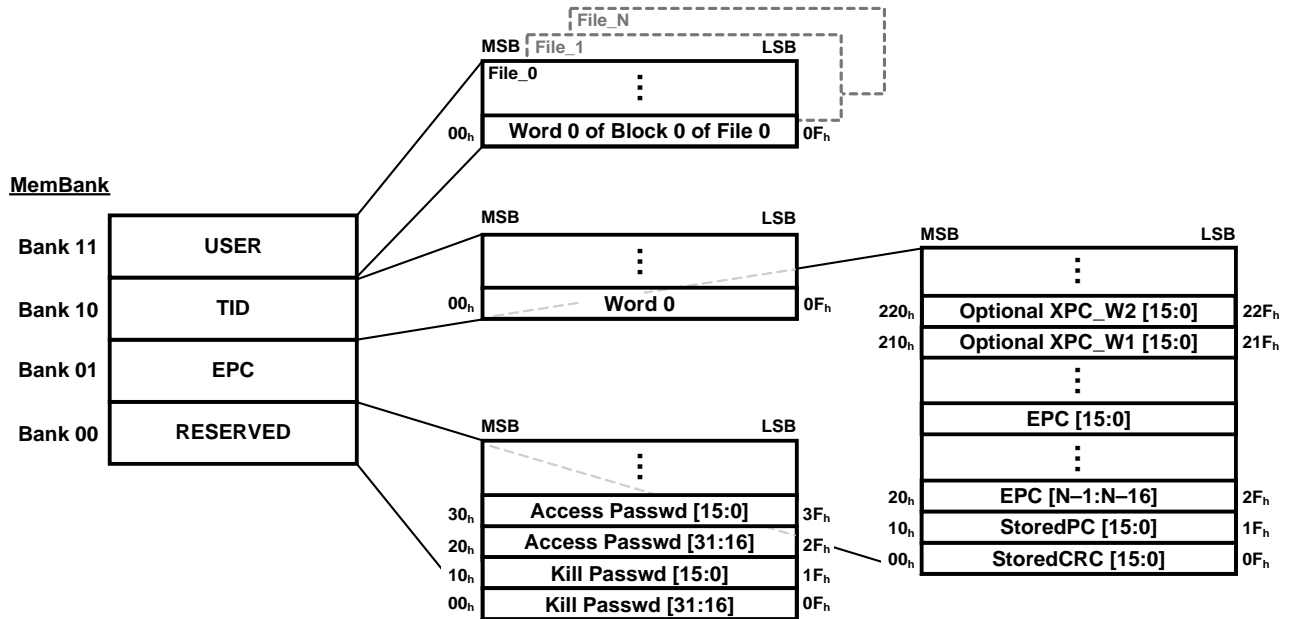
MemBank shall be defined as follows:

| | | | |
|-----------------|----------|-----------------|------|
| 00 ₂ | Reserved | 01 ₂ | EPC |
| 10 ₂ | TID | 11 ₂ | User |

Operations in one logical memory bank shall not access memory locations in another bank.

Memory writes involve the transfer of one or more 16-bit words from Interrogator to Tag. A *Write* command writes 16 bits (i.e. one word) at a time, using link cover-coding to obscure the data during R=>T transmission. The optional *BlockWrite* command writes one or more 16-bit words at a time, without link cover-coding. The optional *BlockErase* command erases one or more 16-bit words at a time. A *Write*, *BlockWrite*, or *BlockErase* shall not alter a Tag's killed status regardless of the memory address (whether valid or invalid) specified in the command.

Figure 6-19: Logical memory map



An Interrogator may issue a *Lock* command (see [6.3.2.12.3.5](#)) to lock, permanently lock, unlock, or permanently unlock the kill password, access password, EPC memory bank, TID memory bank, or File_0 of User memory, thereby preventing or allowing subsequent changes (as appropriate). If the passwords are locked or permanently locked then they are unwriteable and unreadable by any command and usable only by a *Kill* or *Access* command. If EPC memory, TID memory, or File_0 are locked or permanently locked then they are unwriteable but readable, except for the **L** and **U** bits in EPC memory; an Interrogator with an asserted *Untraceable* privilege may alter the **L** and **U** bits regardless of the lock or permalock status of EPC memory (see [6.3.2.12.3.16](#)).

If a Tag implements User memory then it partitions each File_N, $N \geq 0$ of User memory into one or more equal-size blocks. A Tag shall use the same block size for file allocation (see [6.3.2.11.3](#)) as it does for the *BlockPermalock* command (see [6.3.2.12.3.9](#)). A Tag may use different block sizes for the *BlockWrite* and *BlockErase* commands. If a Tag supports the *BlockPermalock* command then an Interrogator may issue a *BlockPermalock* to permanently lock one or more memory blocks. If blocks within File_0 are permalocked then these blocks are permanently unwriteable but readable. If blocks within File_N, $N > 0$ are permalocked then these blocks are permanently unwriteable but readable by an Interrogator with appropriate read privileges (see [Table 6-24](#) and [Table 6-25](#)).

6.3.2.1.1 Reserved Memory

Reserved memory contains the kill (see [6.3.2.1.1.1](#)) and/or access (see [6.3.2.1.1.2](#)) passwords, if passwords are implemented on a Tag. If a Tag does not implement the kill and/or access password(s) then the Tag shall logically operate as though it has zero-valued password(s) that are permanently read/write locked (see [6.3.2.12.3.5](#)), and the corresponding physical memory locations in Reserved memory need not exist.

6.3.2.1.1.1 Kill password

The kill password is a 32-bit value stored in Reserved memory 00h to 1Fh, MSB first. The default (unprogrammed) value shall be zero. A Tag that does not implement a kill password shall behave as though it has a zero-valued kill password that is permanently read/write locked. A Tag shall not execute a password-based kill if its kill password is zero (see [6.3.2.12.3.4](#)). An Interrogator may use a nonzero kill password in a password-based *Kill*-command sequence to kill a Tag and render it nonresponsive thereafter.

6.3.2.1.1.2 Access password

The access password is a 32-bit value stored in Reserved memory 20h to 3Fh, MSB first. The default (unprogrammed) value shall be zero. A Tag that does not implement an access password shall behave as though it has a zero-valued access password that is permanently read/write locked. A

Tag with a zero-valued access password transitions from the **acknowledged** state to the **secured** state upon commencing access, without first entering the **open** state. A Tag with a nonzero-valued access password transitions from the **acknowledged** state to the **open** state upon commencing access; an Interrogator may then use the access password in an *Access* command sequence to transition the Tag from the **open** to the **secured** state.

6.3.2.1.2 EPC Memory

EPC memory contains a StoredCRC at addresses 00_h to 0F_h, a StoredPC at 10_h to 1F_h, an EPC beginning at 20_h, and optional first and second XPC words at 210_h – 21F_h (XPC_W1) and 220_h – 22F_h (XPC_W2), respectively. The StoredCRC, StoredPC, EPC, and XPC word(s) shall be stored MSB first (i.e. the EPC's MSB is at location 20_h).

The StoredCRC and StoredPC are described in [6.3.2.1.2.1](#) and [6.3.2.1.2.2](#), respectively.

The EPC identifies the object to which the Tag is affixed. The EPC for GS1 EPCglobal™ Applications is described in [6.3.2.1.2.3](#); the EPC for non-EPCglobal™ Applications is described in [6.3.2.1.2.4](#). An Interrogator may issue a *Select* that includes all or part of the EPC in its *Mask*. An Interrogator may issue an *ACK* to cause a Tag to backscatter its EPC. Under certain circumstances a Tag may truncate its backscattered EPC (see [6.3.2.12.3.16](#) and [6.3.2.12.1.1](#)). An Interrogator may issue a *Read* to read all or part of the EPC.

The XPC_W1 and XPC_W2 are described in [6.3.2.1.2.5](#).

6.3.2.1.2.1 CRC-16 (StoredCRC and PacketCRC)

A Tag shall implement both a StoredCRC and a PacketCRC. The StoredCRC is stored in EPC memory, is selectable by an Interrogator using a *Select* command, and is readable by an Interrogator using a *Read* command. The PacketCRC is computed and sent by a Tag during backscatter, protects the transmitted PC/XPC and EPC, and is neither selectable nor directly readable by an Interrogator.

A Tag shall compute and store its StoredCRC either (i) when an Interrogator writes or overwrites bits in the EPC (including in the StoredPC), or (ii) every time the Tag powers up. The Tag manufacturer shall choose whether the Tag implements (i) or (ii). A Tag shall perform its computing and storing for these two cases as follows:

- i. The Tag first writes or overwrites the bits, then computes and stores a new StoredCRC, all within the reply times specified in [Table 6-16](#) for the command (*Write*, *BlockWrite*, *BlockErase*, or *Untraceable*) that wrote or overwrote the bits. A Tag shall delay backscattering the success reply shown in [Table 6-13](#) or [Table 6-14](#) for the command that wrote or overwrote the bits until it has stored the new StoredCRC. The Tag shall store its StoredCRC in nonvolatile memory so that the StoredCRC persists through subsequent Tag power cycles.
- ii. The Tag computes and stores the StoredCRC before the end of interval T_s or T_{hs} (as appropriate) in [Figure 6-3](#) or [Figure 6-5](#), respectively. The Tag may store its StoredCRC in volatile or nonvolatile memory. If an Interrogator modifies a Tag's StoredPC or EPC after Tag powerup then the StoredCRC may be incorrect until the Interrogator power-cycles the Tag.

For both cases (i) and (ii) the Tag shall implement the StoredCRC by first calculating a CRC-16 (see [6.3.1.5](#)) over the StoredPC and the EPC specified by the length (**L**) bits in the StoredPC, and then storing the thus-computed StoredCRC into EPC memory 00_h to 0F_h, MSB first. The Tag shall calculate the StoredCRC on word boundaries, shall deassert all Tag-computed StoredPC bit values (**XI** and **UMI** if Tag-computed) when performing the calculation, and shall omit XPC_W1 and XPC_W2 from the calculation.

If an Interrogator attempts to write to EPC memory 00_h – 0F_h then the Tag shall not execute the write and instead treat the command's parameters as unsupported (see [Table C-30](#)).

In response to an *ACK* a Tag backscatters a reply comprising a PC word, in some instances an XPC word or words, the EPC (which may be truncated), and a PacketCRC to protect the backscattered data stream (see [Table 6-17](#)). A Tag shall compute the PacketCRC as specified in [6.3.1.5](#) over the PC word, optional XPC word(s), and backscattered EPC, and shall send the PacketCRC MSB first.

As required by [6.3.1.5](#) an Interrogator shall verify the integrity of the received PC word, optional XPC word or words, and EPC using the PacketCRC.

In some circumstances the PacketCRC will differ from a Tag's StoredCRC, such as, for example, if the Tag has an asserted **XI** or the EPC is truncated.

6.3.2.1.2.2 Protocol-control (PC) word (StoredPC and PacketPC)

A Tag shall implement a StoredPC in addresses 10_h – $1F_h$ of EPC memory. The bit assignments for this StoredPC shall be as shown in [Table 6-18](#) and defined in [Table 6-19](#). Note that some bit assignments are different for GS1 EPCglobal (**T**=0) versus non-GS1 EPCglobal (**T**=1) Applications. Similarly, some bit assignments for XPC_W1 differ with the Application (see [6.3.2.1.2.5](#)), as does the method of computing **XI** (see below).

The StoredPC bits and values shall be as follows:

- **L (EPC length field, bits 10_h – 14_h):** Bits 10_h – 14_h are written by an Interrogator and specify the length of the EPC that a Tag backscatters in response to an *ACK*, in words:
 - 00000_2 : Zero words.
 - 00001_2 : One word (addresses 20_h to $2F_h$ in EPC memory).
 - 00010_2 : Two words (addresses 20_h to $3F_h$ in EPC memory).
 - ...
 - ...
 - 11111_2 : 31 words (addresses 20_h to $20F_h$ in EPC memory).

If a Tag only supports **XI**=0 then the maximum value for the EPC length field in the StoredPC shall be 11111_2 (allows a 496-bit EPC), as shown above. If a Tag supports **XI**=1 then the maximum value for the EPC length field in the StoredPC shall be 11101_2 (allows a 464-bit EPC). A Tag that supports **XI**=1 shall not execute a *Write*, *BlockWrite*, or *Untraceable* that attempts to write an EPC length field larger than 11101_2 and shall instead treat the command's parameters as unsupported (see [Table C-30](#)).

- **UMI (User-memory indicator, bit 15_h):** Bit 15_h may be fixed by the Tag manufacturer or computed by the Tag. In the former (fixed) case, if the Tag does not have and is incapable of allocating memory to File_0 then the Tag manufacturer shall set bit 15_h to 0_2 ; if the Tag has or is capable of allocating memory to File_0 then the Tag manufacturer shall set bit 15_h to 1_2 . In the latter (computed) case, both at power-up and upon writing the first word (bits 00_h – $0F_h$) of File_0 a Tag shall compute the logical OR of bits 03_h – 07_h of File_0 and shall map the computed value into bit 15_h ; if the Tag does not have memory allocated to File_0 then the logical OR result shall be 0_2 . Regardless of the **UMI** method (fixed or computed), when an Interrogator writes the StoredPC the Tag shall not write and instead ignore the data value the Interrogator provides for bit 15_h . For a computed **UMI**, if an Interrogator deallocates File_0 (see [6.3.2.11.3](#)) then the Tag shall set bit 15_h to 0_2 upon deallocation. Also for a computed **UMI**, the untraceability status of User memory (see [6.3.2.11.1](#)) shall not change the **UMI** value (i.e. if **UMI**=1 when a Tag is traceable then **UMI** shall remain 1 even if an Interrogator instructs a Tag to untraceably hide User memory).
- **XI (XPC_W1 indicator, bit 16_h):** If a Tag does not implement XPC_W1 then bit 16_h shall be fixed at 0_2 by the Tag manufacturer. If a Tag implements XPC_W1 then a Tag shall compute **XI** both at powerup and upon changing any bits of XPC_W1 (whether these bits are written or computed) and map the computed value into bit 16_h as follows: If **T**=0 then **XI** may be either (i) the logical OR of bits 210_h – 217_h of EPC memory or (ii) the logical OR of bits 210_h – 218_h of EPC memory; the Tag manufacturer shall choose whether the Tag implements (i) or (ii). If **T**=1 then **XI** is the logical OR of bits 210_h – $21F_h$ of EPC memory. Regardless of whether **XI** is fixed or computed, when an Interrogator writes the StoredPC the Tag shall not write and instead ignore the data value the Interrogator provides for bit 16_h .
- **T (numbering system identifier toggle, bit 17_h):** If bit 17_h is 0_2 then the application is referred to as a GS1 EPCglobal™ Application and PC bits 18_h – $1F_h$ shall be as defined in this protocol. If bit 17_h is 1_2 then the application is referred to as a non-GS1 EPCglobal™ Application and bits 18_h – $1F_h$ shall be as defined in ISO/IEC 15961.
- **RFU or AFI (Reserved for Future Use or Application Family Identifier, bits 18_h – $1F_h$):** If **T**=0 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are writeable) shall set these bits to 00_h . If **T**=1 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are

writeable) shall set these bits as specified in ISO/IEC 15961.

If an Interrogator changes the EPC length (via a memory write operation), and if it wishes the Tag to subsequently backscatter the new EPC length, then it must write new **L** bits into the Tag's StoredPC. If an Interrogator attempts to write **L** bit values that the Tag does not support then the Tag shall not execute the write operation and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag that supports **XI**=1 shall implement a PacketPC in addition to a StoredPC. Which PC word a Tag backscatters in reply to an *ACK* shall be as defined in [Table 6-17](#). A PacketPC differs from a StoredPC in its **L** bits, which a Tag adjusts to match the length of the backscattered data that follow the PC word. Specifically, if **XI**=1 but **XEB**=0 then a Tag backscatters an XPC_W1 before the EPC, so the Tag shall add one to (i.e. increment) its **L** bits. If both **XI**=1 and **XEB**=1 then the Tag backscatters both an XPC_W1 and an XPC_W2 before the EPC, so the Tag shall add two to (i.e. double increments) its **L** bits. Because Tags that support XPC functionality have a maximum **L** value of 11101_2 , double incrementing increases the value to 11111_2 . A Tag shall not, under any circumstances, allow its **L** bits to roll over to 00000_2 . Note that incrementing or double incrementing the **L** bits does not alter the bit values stored in EPC memory $10_h - 14_h$; rather, a Tag increments the **L** bits in the backscattered PacketPC but leaves the memory contents unaltered.

A Tag that does not implement an XPC_W1 or untraceability need not implement a PacketPC.

The fields that a Tag includes in its reply to an *ACK* ([Table 6-17](#)) depend on the values of **T**, **C**, **XI**, and **XEB** (see [Table 6-19](#)); whether the Tag implements an XPC_W1; whether the Tag is truncating its reply (see [6.3.2.12.1.1](#)); and the value of *immed* (see [6.3.2.12.1.2](#)). If a Tag has **T**=0, **XI**=0, implements an XPC_W1, and is not truncating then the Tag substitutes the 8 LSBs of XPC_W1 (i.e. EPC memory $218_h - 21F_h$) for the 8 LSBs of the StoredPC (i.e. PC memory $18_h - 1F_h$) in its reply. Because a Tag calculates its PacketCRC over the backscattered data bits (see [6.3.2.1.2.1](#)), when the Tag does this substitution then it shall calculate its PacketCRC over the 8 substituted XPC_W1 LSBs rather than over the 8 StoredPC LSBs.

An Interrogator shall support Tag replies with **XI**=0, **XI** =1, or both **XI**=1 and **XEB**=1.

When sending a truncated EPC a Tag substitutes 00000_2 for its PC field — see [Table 6-17](#) and [6.3.2.12.1.1](#).

If a Tag has a *response* (*result* or error code) in its ResponseBuffer (i.e. **C**=1) and the Interrogator set *immed*=1 in the *Challenge* command that preceded the inventory round then a Tag shall concatenate *response* and a CRC-16 calculated over *response* to its reply to an *ACK* (see [Table 6-17](#)).

Table 6-17: Tag reply to an *ACK* command

| T | X I | X E B | Trun cati on | C AND <i>immed</i> | Tag Backscatter | | | | | |
|---|--------|-------------|--------------------|-----------------------|--|--------|------------------|---------------|-----------------|------------|
| | | | | | PC | XPC | EPC ¹ | CRC | Respons e | CRC |
| 0 | 0 | 0 | 0 | 0 | It Tag does not implement XPC_W1: StoredPC($10_h - 1F_h$) If Tag implements XPC_W1 (see note 2): StoredPC($10_h - 17_h$), XPC_W1($218_h -$ $21F_h$) | None | Full | PacketCR C | – | – |
| 0 | 0 | 0 | 0 | 1 | C=1 so Tag implements XPC_W1 (note 2): StoredPC($10_h - 17_h$), XPC_W1($218_h -$ $21F_h$) | None | Full | PacketCR C | <u>response</u> | CRC- 16 |
| 0 | 0 | 0 | 1 | 0 | 00000_2 | None | Truncat ed | PacketCR C | – | – |
| 0 | 0 | 0 | 1 | 1 | 00000_2 | None | Truncat ed | PacketCR C | <u>response</u> | CRC- 16 |
| 0 | 0 | 1 | All | | Disallowed ³ | | | | | |
| 0 | 1 | 0 | 0 | 0 | PacketPC | XPC_W1 | Full | PacketCR C | – | – |



| T | X I | XEB | Truncation | C AND Immed | Tag Backscatter | | | | | |
|---|-----|-----|------------|-------------|---|----------------|------------------|-----------|-----------------|--------|
| | | | | | PC | XPC | EPC ¹ | CRC | Response | CRC |
| 0 | 1 | 0 | 0 | 1 | PacketPC | XPC_W1 | Full | PacketCRC | <u>response</u> | CRC-16 |
| 0 | 1 | 0 | 1 | 0 | 00000 ₂ | None | Truncated | PacketCRC | – | – |
| 0 | 1 | 0 | 1 | 1 | 00000 ₂ | None | Truncated | PacketCRC | <u>response</u> | CRC-16 |
| 0 | 1 | 1 | 0 | 0 | PacketPC | XPC_W1, XPC_W2 | Full | PacketCRC | – | – |
| 0 | 1 | 1 | 0 | 1 | PacketPC | XPC_W1, XPC_W2 | Full | PacketCRC | <u>response</u> | CRC-16 |
| 0 | 1 | 1 | 1 | 0 | 00000 ₂ | None | Truncated | PacketCRC | – | – |
| 0 | 1 | 1 | 1 | 1 | 00000 ₂ | None | Truncated | PacketCRC | <u>response</u> | CRC-16 |
| 1 | 0 | 0 | 0 | 0 (C=0) | StoredPC(10 _n –1F _n) | None | Full | PacketCRC | – | – |
| 1 | 0 | 0 | 0 | 0 (C=1) | Disallowed ⁴ | | | | | |
| 1 | 0 | 0 | 0 | 1 | Disallowed ⁴ | | | | | |
| 1 | 0 | 0 | 1 | 0 (C=0) | 00000 ₂ | None | Truncated | PacketCRC | – | – |
| 1 | 0 | 0 | 1 | 0 (C=1) | Disallowed ⁴ | | | | | |
| 1 | 0 | 0 | 1 | 1 | Disallowed ⁴ | | | | | |
| 1 | 0 | 1 | All | | Disallowed ³ | | | | | |
| 1 | 1 | 0 | 0 | 0 | PacketPC | XPC_W1 | Full | PacketCRC | – | – |
| 1 | 1 | 0 | 0 | 1 | PacketPC | XPC_W1 | Full | PacketCRC | <u>response</u> | CRC-16 |
| 1 | 1 | 0 | 1 | 0 | 00000 ₂ | None | Truncated | PacketCRC | – | – |
| 1 | 1 | 0 | 1 | 1 | 00000 ₂ | None | Truncated | PacketCRC | <u>response</u> | CRC-16 |
| 1 | 1 | 1 | 0 | 0 | PacketPC | XPC_W1, XPC_W2 | Full | PacketCRC | – | – |
| 1 | 1 | 1 | 0 | 1 | PacketPC | XPC_W1, XPC_W2 | Full | PacketCRC | <u>response</u> | CRC-16 |
| 1 | 1 | 1 | 1 | 0 | 00000 ₂ | None | Truncated | PacketCRC | – | – |
| 1 | 1 | 1 | 1 | 1 | 00000 ₂ | None | Truncated | PacketCRC | <u>response</u> | CRC-16 |

Note 1: Full means an EPC whose length is specified by the L bits in the StoredPC; truncated means an EPC whose length is shortened by a prior *Select* command specifying truncation (see [6.3.2.12.1.1](#)).

Note 2: If a Tag has T=0, XI=0, implements an XPC_W1, and is not truncating then the Tag substitutes EPC memory bits 218_n–21F_n for EPC memory bits 18_n–1F_n in its reply to an *ACK*.

Note 3: If T=0 then XI may be either (i) the logical OR of bits 210_n–217_n of XPC_W1 or (ii) the logical OR of bits 210_n–218_n of XPC_W1; the Tag manufacturer chooses whether a Tag implements (i) or (ii). If T=1 then XI is the logical OR of the entirety of XPC_W1 (210_n–21F_n). Because XEB is the MSB (210_n) of XPC_W1, if XEB=1 then XI=1 regardless of the T value.

Note 4: If T=1 then XI is the logical OR of the entirety of XPC_W1 (210_n–21F_n), so if C=1 then XI=1.

Table 6-18: StoredPC and XPC_W1 bit assignments

StoredPC bit assignments

| Application | MSB | | | | | | | | | | | | | | | LSB |
|-------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|---------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----|
| | 10 _h | 11 _h | 12 _h | 13 _h | 14 _h | 15 _h | 16 _h | 17 _h | 18 _h | 19 _h | 1A _h | 1B _h | 1C _h | 1D _h | 1E _h | |
| GS1 EPCglobal | L4 | L3 | L2 | L1 | LO | UM I | XI | T=0 | RFU | | | | | | | |
| Non-GS1 EPCglobal | L4 | L3 | L2 | L1 | LO | UM I | XI | T=1 | AFI as defined in ISO/IEC 15961 | | | | | | | |

XPC_W1 bit assignments

| Application | MSB | | | | | | | | | | | | | | | | LSB |
|-------------------|------------------|--------------------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|-----|
| | 210 _h | 211 _h | 212 _h | 213 _h | 214 _h | 215 _h | 216 _h | 217 _h | 218 _h | 219 _h | 21A _h | 21B _h | 21C _h | 21D _h | 21E _h | 21F _h | |
| GS1 EPCglobal | XEB | RFU | | | | | | | B | C | SL I | TN | U | K | NR | H | |
| Non-GS1 EPCglobal | XEB | As defined in ISO/IEC 18000-63 | | | | | | | B | C | SL I | TN | U | K | NR | H | |

Table 6-19: StoredPC and XPC_W1 bit values

| Hex | Name | How Set?¹ | Descriptor | Settings |
|---------|------------|---------------------|-------------------------------------|--|
| 10:14 | L4:L0 | Written | EPC length field | L4:L0 encode a numeric value. See 6.3.2.1.2.2 . |
| 15 | UMI | Fixed or computed | File_0 indicator | 0: If fixed then Tag does not have and is incapable of allocating memory to File_0. If computed then File_0 is not allocated or does not contain data. 1: If fixed then Tag has or is capable of allocating memory to File_0. If computed then File_0 is allocated and contains data. |
| 16 | XI | Computed | XPC_W1 indicator | 0: Either (i) Tag has no XPC_W1, or (ii) T=0 and either bits 210 _h –217 _h or bits 210 _h –218 _h (at Tag-manufacturers option) of EPC memory are all zero, or (iii) T=1 and bits 210 _h –21F _h of EPC memory are all zero. 1: Tag has an XPC_W1 and either (i) T=0 and at least one bit of 210 _h –217 _h or 210 _h –218 _h (at Tag-manufacturer's option) of EPC memory is nonzero, or (ii) T=1 and at least one bit of 210 _h –21F _h of EPC memory is nonzero |
| 17 | T | Written | Numbering System Identifier Toggle | 0: Tag is used in a GS1 EPCglobal™ Application 1: Tag is used in a non-GS1 EPCglobal™ Application |
| 18:1F | RFU or AFI | Per the Application | RFU or AFI | GS1 EPCglobal™ Application: RFU and fixed¹ at zero Non-GS1 EPCglobal™ Application: See ISO/IEC 15961 |
| 210 | XEB | Computed | XPC_W2 indicator | 0: Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued 1: Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero |
| 211:217 | RFU | Per the Application | RFU or assigned by ISO/IEC 18000-63 | GS1 EPCglobal™ Application: RFU and fixed¹ at zero Non-GS1 EPCglobal™ Application: See ISO/IEC 18000-63 |
| 218 | B | Tag mfg defined | Battery-Assisted Passive indicator | 0: Tag is passive or does not support the B flag 1: Tag is battery-assisted |
| 219 | C | Computed | Computed response indicator | 0: ResponseBuffer is empty or Tag does not support a ResponseBuffer 1: ResponseBuffer contains a <u>response</u> |

| Hex | Name | How Set?¹ | Descriptor | Settings |
|-----|------|-----------------|------------------------|---|
| 21A | SLI | Computed | SL indicator | 0: Tag has a deasserted SL flag or does not support the SLI bit 1: Tag has an asserted SL flag |
| 21B | TN | Tag mfg defined | Notification indicator | 0: Tag does not assert a notification or does not support the TN bit 1: Tag asserts a notification |
| 21C | U | Written | Untraceable indicator | 0: Tag is traceable or does not support the U bit 1: Tag is untraceable |
| 21D | K | Computed | Killable indicator | 0: Tag is not killable by <i>Kill</i> command or does not support the K bit 1: Tag can be killed by <i>Kill</i> command. |
| 21E | NR | Written | Nonremovable indicator | 0: Tag is removable from its host item or does not support the NR bit 1: Tag is not removable from its host item |
| 21F | H | Written | Hazmat indicator | 0: Tagged item is not hazardous material or Tag does not support the H bit 1: Tagged item is hazardous material |

Note 1: “Written” indicates that an Interrogator writes the value; “computed” indicates that a Tag computes the value; “fixed” indicates that the Tag manufacturer fixes the value; “Tag mfg defined” indicates that the Tag manufacturer defines the bit settability (written, computed, or fixed). Written bits inherit the lock/permalock status of the EPC memory bank (note: An *Untraceable* command may alter **L** and/or **U** regardless of the lock/permalock status of the EPC memory bank). Computed bits are not writeable and may change despite the lock/permalock status of the EPC memory bank. Fixed bits are not writeable and not changeable in the field.

6.3.2.1.2.3 EPC for a GS1 EPCglobal™ Application

The EPC for an EPCglobal™ Application shall be as defined in the GS1 EPC Tag Data Standard.

6.3.2.1.2.4 EPC for a non-GS1 EPCglobal™ Application

The EPC for a non-EPCglobal™ Application shall be as defined in ISO/IEC 15962.

6.3.2.1.2.5 Extended Protocol Control (XPC) word or words (optional)

A Tag may implement an XPC_W1 logically located at addresses 210_h to 21F_h of EPC memory. If a Tag implements an XPC_W1 then it may additionally implement an XPC_W2 logically located at address 220_h to 22F_h of EPC memory. A Tag shall not implement an XPC_W2 without also implementing an XPC_W1. If implemented, these XPC words shall be exactly 16 bits in length and stored MSB first. If a Tag does not support one or both of these XPC words then the specified memory locations need not exist. When an Interrogator writes the XPC a Tag shall not write and shall instead ignore the data values the Interrogator provides for the **C**, **SLI**, and **K** flags.

A Tag shall not implement any non-XPC memory element at EPC memory locations 210_h to 22F_h, inclusive. This requirement shall apply both to Tags that support an XPC word or words and to those that do not.

If a Tag implements an XPC_W1 then the Tag shall compute **XI** as described in [6.3.2.1.2.2](#). If a Tag implements an XPC_W2 then the Tag shall compute **XEB** as the logical OR of bits 220_h to 22F_h of EPC memory, inclusive. A Tag shall perform these calculations both at powerup and upon changing any bits 220_h to 22F_h of EPC memory. A Tag shall perform its **XEB** calculation prior to performing its **XI** calculation so that if **XEB**=1 then **XI**=1.

If a Tag computes a bit in XPC_W1 or XPC_W2 and, as a result of a commanded operation, the Tag alters the bit value then the Tag shall map the new value into memory prior to executing a subsequent command.

An Interrogator may issue a *Select* command (see [6.3.2.12.1](#)) with a *Mask* covering all or part of XPC_W1 and/or XPC_W2. An Interrogator may read a Tag’s XPC_W1 and XPC_W2 using a *Read* command (see [Q](#)).

The XPC_W1 bits and values shall be as follows (see also [Table 6-18](#) and [Table 6-19](#)):

- **XEB (bit 210_h):** If bit 210_h is 0₂ then either a Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued. If bit 210_h is 1₂ then a Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero.

- **RFU or As Defined in ISO/IEC 18000-63 (bits 211_h–217_h):** If **T**=0 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are writeable) shall set bits 211_h–217_h to zero. If **T**=1 then a Tag and/or an Interrogator shall set bits 211_h–217_h as defined in ISO/IEC 18000-63.
- **B (Battery-assisted passive indicator, bit 218_h):** If bit 218_h is 0₂ then a Tag is either passive or does not support the **B** flag. If bit 218_h is 1₂ then a Tag is battery assisted.
- **C (Computed response indicator, bit 219_h):** If bit 219_h is 0₂ then the Tag's ResponseBuffer is empty or the Tag does not support a ResponseBuffer. If bit 219_h is 1₂ then a Tag has a response in its ResponseBuffer. See [6.3.1.6.4](#).
- **SLI (SL-flag indicator, bit 21A_h):** If bit 21A_h is 0₂ then a Tag has a deasserted **SL** flag or does not support an **SL** indicator. If bit 21A_h is 1₂ then a Tag has an asserted **SL** flag. Upon receiving a *Query* a Tag that implements the **SL** indicator shall map its **SL** flag into the **SLI** and shall retain this **SLI** setting until starting a subsequent inventory round.
- **TN (Tag-notification indicator, bit 21B_h):** If bit 21B_h is 0₂ then a Tag does not have a Tag notification or does not support the **TN** flag. If bit 21B_h is 1₂ then a Tag has a Tag notification. The indication provided by the **TN** bit is Tag-manufacturer defined and not specified by this protocol. A Tag manufacturer may configure the **TN** bit to be writeable, computed, or fixed. Depending on the manufacturer's implementation the **TN** bit may or may not inherit the permalock status of the EPC memory bank.
- **U (Untraceable indicator, bit 21C_h):** If bit 21C_h is 0₂ then either (i) the Tag does not support the **U** bit or (ii) an Interrogator has not asserted the **U** bit. If bit 21C_h is 1₂ then an Interrogator has asserted the **U** bit, typically for the purpose of indicating that the Tag is persistently reducing its operating range and/or is untraceably hiding memory. See [6.3.2.12.3.16](#).
- **K (Killable indicator, bit 21D_h):** If bit 21D_h is 0₂ then a Tag is not killable or does not support the **K** bit. If bit 21D_h is 1₂ then a Tag is killable. Logically, **K** is defined as:
 - **K** = [(logical OR of AuthKill privilege for all keys) OR (logical OR of all 32 bits of the kill password) OR (kill-pwd-read/write=0) OR (kill-pwd-permalock=0)]. See also [Table 6-21](#). In words:
 - If the Tag supports authenticated kill and any key has a AuthKill=1 then the Tag is killable
 - If any bits of the kill password are 1₂ then the Tag is killable
 - If **kill-pwd-read/write** (see [6.3.2.12.3.5](#)) is 0₂ then the Tag is killable
 - If **kill-pwd-permalock** (see [6.3.2.12.3.5](#)) is 0₂ then the Tag is killable
- **NR (Nonremovable indicator, bit 21E_h):** If bit 21E_h is 0₂ then a Tag is either removable or does not support the **NR** flag. If bit 21E_h is 1₂ then a Tag is nonremovable. See [4.1](#).
- **H (Hazmat indicator, bit 21F_h):** If bit 21F_h is 0₂ then a Tag is either not affixed to hazardous material or does not support the **H** flag. If bit 21F_h is 1₂ then a Tag is affixed to hazardous material.

If **T**=0 then a Tag manufacturer (if XPC_W2 exists but is not writeable) or an Interrogator (if XPC_W2 exists and is writeable) shall set all XPC_W2 bits to 0₂. If **T**=1 then a Tag and/or an Interrogator shall set the XPC_W2 bits as defined in ISO/IEC 18000-63.

6.3.2.1.3 TID Memory

TID memory locations 00_h to 07_h shall contain either an E0_h or E2_h ISO/IEC 15963 class-identifier value. The Tag manufacturer assigns the class identifier (E0_h or E2_h), for which ISO/IEC 15963 defines the registration authority. The class-identifier does not specify the Application. TID memory locations above 07_h shall be defined according to the registration authority defined by this class-identifier value and shall contain, at a minimum, sufficient information for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. TID memory may also contain Tag- and manufacturer-specific data (for example, a Tag serial number).

If the class identifier is E0_h then TID memory locations 08_h to 0F_h contain an 8-bit manufacturer identifier, TID memory locations 10_h to 3F_h contain a 48-bit Tag serial number (assigned by the Tag manufacturer), the composite 64-bit TID (i.e. TID memory 00_h to 3F_h) is unique among all classes of

Tags defined in ISO/IEC 15963, and TID memory is permalocked at the time of manufacture.

If the class identifier is E2_h then TID memory above 07_h shall be configured as follows:

- 08_h: XTID (**X**) indicator (whether a Tag implements an XTID – see [5.2](#))
- 09_h: Security (**S**) indicator (whether a Tag supports the *Authenticate* and/or *Challenge* commands)
- 0A_h: File (**F**) indicator (whether a Tag supports the *FileOpen* command)
- 0B_h to 13_h: A 9-bit Tag mask-designer identifier (obtainable from the registration authority)
- 14_h to 1F_h: A Tag-manufacturer-defined 12-bit Tag model number
- Above 1F_h: As defined in the GS1 EPC Tag Data Standard

If the class identifier is E2_h then TID memory locations 00_h to 1F_h shall be permalocked at time of manufacture. If the Tag implements an XTID then the entire XTID shall also be permalocked at time of manufacture.

6.3.2.1.4 User Memory

A Tag may support User memory, configured as one or more files. User memory allows user data storage.

If File_0 of User memory exists and has not yet been written then the 5 LSBs of the first byte (i.e. File_0 memory addresses 03_h to 07_h) shall have the default value 00000₂.

6.3.2.1.4.1 User memory for a GS1 EPCglobal™ Application

If a Tag implements User memory then the file encoding shall be as defined in the GS1 EPC Tag Data Standard.

6.3.2.1.4.2 User memory for a non-GS1 EPCglobal™ Application

If a Tag implements User memory then the file encoding shall be as defined in ISO/IEC 15961 and 15962.

6.3.2.2 Sessions and inventoried flags

Interrogators shall support and Tags shall provide 4 sessions (denoted S0, S1, S2, and S3). Tags shall participate in one and only one session during an inventory round. Two or more Interrogators can use sessions to independently inventory a common Tag population. The sessions concept is illustrated in [Figure 6-20](#).

A Tag shall maintain an independent **inventoried** flag for each of its four sessions. Each **inventoried** flag has two values, denoted *A* and *B*. At the beginning of each and every inventory round an Interrogator chooses to inventory either *A* or *B* Tags in one of the four sessions. Tags participating in an inventory round in one session shall neither use nor modify an **inventoried** flag for a different session. The **inventoried** flags are the only resource that a Tag provides separately and independently to a session; all other Tag resources are shared among sessions.

After singulating a Tag an Interrogator may issue a command that causes the Tag to invert its **inventoried** flag for that session (i.e. $A \rightarrow B$ or $B \rightarrow A$).

The following example illustrates how two Interrogators can use sessions and **inventoried** flags to independently and completely inventory a common Tag population, on a time-interleaved basis:

- Interrogator #1 powers-on, then
 - It initiates an inventory round during which it singulates *A* Tags in session S2 to *B*,
 - It powers off.
- Interrogator #2 powers-on, then
 - It initiates an inventory round during which it singulates *B* Tags in session S3 to *A*,
 - It powers off.

This process repeats until Interrogator #1 has placed all Tags in session S2 into *B*, after which it inventories the Tags in session S2 from *B* back to *A*. Similarly, Interrogator #2 places all Tags in session S3 into *A*, after which it inventories the Tags in session S3 from *A* back to *B*. By this multi-step procedure each Interrogator can independently inventory all Tags in its field, regardless of the initial state of their **inventoried** flags.

A Tag's **inventoried** flags shall have the set and persistence times shown in [Table 6-20](#). A Tag shall power-up with its **inventoried** flags set as follows:

- The S0 **inventoried** flag shall be set to *A*.
- The S1 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the flag was set longer in the past than its persistence time, in which case the Tag shall power-up with its S1 **inventoried** flag set to *A*. Because the S1 **inventoried** flag is not automatically refreshed, it may revert from *B* to *A* even when the Tag is powered.
- The S2 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the Tag has lost power for a time greater than its persistence time, in which case the Tag shall power-up with the S2 **inventoried** flag set to *A*.
- The S3 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the Tag has lost power for a time greater than its persistence time, in which case the Tag shall power-up with its S3 **inventoried** flag set to *A*.

A Tag shall refresh its S2 and S3 flags while powered, meaning that every time a Tag loses power its S2 and S3 **inventoried** flags shall have the set and persistence times shown in [Table 6-20](#).

A Tag shall not change the value of its S1 **inventoried** flag from *B* to *A*, as the result of a persistence timeout, while the Tag is participating in an inventory round, is in the midst of being inventoried, or is in the midst of being accessed. If a Tag's S1 flag persistence time expires during an inventory round then the Tag shall change the flag to *A* only (i) as instructed by an Interrogator (e.g. by a *QueryAdjust* or *QueryRep* with matching session at the end of an inventory or access operation), or (ii) at the end of the round (e.g. upon receiving a *Select* or *Query*). In case (i), if the Tag's S1 flag persistence time expires while the Tag is in the midst of being inventoried or accessed then the Tag shall change the flag to *A* at the end of the inventory or access operation. In case (ii), the Tag shall invert its S1 flag prior to evaluating the *Select* or *Query*.

6.3.2.3 Selected flag

A Tag shall implement a selected flag, **SL**, which an Interrogator may assert or deassert using a *Select* command. The Sel parameter in the *Query* command allows an Interrogator to inventory Tags that have **SL** either asserted or deasserted (i.e. **SL** or **~SL**), or to ignore the flag and inventory Tags regardless of their **SL** value. **SL** is not associated with any particular session; **SL** may be used in any session, and is common to all sessions.

A Tag's **SL** flag shall have the set and persistence times shown in [Table 6-20](#). A Tag shall power-up with its **SL** flag either asserted or deasserted, depending on the stored value, unless the Tag has lost power for a time greater than the **SL** persistence time, in which case the Tag shall power-up with its **SL** flag deasserted (set to **~SL**). A Tag shall refresh its **SL** flag when powered, meaning that every time a Tag loses power its **SL** flag shall have the persistence times shown in [Table 6-20](#).

6.3.2.4 C flag

A Tag's **C** flag (see [6.3.2.1.2.5](#)) shall have the set and persistence times shown in [Table 6-20](#). A Tag retains data in its ResponseBuffer (see [6.3.1.6.4](#)) with the same persistence as its **C** flag. A Tag shall refresh its **C** flag when powered, meaning that every time a Tag loses power its **C** flag shall have the persistence shown in [Table 6-20](#) (of course, if a Tag has a zero-second persistence time then even if the Tag powers down momentarily its **C** flag will be deasserted).

6.3.2.5 Security timeout

A Tag may implement a security timeout after a failed *Access*-command sequence, authenticated *Kill*, password-based *Kill*-command sequence, *Challenge*, *Authenticate*, *SecureComm*, *AuthComm*, and/or *KeyUpdate*. During a security timeout a Tag may participate in an inventory round and access, but until the end of the timeout the Tag does not execute those commands for which it implements a security timeout and instead backscatters an error code (see [Annex J](#)). If a Tag implements a security timeout then it shall use a single timeout timer, so a security timeout caused

by one command failure (such as a failed *Challenge*) shall cause a Tag to disallow all commands for which the Tag implements a security timeout until the end of the timeout period. Although this protocol gives Tag manufacturers the option of choosing which commands are subject to a security timeout, it recommends that Tags implement a security timeout at least for the *Access*-command sequence. This protocol further recommends that a Tag's security timer have the set and persistence times shown in [Table 6-20](#).

6.3.2.6 Tag states and slot counter

A Tag shall implement the states and slot counter shown in [Figure 6-21](#). Note that the states in [Figure 6-21](#) are metastates that characterize a Tag's behavior and response to Interrogator commands; an actual Tag realization is likely to have more internal states than the metastates shown in the [Figure 6-21](#). [Annex B](#) shows the associated state-transition tables; [Annex C](#) shows the associated command-response tables.

6.3.2.6.1 Ready state

Tags shall implement a **ready** state. **Ready** can be viewed as a "holding state" for energized Tags that are neither killed nor currently participating in an inventory round. Upon entering an energizing RF field a Tag that is not killed shall enter **ready**. The Tag shall remain in **ready** until it receives a *Query* command (see [6.3.2.12.2.1](#)) whose *inventoried* parameter (for the *session* specified in the *Query*) and *sel* parameter match its current flag values. Matching Tags shall draw a *Q*-bit number from their RNG (see [6.3.2.7](#)), load this number into their slot counter, and transition to the **arbitrate** state if the number is nonzero, or to the **reply** state if the number is zero. If a Tag in any state except **killed** loses power then it shall return to **ready** upon regaining power.

Figure 6-20: Session diagram

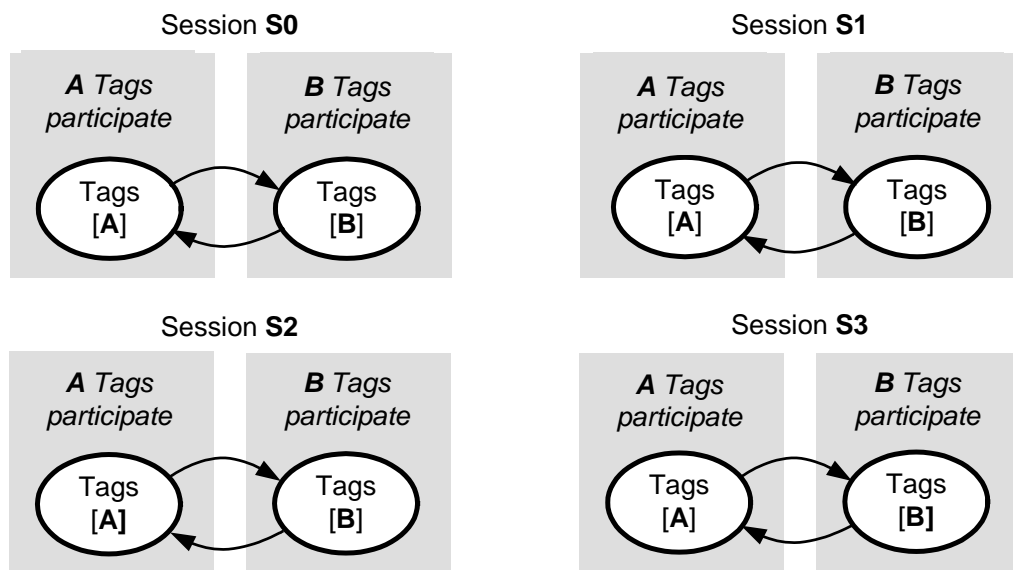


Table 6-20: Tag flags and persistence values

| Flag | Time to Set | Required persistence |
|---------------------|---|--|
| S0 inventoried flag | $\leq 2\text{ms}$ regardless of initial or final value ³ | Tag energized: Indefinite Tag not energized: None |

| Flag | Time to Set | Required persistence |
|----------------------------------|---|--|
| S1 inventoried flag ¹ | $\leq 2\text{ms}$ regardless of initial or final value ³ | Tag energized: Nominal temperature range: $500\text{ms} < \text{persistence} < 5\text{s}$ Extended temperature range: Not specified Tag not energized: Nominal temperature range: $500\text{ms} < \text{persistence} < 5\text{s}$ Extended temperature range: Not specified |
| S2 inventoried flag ¹ | $\leq 2\text{ms}$ regardless of initial or final value ³ | Tag energized: Indefinite Tag not energized: Nominal temperature range: $2\text{s} < \text{persistence}$ Extended temperature range: Not specified |
| S3 inventoried flag ¹ | $\leq 2\text{ms}$ regardless of initial or final value ³ | Tag energized: Indefinite Tag not energized: Nominal temperature range: $2\text{s} < \text{persistence}$ Extended temperature range: Not specified |
| Selected (SL) flag ¹ | $\leq 2\text{ms}$ regardless of initial or final value ³ | Tag energized: Indefinite Tag not energized: Nominal temperature range: $2\text{s} < \text{persistence}$ Extended temperature range: Not specified |
| C flag ^{1,2} | Deassert: $\leq 2\text{ms}$ ³ Assert: $\leq 0\text{ms}$ measured relative to the first rising edge of the Tag's response indicating that the Tag has finished its computation | Tag energized: Indefinite Tag not energized: Nominal temperature range: $0\text{s} \leq \text{persistence} < 5\text{s}$ Extended temperature range: Not specified |
| Optional security timeout | $< T_{1(\text{min})}$ (see Table 6-16), measured relative to the last rising edge of the last bit of the Interrogator command that caused the security timeout. | Tag energized ⁴ : Nominal temperature range: $20\text{ms} \leq \text{persistence} < 200\text{ms}$ Extended temperature range: Not specified Tag not energized ⁴ : Nominal temperature range: $20\text{ms} \leq \text{persistence} < 200\text{ms}$ Extended temperature range: Not specified |

Note 1: For a randomly chosen and sufficiently large Tag population, 95% of the Tag persistence times shall meet the persistence requirement, with a 90% confidence interval.

Note 2: A Tag retains data in its ResponseBuffer with the same persistence as its **C** flag (see [6.3.1.6.4](#)).

Note 3: Measured from the last rising edge of the last bit of the Interrogator transmission that caused the change.

Note 4: The indicated persistence times are recommended but not required.

6.3.2.6.2 Arbitrate state

Tags shall implement an **arbitrate** state. **Arbitrate** can be viewed as a “holding state” for Tags that are participating in the current inventory round but whose slot counters (see [6.3.2.6.8](#)) hold nonzero values. A Tag in **arbitrate** shall decrement its slot counter every time it receives a *QueryRep* command (see [6.3.2.12.2.3](#)) whose *session* parameter matches the session for the inventory round currently in progress, and it shall transition to the **reply** state and backscatter an RN16 when its slot counter reaches 0000_{h} . Tags that return to **arbitrate** (for example, from the **reply** state) with a slot value of 0000_{h} shall decrement their slot counter from 0000_{h} to $7FFF_{\text{h}}$ at the next *QueryRep* (with matching *session*) and, because their slot value is now nonzero, shall remain in **arbitrate**.

6.3.2.6.3 Reply state

Tags shall implement a **reply** state. Upon entering **reply** a Tag shall backscatter an RN16. If the Tag receives a valid acknowledgement (*ACK*) then it shall transition to the **acknowledged** state, backscattering the reply shown in [Table 6-17](#). If the Tag fails to receive an *ACK* within time $T_{2(\text{max})}$,

or receives an invalid *ACK* or an *ACK* with an erroneous RN16 then it shall return to **arbitrate**. Tag and Interrogator shall meet all timing requirements specified in [Table 6-16](#).

6.3.2.6.4 Acknowledged state

Tags shall implement an **acknowledged** state. A Tag in **acknowledged** may transition to any state except **killed**, depending on the received command (see [Figure 6-21](#)). If a Tag in the **acknowledged** state receives a valid *ACK* containing the correct RN16 then it shall re-backscatter the reply shown in [Table 6-17](#). If a Tag in the **acknowledged** state fails to receive a valid command within time $T_{2(\max)}$ then it shall return to **arbitrate**. Tag and Interrogator shall meet all timing requirements specified in [Table 6-16](#).

6.3.2.6.5 Open state

Tags shall implement an **open** state. A Tag in the **acknowledged** state whose access password is nonzero shall transition to **open** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the Interrogator shall use in subsequent commands and the Tag shall use in subsequent replies. A Tag in the **open** state may execute some access commands – see [Table 6-27](#). A Tag in **open** may transition to any state except **acknowledged**, depending on the received command (see [Figure 6-21](#)). If a Tag in the **open** state receives a valid *ACK* containing the correct handle then it shall re-backscatter the reply shown in [Table 6-17](#). Tag and Interrogator shall meet all timing requirements specified in [Table 6-16](#) except $T_{2(\max)}$; in the **open** state the maximum delay between Tag response and Interrogator transmission is unrestricted.

6.3.2.6.6 Secured state

Tags shall implement a **secured** state. A Tag in the **acknowledged** state whose access password is zero shall transition to **secured** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the Interrogator shall use in subsequent commands and the Tag shall use in subsequent replies. A Tag in the **open** state shall transition to **secured** following a successful *Access* command sequence or Interrogator authentication (where success in the latter case is defined by the cryptographic suite specified in the *Authenticate* command that initiated the authentication), maintaining the same handle that it previously backscattered when it transitioned from the **acknowledged** state to the **open** state. A Tag in the **secured** state with the appropriate Tag and file privileges (see [6.3.2.11.2](#) and [6.3.2.11.3](#)) may execute all access commands. A Tag in **secured** may transition to any state except **acknowledged**, depending on the received command (see [Figure 6-21](#)). If a Tag in the **secured** state receives a valid *ACK* containing the correct handle then it shall re-backscatter the reply shown in [Table 6-17](#). Tag and Interrogator shall meet all timing requirements specified in [Table 6-16](#) except $T_{2(\max)}$; in the **secured** state the maximum delay between Tag response and Interrogator transmission is unrestricted.

6.3.2.6.7 Killed state

Tags shall implement a **killed** state. A Tag in either the **open** or **secured** state shall enter the **killed** state upon receiving a successful password-based *Kill*-command sequence with a correct nonzero kill password and handle. A Tag in the **secured** states shall enter the **killed** state upon a successful authenticated *Kill* (see [6.3.2.12.3.4](#)). *Kill* permanently disables a Tag. Upon entering the **killed** state a Tag shall notify the Interrogator that the kill was successful and shall not respond to an Interrogator thereafter. Killed Tags shall remain in the **killed** state under all circumstances, and shall immediately enter killed upon subsequent power-ups. Killing a Tag is irreversible.

6.3.2.6.8 Slot counter

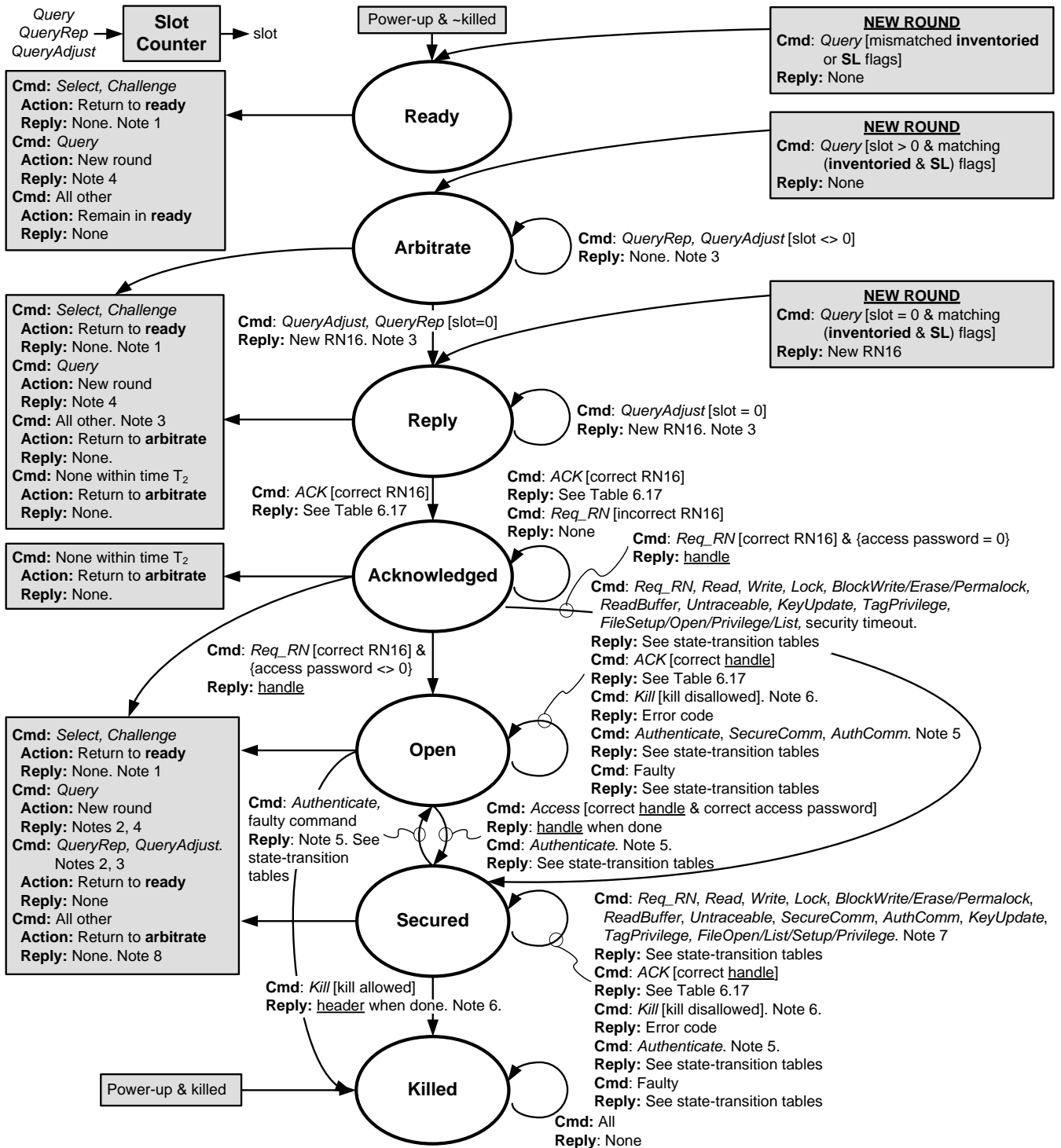
Tags shall implement a 15-bit slot counter. Upon receiving a *Query* or *QueryAdjust* command a Tag shall preload into its slot counter a value between 0 and $2^Q - 1$, drawn from the Tag's RNG (see [6.3.2.7](#)). Q is an integer in the range (0, 15). A *Query* specifies Q ; a *QueryAdjust* may modify Q from the prior *Query*.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *QueryRep* with matching session, transitioning to the **reply** state and backscattering an RN16 when their slot counter reaches 0000_h . Tags whose slot counter reached 0000_h , who replied, and who were not acknowledged (including Tags that responded to an original *Query* and were not acknowledged) shall return to **arbitrate** with a slot value of 0000_h and shall decrement this slot value from 0000_h to $7FFF_h$ at the next *QueryRep*. The slot counter shall be capable of continuous counting, meaning



that, after the slot counter rolls over to $7FFF_h$ it begins counting down again, thereby effectively preventing subsequent replies until the Tag loads a new random value into its slot counter. See also [Annex J](#).

Figure 6-21: Tag state diagram



NOTES

1. *Select*: Assert/deassert **SL** or set **inventoried** to A or B.
Challenge: Perform action(s) indicated by message, store result, and assert **C** flag in XPC_W1.
2. *Query*: A → B or B → A if the new session matches the prior session; otherwise no change to the **inventoried** flag.
QueryRep/QueryAdjust: A → B or B → A if session matches that of the prior *Query*.
3. If the command is a *QueryRep* or *QueryAdj* and session does not match that of the prior *Query* than the Tag ignores the command.
4. *Query* starts a new round and may change the session. Tags may go to **ready**, **arbitrate**, or **reply**.
5. See the state-transition tables and the cryptographic suite for conditions, message formatting, tag responses, and state changes.
6. Whether a kill is allowed or disallowed depends on the kill pwd, Tag privileges, and security timeout. See the *Kill* command-response table.
7. If the Interrogator is authenticated then certain commands require encapsulation in an *AuthComm* or a *SecureComm*. See Table 6.28.
8. A Tag that returns to **arbitrate** as a result of an unsuccessful access or kill, or a cryptographic error, may set a security timeout. See 6.3.2.5.

6.3.2.7 Tag random or pseudo-random number generator

A Tag shall implement a random or pseudo-random number generator (RNG). The RNG shall meet the following randomness criteria independent of the strength of the energizing RF field, the R=>T link rate, and the data stored in the Tag (including but not limited to the StoredPC, XPC word or words, EPC, and StoredCRC). Tags shall generate 16-bit random or pseudo-random numbers (RN16) using the RNG, and shall have the ability to extract *Q*-bit subsets from its RNG to preload the Tag's slot counter (see 6.3.2.6.8). Tags shall have the ability to temporarily store at least two RN16s while powered, to use, for example, as a handle and a 16-bit cover-code during password transactions (see Figure 6-24 or Figure 6-26).

- **Probability of a single RN16:** The probability that any RN16 drawn from the RNG has value $RN16 = j$, for any j , shall be bounded by $0.8/2^{16} < P(RN16 = j) < 1.25/2^{16}$.
- **Probability of simultaneously identical sequences:** For a Tag population of up to 10,000 Tags, the probability that any two or more Tags simultaneously generate the same sequence of RN16s shall be less than 0.1%, regardless of when the Tags are energized.
- **Probability of predicting an RN16:** An RN16 drawn from a Tag's RNG 10ms after the end of T_r in Figure 6-3 shall not be predictable with a probability greater than 0.025% if the outcomes of prior draws from the RNG, performed under identical conditions, are known.

This protocol recommends that Interrogators wait 10ms after T_r in Figure 6-3 or T_{hr} in Figure 6-5 before issuing passwords to Tags.

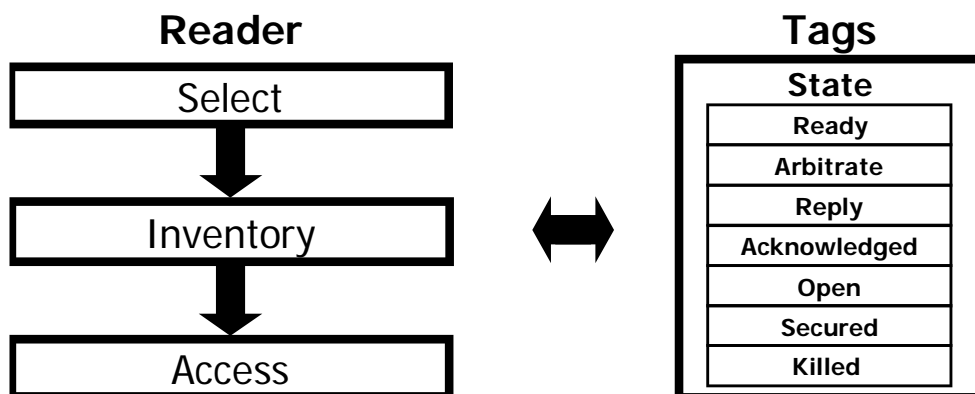
A cryptographic suite defines RNG requirements and randomness criteria for cryptographic operations. These requirements and criteria may be different, and in particular may be more stringent, than those defined above for inventory and password operations.

6.3.2.8 Managing Tag populations

Interrogators manage Tag populations using the three basic operations shown in Figure 6-22. Each of these operations comprises multiple commands. The operations are defined as follows:

- a. **Select:** The process by which an Interrogator selects a Tag population for subsequent inventory or cryptographically challenges a Tag population for subsequent authentication. Select comprises the *Select* and *Challenge* commands.
- b. **Inventory:** The process by which an Interrogator identifies Tags. An Interrogator begins an inventory round by transmitting a *Query* command in one of four sessions. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the PC, optional XPC word(s), EPC, and CRC-16 from the Tag. An inventory round operates in one and only one session at a time. Annex E shows an example of an Interrogator inventorying and accessing a single Tag. Inventory comprises multiple commands.
- c. **Access:** The process by which an Interrogator transacts with (reads, writes, authenticates, or otherwise engages with) an individual Tag. An Interrogator singulates and uniquely identifies a Tag prior to access. Access comprises multiple commands.

Figure 6-22: Interrogator/Tag operations and Tag state



6.3.2.9 Selecting Tag populations

The select process comprises two commands, *Select* and *Challenge*. *Select* allows an Interrogator to select a Tag population for subsequent inventorying. *Challenge* allows an Interrogator to challenge a Tag population for subsequent authentication. *Select* and *Challenge* are the only two commands that an Interrogator may issue prior to inventory, and they are not mutually exclusive (i.e. an Interrogator may issue both a *Select* and a *Challenge* prior to starting an inventory round). *Select* is a mandatory command; *Challenge* is optional.

A *Select* command allows an Interrogator to select a particular Tag population prior to inventorying. The selection is based on user-defined criteria, enabling union (\cup), intersection (\cap), and negation (\sim) based Tag partitioning. Interrogators perform \cup and \cap operations by issuing successive *Select* commands. *Select* can assert or deassert a Tag's **SL** flag, or it can set a Tag's **inventoried** flag to either *A* or *B* in any one of the four sessions.

Upon receiving a *Select* a not-killed Tag returns to the **ready** state, evaluates the criteria, and depending on the evaluation may modify the indicated **SL** or **inventoried** flag. A *Query* command uses these flags to choose which Tags participate in a subsequent inventory round. An Interrogator may inventory and access **SL** or \sim **SL** Tags, or it may choose to not use the **SL** flag at all. *Select* may begin with a Tag in any state except **killed**, and ends with a Tag in **ready**.

Select contains the parameters Target, Action, MemBank, Pointer, Length, Mask, and Truncate.

- Target and Action indicate whether and how a *Select* modifies a Tag's **SL** or **inventoried** flag, and in the case of an **inventoried** flag, for which session. A *Select* that modifies the **SL** flag does not modify an **inventoried** flag, and vice versa.
- MemBank specifies if the Mask applies to EPC, TID, or User memory. *Select* commands apply to a single memory bank. Successive *Selects* may apply to different memory banks.
- Pointer, Length, and Mask: Pointer and Length describe a memory range. Mask, which is Length bits long, contains a bit string that a Tag compares against the specified memory range.
- Truncate specifies whether a Tag backscatters its entire EPC, or only that portion of the EPC immediately following Mask, when replying to an *ACK*.

A *Challenge* command allows an Interrogator to instruct multiple Tags to simultaneously yet independently precompute and store a cryptographic result for use in a subsequent authentication. Because cryptographic algorithms often require significant computation time, parallel precomputation may significantly accelerate the authentication of a Tag population. *Challenge* contains an immed parameter which, if asserted, instructs the Tags to concatenate their response (result or error code) to the EPC backscattered in reply to an *ACK*.

Upon receiving a *Challenge* a not-killed Tag that supports the command returns to the **ready** state, evaluates the command (including whether it supports the CSI specified in the *Challenge*), and depending on the evaluation may compute and store a cryptographic result in its ResponseBuffer. In some instances a Tag may use the stored result during a subsequent authentication. In such instances the Interrogator will transmit a subsequent *Authenticate* command (see [6.3.2.12.3.10](#)) to the previously challenged Tag. In other instances the Tag's stored result may be usable without a subsequent *Authenticate*. For an example of the latter case, in some cryptographic suites an Interrogator can verify a Tag's authenticity simply by evaluating the precomputed result. *Challenge* may begin with a Tag in any state except **killed**, and ends with a Tag in **ready**.

6.3.2.10 Inventorying Tag populations

The inventory command set includes *Query*, *QueryAdjust*, *QueryRep*, *ACK*, and *NAK*. *Query* initiates an inventory round and decides which Tags participate in the round ("inventory round" is defined in [4.1](#)).

Query contains a slot-count parameter *Q*. Upon receiving a *Query* participating Tags pick a random value in the range $(0, 2^Q - 1)$, inclusive, and load this value into their slot counter. Tags that pick a zero transition to the **reply** state and reply immediately. Tags that pick a nonzero value transition to the **arbitrate** state and await a *QueryAdjust* or *QueryRep* command. Assuming a single Tag replies, the query-response algorithm proceeds as follows:

- a. The Tag backscatters an RN16 as it enters **reply**,
- b. The Interrogator acknowledges the Tag with an *ACK* containing this same RN16,

- c. The acknowledged Tag transitions to the **acknowledged** state, backscattering a reply as in [Table 6-17](#),
- d. The Interrogator issues a *QueryAdjust* or *QueryRep*, causing the identified Tag to invert its **inventoried** flag (i.e. $A \rightarrow B$ or $B \rightarrow A$) and transition to **ready**, and potentially causing another Tag to initiate a query-response dialog with the Interrogator, starting in step (a), above.

If the Tag fails to receive the *ACK* in step (b) within time T_2 (see [Figure 6-18](#)), or receives the *ACK* with an erroneous RN16, then it returns to **arbitrate**.

If multiple Tags reply in step (a) but the Interrogator, by detecting and resolving collisions at the waveform level, can resolve an RN16 from one of the Tags, the Interrogator can *ACK* the resolved Tag. Unresolved Tags receive erroneous RN16s and return to **arbitrate** without backscattering the reply shown in [Table 6-17](#).

If the Interrogator sends a valid *ACK* (i.e. an *ACK* containing the correct RN16) to the Tag in the **acknowledged** state, the Tag re-backscatters the reply shown in [Table 6-17](#).

At any point the Interrogator may issue a *NAK*, in response to which all Tags in the inventory round that receive the *NAK* return to **arbitrate** without changing their **inventoried** flag.

After issuing a *Query* to initiate an inventory round, the Interrogator typically issues one or more *QueryAdjust* or *QueryRep* commands. *QueryAdjust* repeats a previous *Query* and may increment or decrement Q , but does not introduce new Tags into the round. *QueryRep* repeats a previous *Query* without changing any parameters and without introducing new Tags into the round. An inventory round can contain multiple *QueryAdjust* or *QueryRep* commands. At some point the Interrogator will issue a new *Query*, thereby starting a new inventory round.

Tags in the **arbitrate** or **reply** states that receive a *QueryAdjust* first adjust Q (increment, decrement, or leave unchanged), then pick a random value in the range $(0, 2^Q - 1)$, inclusive, and load this value into their slot counter. Tags that pick zero transition to the **reply** state and reply immediately. Tags that pick a nonzero value transition to the **arbitrate** state and await a *QueryAdjust* or a *QueryRep* command.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *QueryRep*, transitioning to the **reply** state and backscattering an RN16 when their slot counter reaches 0000_h . Tags whose slot counter reached 0000_h , who replied, and who were not acknowledged (including Tags that responded to the original *Query* and were not acknowledged) return to **arbitrate** with a slot value of 0000_h and decrement this slot value from 0000_h to $7FFF_h$ at the next *QueryRep*, thereby effectively preventing subsequent replies until the Tag loads a new random value into its slot counter.

Although Tag inventory is based on a random protocol, the Q -parameter affords network control by allowing an Interrogator to regulate the probability of Tag responses. Q is an integer in the range $(0, 15)$; thus, the associated Tag-response probabilities range from $2^0 = 1$ to $2^{-15} = 0.000031$.

[Annex D](#) describes an exemplary Interrogator algorithm for choosing Q .

The scenario outlined above assumed a single Interrogator operating in a single session. However, as described in [6.3.2.2](#), an Interrogator can inventory a Tag population in one of four sessions. Furthermore, as described in [6.3.2.12.2](#), the *Query*, *QueryAdjust*, and *QueryRep* commands each contain a session parameter. How a Tag responds to these commands varies with the command, session parameter, and Tag state, as follows:

- *Query*: A *Query* command starts an inventory round and chooses the session for the round. Tags in any state except **killed** execute a *Query*, starting a new round in the specified session and transitioning to **ready**, **arbitrate**, or **reply**, as appropriate (see [Figure 6-21](#)).
 - If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter matches the prior session then it inverts its **inventoried** flag (i.e. $A \rightarrow B$ or $B \rightarrow A$) for the session before it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.
 - If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter does not match the prior session then it leaves its **inventoried** flag for the prior session unchanged as it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.
- *QueryAdjust*, *QueryRep*: Tags in any state except **ready** or **killed** execute a *QueryAdjust* or *QueryRep* command if, and only if, (i) the session parameter in the command matches the

session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively). Tags ignore a *QueryAdjust* or *QueryRep* with mismatched session.

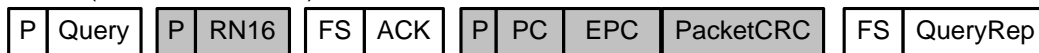
- If a Tag in the **acknowledged**, **open**, or **secured** states receives a *QueryAdjust* or *QueryRep* whose session parameter matches the session parameter in the prior *Query*, and the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively), then it inverts its **inventoried** flag (i.e. $A \rightarrow B$ or $B \rightarrow A$) for the current session and then transitions to **ready**.

Figure 6-23: One Tag reply

| Condition | Description | Symbol | Description |
|--------------|--|--------|---------------------------------|
| <u>immed</u> | <u>immed</u> field in <i>Challenge</i> | P | Preamble (R=>T or T=>R) |
| C | C flag in XPC_W1 | FS | Frame-Sync |
| XI | XI flag in StoredPC | RN16 | 16-bit Random Number |
| | | PC | Either StoredPC or PacketPC |
| | | XPC | Optional XPC word or words |
| | | result | ResponseBuffer contents |
| | | CRC-16 | 16-bit CRC computed over result |

| | |
|--|----------------|
| | R=>T Signaling |
| | T=>R Signaling |

XI=0, (immed=0 OR **C**=0)



XI=1, (immed=1 AND **C**=1)



To illustrate an inventory operation, consider a specific example: Assume a population of 64 powered Tags in the **ready** state. An Interrogator first issues a *Select* to select a subpopulation of Tags. Assume that 16 Tags match the selection criteria. Further assume that 12 of the 16 selected Tags have their **inventoried** flag set to *A* in session *S0*. The Interrogator issues a *Query* specifying (**SL**, *Q* = 4, *S0*, *A*). Each of the 12 Tags picks a random number in the range (0,15) and loads the value into its slot counter. Tags that pick a zero respond immediately. The *Query* has 3 possible outcomes:

- No Tags reply:** The Interrogator may issue another *Query*, or it may issue a *QueryAdjust* or *QueryRep*.
- One Tag replies** (see Figure 6-23): The Tag transitions to the **reply** state and backscatters an RN16. The Interrogator acknowledges the Tag by sending an *ACK*. If the Tag receives the *ACK* with a correct RN16 it backscatters the reply shown in Table 6-17 and transitions to the **acknowledged** state. If the Tag receives the *ACK* with an incorrect RN16 it transitions to **arbitrate**. Assuming a successful *ACK*, the Interrogator may either access the acknowledged Tag or issue a *QueryAdjust* or *QueryRep* with matching session parameter to invert the Tag's **inventoried** flag from $A \rightarrow B$ and send the Tag to **ready** (a *Query* with matching prior-round session parameter will also invert the **inventoried** flag from $A \rightarrow B$).
- Multiple Tags reply:** The Interrogator observes a backscattered waveform comprising multiple RN16s. It may try to resolve the collision and issue an *ACK*; not resolve the collision and issue a *QueryAdjust*, *QueryRep*, or *NAK*; or quickly identify the collision and issue a *QueryAdjust* or *QueryRep* before the collided Tags have finished backscattering. In the latter case the collided Tags, not observing a valid reply within T_2 (see Figure 6-18), return to **arbitrate** and await the next *Query* or *QueryAdjust* command.

6.3.2.11 Accessing individual Tags

An Interrogator may choose to access a Tag after acknowledging it. The access commands are Req_RN, Read, Write, Lock, Kill, Access, BlockWrite, BlockErase, BlockPermalock, Authenticate, ReadBuffer, SecureComm, AuthComm, KeyUpdate, Untraceable, FileOpen, FileList, FilePrivilege, FileSetup, and TagPrivilege. A Tag shall execute access commands only in the states shown in Table 6-27. A Tag shall treat as invalid (see Table C-30) optional access commands that it does not support. See Annex K for an example of a data-flow exchange during which an Interrogator

accesses a Tag and reads its kill password.

Access always begins with an Interrogator moving a Tag from the **acknowledged** state to either the **open** or the **secured** state as follows:

Step 1: The Interrogator issues a Req_RN to the acknowledged Tag.

Step 2: The Tag generates and stores a new RN16 (denoted handle), backscatters the handle, and transitions to the **open** state if its access password is nonzero, or to the **secured** state if its access password is zero. The Interrogator may now issue further access commands.

All access commands include a Tag's handle. Upon receiving an access command a Tag verifies that the handle is correct prior to executing the command, and does not execute access commands with an incorrect handle. The handle value is fixed for the entire duration of a Tag access.

An Interrogator may issue an ACK to a Tag in the **open** or **secured** states, with the Tag's handle as the RN in the command, thereby causing the Tag to backscatter the reply shown in [Table 6-17](#). A Tag in the **open** or **secured** states that receives an ACK with an incorrect handle transitions to the **arbitrate** state without replying and without changing its inventoried flag.

As shown in [Table 6-27](#), some access commands require a prior Req_RN and some a prior authentication before execution. A Tag's response to an access command includes, at a minimum, the Tag's handle; the response may include other information as well (for example, the result of a Read). An Interrogator shall verify the correctness of the handle in a Tag's response to an access command.

The Authenticate and Access commands provide the only means to transition a Tag from the **open** state to the **secured** state. The Authenticate command or a faulty security command provide the only means to transition a Tag from the **secured** state back to the **open** state. See [Table C-18](#) and [Table C-30](#).

The privileges that a Tag in the **open** state grants to an Interrogator depend on the authorization level of the **open** state. The privileges that a Tag in the **secured** state grants to an Interrogator depend on the authorization level of the access or authentication that most recently moved the Tag to that state. An Interrogator that moved a Tag to the **secured** state using one means (for example, an Access command) may later cause the Tag to re-enter the **secured** state using a different means (for example, an Authenticate command), affording the Interrogator different privileges. See [6.3.2.11.2](#) for a discussion of privileges and keys.

A Tag may enter the **secured** state by means of a:

- *Req_RN*: If a Tag's access password is zero then the Tag transitions from the **acknowledged** state to the **secured** state at the beginning of access (i.e. upon receiving a *Req_RN*), bypassing the **open** state.
- *Access*: A Tag whose access password is nonzero transitions from the **open** or **secured** state to the **secured** state upon successfully executing an *Access*-command sequence.
- *Authenticate*: A Tag transitions from the **open** or **secured** state to the **secured** state upon successfully executing an Interrogator or mutual authentication.

A Tag may limit an Interrogator's access to the **secured** state via one or more physical mechanisms. For example, a Tag may require that its received RF power exceed a threshold before it will enter the **secured** state. This protocol does not specify such physical mechanisms but allows them at a Tag manufacturer's discretion.

An Interrogator and a Tag can communicate indefinitely in the **open** or **secured** states. The Interrogator may end the communications at any time by issuing a *Select*, *Challenge*, *Query*, *QueryAdjust*, *QueryRep*, or *NAK*. The Tag's response to a *Query*, *QueryAdjust*, or *QueryRep* is described in [6.3.2.10](#). A *NAK* causes all Tags in the inventory round to return to **arbitrate** without changing their **inventoried** flag(s).

Interrogators in some regulatory regions are required to hop frequency at periodic intervals, ending the inventory round and any access operations. Unfortunately, some cryptographic operations take longer than a hop interval to complete. This protocol allows a cryptographic suite to specify that a Tag retain one or more cryptographic state variables during a temporary power loss such as a frequency hop, and allows an Interrogator to re-acquire the Tag in a subsequent inventory round and resume the cryptographic operation.

This protocol recommends that Interrogators avoid powering-off while a Tag is in the **reply**, **acknowledged**, **open** or **secured** states. Rather, Interrogators should end (or in the case of a long cryptographic operation, suspend) their dialog with a Tag before powering off, leaving the Tag in either the **ready** or **arbitrate** state.

This protocol partitions the access commands into the subclasses Core, Security, and File Management (see also [Table 6-27](#)). The purpose of this subclass partitioning is solely for ease of discussion and the particular subclass does not convey or deny requirements to or from any access command.

6.3.2.11.1 Core access commands

The core access commands are Req_RN, Read, Write, Lock, Kill, Access, BlockWrite, BlockErase, BlockPermalock, and Untraceable. Req_RN, Read, Write, Lock, and Kill are mandatory. Access, BlockWrite, BlockErase, BlockPermalock, and Untraceable are optional. A Tag may implement one or more of the optional commands regardless of whether the Tag supports cryptographic security or file management.

A Req_RN command allows an Interrogator to (a) transition a Tag from the **acknowledged** state to the **open** or **secured** states, obtaining the Tag's handle in the process, or (b) ask a Tag in the **open** or **secured** states to backscatter a 16-bit random number.

A Read command allows an Interrogator to read Tag memory. An Interrogator may read a Tag's kill and/or access passwords depending on Tag state and the password's lock status. An Interrogator with an asserted Untraceable privilege may read EPC and TID memory, and User-memory files for which it has read privileges. An Interrogator with a deasserted Untraceable privilege may read the portions of EPC and TID memory that are not untraceably hidden and may read User-memory files for which it has read privileges if User memory is not untraceably hidden.

Table 6-21: Conditions for Killing a Tag

| Tag supports authenticated kill? | Kill Pwd | Kill-Pwd Locked? | Tag Killable ? | How? |
|---|----------|---------------------------------|----------------|---|
| No | Zero | Permalocked | No | – |
| | | Locked, unlocked, permaunlocked | Yes | Write nonzero kill pwd then use <i>Kill</i> command. If kill pwd is locked and access pwd is nonzero then requires access before writing new kill pwd |
| | Nonzero | All | Yes | Use <i>Kill</i> command with kill pwd. |
| Yes (and at least one key has <u>AuthKill</u> =1) | Zero | Permalocked | Yes | Authenticate Interrogator then perform authenticated kill. |
| | | Locked, unlocked, permaunlocked | Yes | Authenticate Interrogator then perform authenticated kill, or write nonzero kill pwd then use <i>Kill</i> command. If kill pwd is locked and access pwd is nonzero then requires access before writing new kill pwd |
| | Nonzero | All | Yes | Authenticate Interrogator then perform authenticated kill or use <i>Kill</i> command with kill pwd. |

The *Write*, *BlockWrite*, and *BlockErase* commands allow an Interrogator to write or erase portions of Tag memory. Whether, in what states, and with what privileges an Interrogator may write/erase Tag memory is described in [Table 6-24](#), [Table 6-25](#), and [Table 6-27](#).

The *Lock* and *BlockPermalock* commands allow an Interrogator to configure portions of Tag memory to be changeably or permanently writeable or unwriteable. The EPC memory bank, TID memory bank, File_0, and the access and kill passwords may be unlocked, permanently unlocked, locked, or permanently locked for writing. Blocks within File_0 and File_N (N>0) may also be unlocked or permanently locked for writing. An *Untraceable* is the only command that can write to permanently locked memory, but its writing ability is limited to the **L** and **U** bits in EPC memory (see [6.3.2.12.3.16](#)).

An *Access* command allows an Interrogator to transition a Tag from the **open** to the **secured** state. The transition is a multi-step procedure described in [6.3.2.12.3.6](#) and outlined in [Figure 6-26](#), in

which an Interrogator sends two successive *Access* commands to a Tag. The first *Access* command contains the first half of the access password; the second *Access* command contains the second half. If a Tag receives a properly formatted *Access*-command sequence with the correct access password then it transitions to the **secured** state.

The *Untraceable* command allows an Interrogator with an asserted *Untraceable* privilege ([Table 6-22](#) and [Table 6-23](#)) to instruct a Tag to (a) overwrite the **L** bits in its StoredPC and **U** bit in XPC_W1, (b) hide part of its memory from Interrogators with a deasserted *Untraceable* privilege and/or (c) reduce its operating range. An Interrogator may use the **U** bit of XPC_W1, if supported, to indicate whether a Tag is hiding memory and/or is reducing its operating range (collectively, untraceable). An untraceable Tag behaves identically, from a command-response and state-machine perspective, to a traceable Tag, but behaves as though portions of its memory do not exist and/or as though it has reduced sensitivity. An untraceable Tag does not erase hidden memory; an Interrogator with an asserted *Untraceable* privilege may subsequently reexpose untraceably hidden memory to all Interrogators and/or reenables full operating range. An Interrogator may also subsequently overwrite the **L** and **U** bits.

A *Kill* command allows an Interrogator to kill a Tag. If a Tag's kill password is nonzero then an Interrogator may kill the Tag using the multi-step password-based *Kill*-command sequence shown in [Figure 6-24](#). If a Tag supports authenticated killing then an Interrogator that authenticated itself using a key with an *AuthKill* privilege (see [Table 6-23](#)) may kill the Tag regardless of its kill-password value (zero or nonzero) using the abbreviated, authenticated kill process shown in [Figure 6-24](#). A Tag that does not implement a kill password, or whose kill password is zero, is not killable except by the authenticated kill process. A successful *Kill* moves a Tag from the **open** or **secured** state to the **killed** state. A Tag, once killed, shall not respond to an Interrogator thereafter.

To minimize the risk of illicit Tag killing, this protocol recommends that killable Tags use either (1) unique kill passwords or (2) permalocked zero-valued kill passwords and authenticated kill. This protocol also recommends against a zero-valued access password.

The **K** flag of XPC_W1 indicates whether a Tag is killable. As shown in [Table 6-21](#), the only situation in which a Tag is not killable by over-the-air commands is if the Tag has a permalocked, zero-valued kill password and either does not support authenticated kill or does not grant the *AuthKill* privilege to any key.

The *Write*, *Kill*, and *Access* commands send 16-bit words (either data or half-passwords) from Interrogator to Tag using one-time-pad-based link cover coding to obscure the word being transmitted, as follows:

Step 1: The Interrogator issues a *Req_RN*, to which the Tag responds by backscattering a new RN16. The Interrogator then generates a 16-bit string comprising a bit-wise EXOR of the 16-bit word to be transmitted with this new RN16, both MSB first, and issues the command with this string as a parameter.

Step 2: The Tag recovers the 16-bit word by performing a bit-wise EXOR of the received 16-bit string with the original RN16.

If an Interrogator issues a command containing cover-coded data or half-password and fails to receive a response from the Tag then the Interrogator may subsequently reissue the command unchanged. If the Interrogator issues a subsequent command containing new data or a new half-password then it shall first issue a *Req_RN* to obtain a new RN16 and shall use this new RN16 for the cover-coding.

The *BlockWrite* command (see [6.3.2.12.3.7](#)) communicates multiple 16-bit words from Interrogator to Tag. Unlike a *Write*, *BlockWrite* does not use link cover coding.

Although the *Access* command uses a password, an *Access*-command sequence is not cryptographically secure. Neither Tag nor Interrogator shall consider themselves authenticated following an *Access*-command sequence. A Tag or an Interrogator shall only consider themselves authenticated after executing a cryptographic authentication in accordance with a cryptographic suite.

6.3.2.11.2 Security access commands

The security access commands are *Authenticate*, *SecureComm*, *AuthComm*, *KeyUpdate*, and *TagPrivilege*. All are optional. A Tag may implement one or more of these commands regardless of

whether the Tag supports optional core commands and/or file management. Some of these commands require prior authentication.

An *Authenticate* command may implement Tag, Interrogator, and/or mutual authentication, depending on the Tag's implementation of the cryptographic suite specified by CSI in the command. Authentication may include deriving session keys and exchanging parameters for subsequent communications. Depending on the cryptographic suite, the message field in the *Authenticate* command may include a KeyID, the type of authentication, and for some multi-step authentications the step number in the authentication sequence.

An *AuthComm* command allows authenticated R=>T communications. [Table 6-28](#) shows which commands an Interrogator may, and an authenticated Interrogator shall, encapsulate in an *AuthComm*. An *AuthComm* protects communications according to the cryptographic suite specified by CSI in the *Challenge* or *Authenticate* that preceded the *AuthComm*.

A *SecureComm* command allows secure R=>T communications. [Table 6-28](#) shows which commands an Interrogator may, and an authenticated Interrogator shall, encapsulate in a *SecureComm*. A *SecureComm* protects communications according to the cryptographic suite specified by CSI in the *Challenge* or *Authenticate* that preceded the *SecureComm*.

A *SecureComm* is configured to allow more robust security than an *AuthComm*; an *AuthComm* is configured to be faster with simplified Tag processing. This protocol recommends that Interrogators not intermix *SecureComm* and *AuthComm* commands when engaging in an authenticated dialog with a Tag.

A *KeyUpdate* command allows an authenticated Interrogator to write or change a key. If a Tag does not write the new key successfully then it defaults to the prior stored key. An Interrogator may use a *KeyUpdate* to change the key that it used during authentication; if the Interrogator has an asserted CryptoSuperuser privilege (see [Table 6-23](#)) then it may also change value(s) for other key(s) in the cryptographic suite. A cryptographic suite may place additional restrictions, beyond those specified in this protocol, on when and whether a key may be updated.

A *TagPrivilege* command allows an Interrogator to read or modify the privileges in [Table 6-22](#) or [Table 6-23](#) for the access password or for a key, respectively. Whether a Tag executes a *TagPrivilege* depends on the privilege level of the access password or the key that the Interrogator supplied during the access or authentication.

A Tag may support zero, one, or more than one cryptographic suite(s). A cryptographic suite defines how a Tag and an Interrogator implement a cryptographic algorithm and its functions. The Tag manufacturer shall choose the number and type of cryptographic suites that a Tag supports; this assignment shall not be alterable in the field. An Interrogator selects one from among the implemented cryptographic suites using the CSI field in the *Challenge* and *Authenticate* commands.

A Tag may support up to 256 keys, numbered Key_0 to Key_255. The Tag manufacturer shall choose the number of available keys and assign them to the cryptographic suite(s); this assignment shall not be alterable in the field. No two keys shall have the same number, even if used for different cryptographic suites. A Tag shall not indicate where in memory it stores its keys, nor shall it allow an Interrogator to read this memory location.

Although the functions and security of cryptographic suites may vary, this protocol anticipates that some suites may perform only Tag authentication, whereas others may perform Interrogator and/or mutual authentication.

A Tag that supports the *Untraceable* command shall provide the Tag privileges shown in [Table 6-22](#). A Tag that supports one or more cryptographic suites shall provide the Tag privileges shown in [Table 6-23](#).

The privileges field in a *TagPrivilege* command is 16 bits in length, with a bit for each corresponding Tag privilege. Privileges 12–15 in [Table 6-22](#) and [Table 6-23](#) are assigned by this protocol. Privileges 8–11 are RFU for the access password ([Table 6-22](#)); they are assigned by the cryptographic suite for all keys ([Table 6-23](#)). Privileges 4–7 are RFU for all keys. Privileges 0–3 are defined by the Tag manufacturer and not specified by this protocol.

The labels in [Table 6-22](#) and [Table 6-23](#) are defined as follows:

- **Privilege Name:** The name of the Tag privilege

- **Bit Assignment:** The bit location in the privileges field for the named privilege, MSB first (i.e. bit 15 is the leading bit in the privileges field in a *TagPrivilege* command and in a Tag's reply).
- **Privilege:** Whether a Tag grants or denies the privilege. A 1₂ means an asserted or granted privilege; a 0₂ means a deasserted or denied privilege.
- **CryptoSuperuser:** Whether a Tag grants the crypto superuser privilege to a key. If **CryptoSuperuser**=1 then a Tag grants the crypto superuser privilege to the key; if **CryptoSuperuser**=0 then a Tag denies the privilege. See below for a description of the crypto superuser privilege.
- **AuthKill:** Whether a Tag grants the authenticated-kill privilege to a key. If **AuthKill**=1 then a Tag grants the authenticated-kill privilege to the key; if **AuthKill**=0 then a Tag denies the privilege.
- **Untraceable:** Whether a Tag executes an *Untraceable* command from, and exposes untraceably hidden memory to, an Interrogator that supplies the access password or key. If **Untraceable**=1 then a Tag grants the privilege; if **Untraceable**=0 then a Tag denies the privilege.
- **DecFilePriv:** Whether a Tag allows an Interrogator that supplies the access password or key the privilege of decrementing file privileges using a *FilePrivilege* command. If **DecFilePriv**=1 in [Table 6-22](#) then a Tag permits an Interrogator that supplies the access password to decrement file privileges for the **open** state and for the access password. If **DecFilePriv**=1 in [Table 6-23](#) then a Tag permits an Interrogator that supplies the key to decrement file privileges for the **open** state and for that key. If **DecFilePriv**=0 then the Tag denies the associated privilege.
- **KeyProperty_N:** One of four key properties (N = 1, 2, 3, 4) defined by the cryptographic suite with which the key is associated.
- **Custom:** One of four key properties (N = 1, 2, 3, 4) defined by the Tag manufacturer.

A Tag that implements the *TagPrivilege* command shall permit an Interrogator that authenticated itself as a crypto superuser in a cryptographic suite to:

- change the value of any key in that cryptographic suite, including its own, using a *KeyUpdate*.
- read or modify privileges (value in [Table 6-23](#)) for any key in that cryptographic suite, including its own.

A Tag shall not permit an Interrogator that did not authenticate itself as a crypto superuser to:

- change the value of any key other than the one it used to authenticate itself.
- read or modify privileges (value in [Table 6-23](#)) for any key other than the one it used to authenticate itself
- assert a deasserted privilege (value in [Table 6-23](#)) for the key it used to authenticate itself.

A Tag that supports the *TagPrivilege* command shall permit an Interrogator that supplies the access password (even if zero-valued) or a key to deassert a privilege for the access password or that key, respectively, regardless of the CryptoSuperuser value.

Because only a crypto superuser can assert a deasserted privilege but there is no crypto superuser for the access password, an access-password privilege, once deasserted, cannot be reasserted.

A Tag manufacturer may configure one or more Tag privileges as permanent and unchangeable, in which case these Tag privileges will not be changeable even by a crypto superuser. A Tag that receives a *TagPrivilege* that attempts to change an unchangeable Tag privilege value shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Table 6-22: Tag privileges associated with the access password

| Privilege Name | Bit Assignment | Privilege |
|-----------------|----------------|------------------|
| CryptoSuperuser | 15 | 0 (unchangeable) |
| AuthKill | 14 | 0 (unchangeable) |
| Untraceable | 13 | 0/1 |
| DecFilePriv | 12 | 0/1 |
| RFU | 11 | 0 |
| RFU | 10 | 0 |

| Privilege Name | Bit Assignment | Privilege |
|----------------|----------------|-----------------------------|
| RFU | 9 | 0 |
| RFU | 8 | 0 |
| RFU | 7 | 0 |
| RFU | 6 | 0 |
| RFU | 5 | 0 |
| RFU | 4 | 0 |
| Custom_1 | 3 | Defined by Tag manufacturer |
| Custom_2 | 2 | Defined by Tag manufacturer |
| Custom_3 | 1 | Defined by Tag manufacturer |
| Custom_4 | 0 | Defined by Tag manufacturer |

Table 6-23: Tag privileges associated with a cryptographic suite

| Privilege Name | Bit Assignment | Privilege for Key_0 ¹ | ... | Privilege for Key_N ¹ |
|-----------------|----------------|----------------------------------|-----|----------------------------------|
| CryptoSuperuser | 15 | 0/1 | | 0/1 |
| AuthKill | 14 | 0/1 | | 0/1 |
| Untraceable | 13 | 0/1 | | 0/1 |
| DecFilePriv | 12 | 0/1 | | 0/1 |
| KeyProperty_1 | 11 | Defined by crypto suite | | Defined by crypto suite |
| KeyProperty_2 | 10 | Defined by crypto suite | | Defined by crypto suite |
| KeyProperty_3 | 9 | Defined by crypto suite | | Defined by crypto suite |
| KeyProperty_4 | 8 | Defined by crypto suite | | Defined by crypto suite |
| RFU | 7 | 0 | | 0 |
| RFU | 6 | 0 | | 0 |
| RFU | 5 | 0 | | 0 |
| RFU | 4 | 0 | | 0 |
| Custom_1 | 3 | Defined by Tag manufacturer | | Defined by Tag manufacturer |
| Custom_2 | 2 | Defined by Tag manufacturer | | Defined by Tag manufacturer |
| Custom_3 | 1 | Defined by Tag manufacturer | | Defined by Tag manufacturer |
| Custom_4 | 0 | Defined by Tag manufacturer | | Defined by Tag manufacturer |

Note 1: Each key is assigned to one and only one cryptographic suite.

If a Tag supports the *TagPrivilege* command then this protocol recommends that the Tag manufacturer provide at least one nonzero-valued key with crypto superuser privileges for each cryptographic suite supported by the Tag.

A cryptographic suite defines whether and when:

- a Tag considers an Interrogator to be authenticated.
- an Interrogator considers a Tag to be authenticated.

The cryptographic suite also defines:

- cryptographic conditions that cause a Tag to treat a command's parameters as unsupported.
- cryptographic errors that cause a Tag to transition from the **open** or **secured** state to the **arbitrate** state.

After a successful Interrogator authentication a Tag in the **open** state shall transition to the **secured** state. If the Tag was already in the **secured** state then it remains in the **secured** state. The authenticated Interrogator shall subsequently encapsulate all commands designated "Mandatory

Encapsulation" in [Table 6-28](#) in an *AuthComm* or *SecureComm*. If a Tag receives such a command from an authenticated Interrogator without encapsulation then it shall not execute the command and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag shall transition back to the **open** state, reset its cryptographic engine, and revert to **open**-state file privileges (see below) when an authenticated Interrogator loses its authentication. There are many reasons why an Interrogator may lose its authentication, including but not limited to the Tag receiving a security access command with an incorrect handle, the Tag receiving an invalid command, or the Interrogator starting a new authentication. As a consequence of an Interrogator losing its authentication a Tag with a zero-valued access password may be in the **open** state, in which case the Interrogator may issue an *Access*-command sequence with the zero-valued access password to move the Tag back to the **secured** state (but the Interrogator will still not be authenticated — only a successful *Authenticate* command or *Authenticate*-command sequence authenticates an Interrogator).

An unauthenticated Interrogator may issue an *AuthComm* or a *SecureComm* to an authenticated Tag in the **open** or **secured** state. If the Tag was not previously authenticated by a *Challenge* or *Authenticate* command then it shall not execute the command and instead treat the command's parameters as unsupported (see [Table C-30](#)).

If a condition of a cryptographic suite causes a Tag to transition from the **open** or **secured** state to the **arbitrate** state then the Tag (i) shall not change the value of its inventoried flag, and (ii) shall reset its cryptographic engine.

6.3.2.11.3 File-management access commands

The file-management access commands are *FileOpen*, *FileList*, *FileSetup*, and *FilePrivilege*. All are optional. A Tag that supports *File_N*, $N > 0$ shall implement *FileOpen*; it may implement *FileList*, *FileSetup*, and *FilePrivilege* as well. A Tag may implement one or more of these commands regardless of whether the Tag supports optional core commands and/or cryptographic security.

A Tag may implement zero, one, or more than one file in User memory. If a Tag implements a single file then that file shall be *File_0*. A Tag with User memory shall open *File_0* upon first entering the **open** or **secured** state. If a Tag implements multiple files then it may subsequently close *File_0* and open another file. A Tag shall have only a single file open at any time. All access commands operate on the currently open file.

Each file shall have an 8-bit FileType and a 10-bit FileNum, unless a Tag does not support any file-management access commands, in which case a Tag that implements *File_0* may omit FileType and FileNum.

- A Tag manufacturer shall preassign a FileType to each file supported by the Tag. If a Tag does not support the *FileSetup* command then FileType is not changeable in the field and all files have FileType=00_h. If a Tag supports the *FileSetup* command then FileType is changeable in the field and FileType=00_h indicates that a file's type is currently unassigned.
- A Tag manufacturer shall preassign a unique FileNum to each file supported by the Tag. FileNum is not changeable in the field. A Tag may support up to 1023 files, numbered 0 to 1022 (0000000000₂ – 111111110₂). The files may have different size (including zero size). FileNum=000000000₂ shall be reserved for the base file (*File_0*) of User memory. FileNum=111111111₂ shall be RFU. This protocol recommends, but does not require, that Tag manufacturers number files sequentially.

A *FileOpen* command allows an Interrogator to open a file. Upon receiving a *FileOpen* a Tag shall first close the currently open file and then open the new file, with the new file's starting address mapped to 00_h of User memory. An Interrogator may be able to subsequently read, write, erase, blockpermalock, resize, or modify privileges for the newly opened file depending on the Tag state and if/how the Interrogator authenticated itself.

A *FileList* command allows an Interrogator to determine the existence of, size of, attributes of, and its privileges to, one or more files.

A *FileSetup* command allows an Interrogator to change the FileType for, and/or resize, the currently open file. Only a *dynamic* Tag (see below) is capable of resizing a file.

A *FilePrivilege* command allows an Interrogator to read or alter the privileges (see below) granted by the currently open file to the **open** state, access password, or a key.

A Tag manufacturer shall precreate all files; the number of files shall not be changeable in the field. This protocol defines two types of Tags, *static* and *dynamic*, according to their memory-allocation features as follows:

Static: A manufacturer of a *static* Tag shall preallocate all User memory to files. A *static* Tag may permit changing a file's FileType but shall not permit file resizing.

Dynamic: A manufacturer of a *dynamic* Tag may preallocate no, some, or all User memory to files. A *dynamic* Tag may permit file resizing by an Interrogator that has a file superuser privilege.

A Tag manufacturer shall decide where a Tag stores its FileType and FileNum data and may choose a readable portion of memory (if desired). Regardless of the location, a Tag shall not allow an Interrogator to modify a file's type by any command except *FileSetup*, and shall not allow an Interrogator to modify a FileNum by any means.

Files may range in size from a minimum of zero to a maximum of 1022 blocks. Commands that include a FileSize parameter use 10 bits to specify sizes from zero to 1022 blocks (000000000_2 – 111111110_2 , respectively). FileSize 111111111_2 shall be RFU.

Block size may be one to 1024 words. A Tag manufacturer shall predefine a single fixed, unchangeable block size that the Tag shall use for all file allocation as well as for the *BlockPermalock* command. Tag manufacturers shall not use block sizes exceeding 1024 words. Tag replies that return a BlockSize value use 10 bits to specify the size from one (000000000_2) to 1024 (111111111_2) words. BlockSize does not have an RFU value.

This protocol allows file sizes from 0 to 16,744,448 bits (max FileSize=1022 blocks, max BlockSize=1024 words, word=16 bits).

If a Tag supports File_0 then it shall provide the file privileges shown in [Table 6-24](#). If a Tag supports File_N, N>0 then it shall also provide the file privileges shown in [Table 6-25](#). Each file has a 4-bit privilege for the **open** state, for the access password in the **secured** state, and for each key in the **secured** state, as follows:

Open state: Each file has a 4-bit **open**-state privilege. A Tag with M files shall implement M 4-bit **open**-state file privileges, one for each file.

Access password (secured state): Each file has a single 4-bit privilege for the access password (even if the access password is zero-valued). A Tag with M files shall implement M 4-bit **secured**-state access-password file privileges, one for each file.

Key (secured state): Each file has a 4-bit privilege for each key implemented by the Tag. A Tag with M files and N keys shall implement M×N 4-bit **secured**-state key file privileges.

Given the above, a Tag with N keys and M files supports $(2+N) \times M$ independent 4-bit privileges. For example, suppose a Tag implements N=2 keys and File_0, File_1, and File_2. Then the Tag has $4 \times 3 = 12$ 4-bit privileges.

A *FilePrivilege* may assign privileges 0000_2 – 0011_2 and 1100_2 – 1111_2 in [Table 6-24](#) and [Table 6-25](#). If a Tag does not implement any manufacturer-defined privileges then the Tag may store only the 2 LSBs of the 4-bit privilege, but all communications still use 4-bit values. In this latter case, if an Interrogator sends a 4-bit privilege with either MSB being nonzero then the Tag shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

The access password or a key with a 0011_2 **secured**-state file privilege in [Table 6-24](#) or [Table 6-25](#) is defined to be a superuser for that file.

A Tag shall permit an Interrogator that accessed or authenticated itself as a file superuser to:

- read or assign a new 4-bit privilege for the **open** state, access password, or any key (including its own) regardless of the cryptographic suite to which the key is assigned, for the currently open file, using a *FilePrivilege* command.
- change the FileType of the currently open file using a *FileSetup* command, for a *static* or a *dynamic* Tag.
- resize the currently open file using a *FileSetup* command, but only if the file contains no permalocked or permanunlocked memory and only if the Tag is *dynamic*.

A Tag shall not permit an Interrogator that did not access or authenticate itself as a file superuser to:

- read or assign the 4-bit privilege for the **open** state, for the currently open file.
- read or assign the 4-bit privilege for the access password or for any key other than the one it used to enter the **secured** state, for the currently open file.
- increase the privileges (move down one or more rows in [Table 6-24](#) or [Table 6-25](#)) for the access password or for any key, for the currently open file.

If the access password or key that a Tag used to enter the **secured** state has $\text{DecFilePriv}=1$ (see [Table 6-22](#) and [Table 6-23](#)) then a Tag shall permit an Interrogator to self-reduce its privileges (move up one or more rows in [Table 6-24](#) or [Table 6-25](#)) to the currently open file for this access password or key. To be clear, if $\text{DecFilePriv}=1$ then an Interrogator may, via a *FilePrivilege* command, instruct a Tag to decrement a 0011_2 privilege to 0010_2 , 0001_2 , or 0000_2 ; a 0010_2 privilege to 0001_2 or 0000_2 ; or a 0001_2 privilege to 0000_2 for the currently open file for the access password or key that the Tag used to enter the **secured** state.

Table 6-24: File_0 privileges

| Privilege value | Open State (Privilege by File) | | | | | Privilege value | Secured State (Privilege by Access Password or Key) | | | | |
|-----------------|--------------------------------|-----------------------------------|-------------------------|----------------|------------|-------------------|---|-----------------------------------|-------------------------|----------------|------------|
| | Read | Write BlockWrite BlockErase | Lock Block-Permalock | File-Privilege | File-Setup | | Read | Write BlockWrite BlockErase | Lock Block-Permalock | File-Privilege | File-Setup |
| 0000 | ✓ | × | D | D | D | 0000 | ✓ | × | × | × | × |
| 0001 | ✓ | × | D | D | D | 0001 | ✓ | L | × | P | × |
| 0010 | ✓ | L | D | D | D | 0010 | ✓ | L | ✓ | P | × |
| 0011 | ✓ | L | D | D | D | 0011 ¹ | ✓ | L | ✓ | ✓ | ✓ |
| 0100 | RFU | | | | | 0100 | RFU | | | | |
| 0101 | RFU | | | | | 0101 | RFU | | | | |
| 0110 | RFU | | | | | 0110 | RFU | | | | |
| 0111 | RFU | | | | | 0111 | RFU | | | | |
| 1000 | RFU | | | | | 1000 | RFU | | | | |
| 1001 | RFU | | | | | 1001 | RFU | | | | |
| 1010 | RFU | | | | | 1010 | RFU | | | | |
| 1011 | RFU | | | | | 1011 | RFU | | | | |
| 1100 | Manufacturer defined | | | | | 1100 | Manufacturer defined | | | | |
| 1101 | Manufacturer defined | | | | | 1101 | Manufacturer defined | | | | |
| 1110 | Manufacturer defined | | | | | 1110 | Manufacturer defined | | | | |
| 1111 | Manufacturer defined | | | | | 1111 | Manufacturer defined | | | | |

Key: ✓=allowed by privilege; ×=disallowed by privilege; L=allowance determined by lock and blockpermalock status of the specified memory banks/blocks; P=allowance determined by DecFilePriv for the password or key; D=command not permitted in the state.

Note 1: This 0011 **secured**-state privilege is a superuser for File_0.

Table 6-25: File_N (N>0) privileges

| Privilege value | Open State (Privilege by File) | | | | | Privilege value | Secured State (Privilege by Access Password or Key) | | | | |
|-----------------|--------------------------------|-----------------------------------|-----------------|----------------|------------|-------------------|---|-----------------------------------|-----------------|----------------|------------|
| | Read | Write BlockWrite BlockErase | Block-Permalock | File-Privilege | File-Setup | | Read | Write BlockWrite BlockErase | Block-Permalock | File-Privilege | File-Setup |
| 0000 | x | x | D | D | D | 0000 | x | x | x | x | x |
| 0001 | ✓ | x | D | D | D | 0001 | ✓ | x | x | P | x |
| 0010 | ✓ | L | D | D | D | 0010 | ✓ | L | x | P | x |
| 0011 | ✓ | L | D | D | D | 0011 ¹ | ✓ | L | ✓ | ✓ | ✓ |
| 0100 | RFU | | | | | 0100 | RFU | | | | |
| 0101 | RFU | | | | | 0101 | RFU | | | | |
| 0110 | RFU | | | | | 0110 | RFU | | | | |
| 0111 | RFU | | | | | 0111 | RFU | | | | |
| 1000 | RFU | | | | | 1000 | RFU | | | | |
| 1001 | RFU | | | | | 1001 | RFU | | | | |
| 1010 | RFU | | | | | 1010 | RFU | | | | |
| 1011 | RFU | | | | | 1011 | RFU | | | | |
| 1100 | Manufacturer defined | | | | | 1100 | Manufacturer defined | | | | |
| 1101 | Manufacturer defined | | | | | 1101 | Manufacturer defined | | | | |
| 1110 | Manufacturer defined | | | | | 1110 | Manufacturer defined | | | | |
| 1111 | Manufacturer defined | | | | | 1111 | Manufacturer defined | | | | |

Key: ✓=allowed by privilege; x=disallowed by privilege; L=allowance determined by lock and blockpermalock status of the specified memory banks/blocks; P=allowance determined by DecFilePriv for the password or key; D=command not permitted in the state.

Note 1: This 0011 secured-state privilege is a superuser for File_0.

Table 6-26: Allowed file resizing

| File number | File is permaunlocked or permalocked | One or more file blocks are permaunlocked or permalocked | Increasing file size allowed? | Decreasing file size allowed? |
|-------------|--------------------------------------|--|-------------------------------|-------------------------------|
| N = 0 | No | No | Yes | Yes |
| | No | Yes | Yes | No |
| | Yes | No | No | No |
| | Yes | Yes | No | No |
| N > 0 | N/A | No | Yes | Yes |
| | | Yes | Yes | No |

Table 6-27: Access commands and Tag states in which they are permitted

| Command | State | | | Subclass | Remark |
|---------------|--------------|---------|---------|----------|---|
| | Acknowledged | Open | Secured | | |
| <i>Req_RN</i> | allowed | allowed | allowed | Core | mandatory command |
| <i>Read</i> | disallowed | allowed | allowed | Core | mandatory command |
| <i>Write</i> | disallowed | allowed | allowed | Core | mandatory command; requires prior <i>Req_RN</i> |

| Command | State | | | Subclass | Remark |
|---------------------------------|--------------|------------|---------|----------|--|
| | Acknowledged | Open | Secured | | |
| <i>Kill</i> (password-based) | disallowed | allowed | allowed | Core | mandatory command; requires prior <i>Req_RN</i> |
| <i>Kill</i> (authenticated) | disallowed | disallowed | allowed | Core | optional usage of mandatory <i>Kill</i> command |
| <i>Lock</i> | disallowed | disallowed | allowed | Core | mandatory command |
| <i>Access</i> | disallowed | allowed | allowed | Core | optional command; requires prior <i>Req_RN</i> |
| <i>BlockWrite</i> | disallowed | allowed | allowed | Core | optional command |
| <i>BlockErase</i> | disallowed | allowed | allowed | Core | optional command |
| <i>BlockPermalock</i> | disallowed | disallowed | allowed | Core | optional command |
| <i>ReadBuffer</i> | disallowed | allowed | allowed | Core | optional command |
| <i>Untraceable</i> | disallowed | disallowed | allowed | Core | optional command |
| <i>Authenticate</i> | disallowed | allowed | allowed | Security | optional command |
| <i>AuthComm</i> | disallowed | allowed | allowed | Security | optional command; requires prior authentication |
| <i>SecureComm</i> | disallowed | allowed | allowed | Security | optional command; requires prior authentication |
| <i>KeyUpdate</i> | disallowed | disallowed | allowed | Security | optional command; requires prior authentication |
| <i>TagPrivilege</i> | disallowed | disallowed | allowed | Security | optional command |
| <i>FileOpen</i> | disallowed | allowed | allowed | File | optional command |
| <i>FileList</i> | disallowed | allowed | allowed | File | optional command |
| <i>FilePrivilege</i> | disallowed | disallowed | allowed | File | optional command |
| <i>FileSetup</i> | disallowed | disallowed | allowed | File | optional command |

A Tag manufacturer may assign file privileges to the **open** state and to the access password or keys as required by the Tag's intended use case. A Tag manufacturer may assign file superuser privileges to the access password or to any key. Although this protocol recommends against a Tag manufacturer assigning file superuser privileges to a zero-valued access password or key, it does not prohibit a Tag manufacturer from doing so.

As described above, a Tag opens *File_0* upon first entering the **open** or **secured** state. If an Interrogator attempts to subsequently open another file for which its privilege level is 0000₂ then the Tag opens the file but does not grant the Interrogator any file privileges.

Some privilege fields in [Table 6-24](#) and [Table 6-25](#) show "x" (disallowed by privilege). If *Read* is "x" for a privilege value then a Tag shall behave as if the memory location does not exist. Otherwise, if *Write*, *BlockWrite*, or *BlockErase* are "x" then the Tag shall behave as if the memory location is permalocked; and if *Lock* or *BlockPermalock* are "x" then the Tag shall behave as if the memory location is neither lockable nor unlockable. If *FilePrivilege* or *FileSetup* are "x" then the Tag shall behave as if the Interrogator has insufficient privileges. If a Tag implements the *BlockPermalock* command then all files shall support the *BlockPermalock* command.

If a Tag's User memory is untraceably hidden then the Tag shall only execute a *FileOpen*, *FileList*, *FileSetup*, or *FilePrivilege* issued by an Interrogator with an asserted Untraceable privilege (see [Table 6-22](#) and [Table 6-23](#)); if the Interrogator has a deasserted Untraceable privilege then the Tag shall treat these commands' parameters as unsupported (see [Table C-30](#)).

A Tag shall not permit a permalocked portion of memory to be erased or overwritten, except for the **L** and **U** bits in EPC memory, which an Interrogator with an asserted Untraceable privilege may overwrite.

In some instances a *dynamic* Tag may allow file resizing. Whether a Tag allows resizing shall depend on whether the Tag accepts a *FileSetup* command (varies by privilege and state), whether the Tag has free memory available for the resizing, and whether the file or any blocks in it are permalocked or permaunlocked. See [Table 6-26](#).

6.3.2.12 Interrogator commands and Tag replies

Interrogator-to-Tag commands shall use the command codes, protection, and parameters shown in [Table 6-28](#).

- *QueryRep* and *ACK* have 2-bit command codes beginning with 0₂.
- *Query*, *QueryAdjust*, and *Select* have 4-bit command codes beginning with 10₂.
- Other commands that are sensitive to link throughput use 8-bit command codes beginning with 110₂.
- Other commands that are insensitive to link throughput use 16-bit command codes beginning with 1110₂.
- *QueryRep*, *ACK*, *Query*, *QueryAdjust*, and *NAK* have the unique command lengths shown in [Table 6-28](#). No other commands shall have these lengths. If a Tag receives one of these commands with an incorrect length then it shall treat the command as invalid (see [Table C-30](#)).
- *Query* is protected by a CRC-5, shown in [Table 6-12](#) and detailed in [Annex F](#).
- *Select*, *Req_RN*, *Read*, *Write*, *Kill*, *Lock*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, *Authenticate*, *SecureComm*, *AuthComm*, *KeyUpdate*, *ReadBuffer*, *Challenge*, *Untraceable*, *FileOpen*, *FileList*, *FilePrivilege*, *FileSetup*, and *TagPrivilege* are protected by a CRC-16, defined in [6.3.1.5](#) and detailed in [Annex F](#).
- R=>T commands begin with either a preamble or a frame-sync, as described in [6.3.1.2.8](#). The command-code lengths specified in [Table 6-28](#) do not include the preamble or frame-sync.
- A Tag's behavior upon receiving a faulty command depends on the fault type and the Tag state. [Annex B](#) and [Annex C](#) define the fault types by state. In general, the faults are (1) unsupported parameters, (2) incorrect handle, (3) improper, and (4) invalid. Note that, for some cryptographic suites, a Tag may reset its cryptographic engine and change state upon receiving a faulty command.

Table 6-28: Interrogator Commands

| Command | Code | Length (bits) | Mandatory Command (Y/N)? | Reply Type | Encapsulation | | | Protection |
|--------------------------------|----------|---------------|--------------------------|------------|--------------------------------|------------------------------|-------------------------------|---------------------------|
| | | | | | SecureComm ² (Y/N)? | AuthComm ² (Y/N)? | Mandatory ³ (Y/N)? | |
| <i>QueryRep</i> | 00 | 4 | Yes | Immediate | No | No | No | Unique length |
| <i>ACK</i> | 01 | 18 | Yes | Immediate | Yes | Yes | No | Unique length |
| <i>Query</i> | 1000 | 22 | Yes | Immediate | No | No | No | Unique length and a CRC-5 |
| <i>QueryAdjust</i> | 1001 | 9 | Yes | Immediate | No | No | No | Unique length |
| <i>Select</i> | 1010 | > 44 | Yes | None | No | No | No | CRC-16 |
| <i>Reserved for future use</i> | 1011 | – | – | – | – | – | – | – |
| <i>NAK</i> | 11000000 | 8 | Yes | Immediate | No | No | No | Unique length |
| <i>Req_RN</i> | 11000001 | 40 | Yes | Immediate | Yes | Yes | No | CRC-16 |



| Command | Code | Length (bits) | Mandatory Command (Y/N)? | Reply Type | Encapsulation | | | Protection |
|--|-----------------------------|---------------|--------------------------|---------------------|--------------------------------|------------------------------|-------------------------------|----------------------|
| | | | | | SecureComm ² (Y/N)? | AuthComm ² (Y/N)? | Mandatory ³ (Y/N)? | |
| <i>Read</i> | 11000010 | > 57 | Yes | Immediate | Yes | Yes | Yes | CRC-16 |
| <i>Write</i> | 11000011 | > 58 | Yes | Delayed | No | No | No | CRC-16 |
| <i>Kill</i> | 11000100 | 59 | Yes | Delayed | Yes ¹ | Yes ¹ | Yes ¹ | CRC-16 |
| <i>Lock</i> | 11000101 | 60 | Yes | Delayed | Yes | Yes | Yes | CRC-16 |
| <i>Access</i> | 11000110 | 56 | No | Immediate | No | No | No | CRC-16 |
| <i>BlockWrite</i> | 11000111 | > 57 | No | Delayed | Yes | Yes | Yes | CRC-16 |
| <i>BlockErase</i> | 11001000 | > 57 | No | Delayed | Yes | Yes | Yes | CRC-16 |
| <i>BlockPermalock</i> | 11001001 | > 66 | No | Immediate & Delayed | Yes | Yes | Yes | CRC-16 |
| <i>Used by ISO 18000-63</i> ... | 11001010 ... | – | – | – | – | – | – | – |
| 11010001 | | | | | | | | |
| <i>ReadBuffer</i> | 11010010 | 67 | No | Immediate | No | Yes | No | CRC-16 |
| <i>FileOpen</i> | 11010011 | 52 | No ⁴ | Immediate | Yes | Yes | Yes | CRC-16 |
| <i>Challenge</i> | 11010100 | > 48 | No | None | No | No | No | CRC-16 |
| <i>Authenticate</i> | 11010101 | > 64 | No | In-process | No | No | No | CRC-16 |
| <i>SecureComm</i> | 11010110 | > 56 | No | In-process | No | No | No | CRC-16 |
| <i>AuthComm</i> | 11010111 | > 42 | No | In-process | No | No | No | CRC-16 |
| <i>Reserved for future use</i> | 11011000 | – | – | – | – | – | – | – |
| <i>Used by ISO 18000-63</i> | 11011001 | – | – | – | – | – | – | – |
| <i>Reserved for future use</i> ... | 11011010 ... | – | – | – | – | – | – | – |
| 11011111 | | | | | | | | |
| <i>Reserved for custom commands</i> | 11100000 00000000 ... | – | – | – | – | – | – | Manufacturer defined |
| 11100000 | | | | | | | | |
| 11111111 | | | | | | | | |
| <i>Reserved for proprietary commands</i> | 11100001 00000000 ... | – | – | – | – | – | – | Manufacturer defined |
| 11100001 | | | | | | | | |
| 11111111 | | | | | | | | |
| <i>Untraceable</i> | 11100010 00000000 | 62 | No | Delayed | Yes | Yes | Yes | CRC-16 |
| <i>FileList</i> | 11100010 00000001 | 71 | No | In-process | Yes | Yes | Yes | CRC-16 |

| Command | Code | Length (bits) | Mandatory Command (Y/N)? | Reply Type | Encapsulation | | | Protection |
|--------------------------------|---|---------------|--------------------------|------------|--------------------------------|------------------------------|-------------------------------|------------|
| | | | | | SecureComm ² (Y/N)? | AuthComm ² (Y/N)? | Mandatory ³ (Y/N)? | |
| <i>KeyUpdate</i> | 11100010 00000010 | > 72 | No | In-process | Yes | Yes | No | CRC-16 |
| <i>TagPrivilege</i> | 11100010 00000011 | 78 | No | In-process | Yes | Yes | Yes | CRC-16 |
| <i>FilePrivilege</i> | 11100010 00000100 | 68 | No | In-process | Yes | Yes | Yes | CRC-16 |
| <i>FileSetup</i> | 11100010 00000101 | 71 | No | In-process | Yes | Yes | Yes | CRC-16 |
| <i>Reserved for future use</i> | 11100010 00000110 ... 11101111 11111111 | – | – | – | – | – | – | – |

Note 1: An authenticated *Kill* shall be encapsulated in a *SecureComm* or an *AuthComm*; a password-based *Kill* shall not be encapsulated.

Note 2: Commands with a “yes” may be encapsulated in a *SecureComm* or an *AuthComm*, as appropriate.

Note 3: For an authenticated Interrogator and commands with a “yes”, encapsulation in a *SecureComm* or an *AuthComm* is mandatory.

Note 4: If a Tag supports File_N, N>0 then *FileOpen* is mandatory.

6.3.2.12.1 Select commands

The select command set comprises *Select* and *Challenge*.

6.3.2.12.1.1 Select (mandatory)

Interrogators and Tags shall implement the *Select* command shown in [Table 6-29](#). A *Select* allows an Interrogator to select a Tag subpopulation based on user-defined criteria, enabling union (U), intersection (∩), and negation (–) based Tag partitioning. Interrogators perform U and ∩ operations by issuing successive *Select* commands. *Select* can assert or deassert a Tag’s **SL** flag, which applies across all four sessions, or it can set a Tag’s **inventoried** flag to either *A* or *B* in any one of the four sessions. A Tag executes a *Select* from any state except **killed**. *Select* passes the following parameters from Interrogator to Tags:

- **Target** indicates whether the *Select* modifies a Tag’s **SL** flag or its **inventoried** flag, and in the case of **inventoried** it further specifies one of four sessions. A *Select* that modifies the **SL** flag shall not modify an **inventoried** flag, and vice versa. A Tag shall ignore a *Select* whose **Target** is 10₁₂, 110₂, or 111₂.
- **Action** elicits the Tag behavior in [Table 6-30](#), in which matching and not-matching Tags assert or deassert **SL** or set their **inventoried** flag to *A* or *B*. A Tag conforming to the contents of the **MemBank**, **Pointer**, **Length**, and **Mask** fields is matching. A Tag not conforming to the contents of these fields is not-matching. The criteria for determining whether a Tag is matching or not-matching are specified by the **MemBank**, **Pointer**, **Length** and **Mask** fields.
- **MemBank** specifies how a Tag applies **Mask**. If **MemBank**=00₂ then the Tag searches for at least one file whose **FileType** matches **Mask**. If **MemBank**=01₂, 10₂, 11₂ then a Tag applies **Mask** to the EPC memory bank, TID memory bank, or File₀, respectively. A *Select* specifies a single **FileType** or memory bank. Successive *Selects* may apply to different file types and/or memory banks.
- **Pointer** specifies a starting bit address for the **Mask** comparison. **Pointer** uses EBV formatting (see [Annex A](#)) and bit (not word) addressing. If **MemBank**=00₂ then an Interrogator shall set **Pointer** to 00_h; if a Tag receives a *Select* with **MemBank**=00₂ and a nonzero **Pointer** value then it shall ignore the *Select*.
- **Length** specifies the length of **Mask**. **Length** is 8 bits, allowing **Masks** from 0 to 255 bits in length. If **MemBank**=00₂ then an Interrogator shall set **Length**=00001000₂; if a Tag receives a *Select* with **MemBank**=00₂ and **Length**<>00001000₂ then it shall ignore the *Select*.

- Mask is either a FileType (if MemBank=00₂) or a bit string that a Tag compares to a memory location that begins at Pointer and ends Length bits later (if MemBank<>00₂). An untraceable Tag shall process a *Select* with MemBank=00₂ whose User memory is traceable, or with MemBank<>00₂ whose Mask operates on a completely traceable bit string. A Tag shall treat as not-matching a *Select* command whose Mask includes untraceably hidden memory.
 - MemBank=00₂: If a Tag has a file with the specified FileType then the Tag is matching. If the Tag does not support files or does not have a file with the specified FileType then the Tag is not-matching.
 - MemBank<>00₂: If Mask matches the string specified by Pointer and Length then the Tag is matching. If Pointer and Length reference a memory location that does not exist then the Tag is not-matching. If Length is zero then the Tag is matching, unless Pointer references a memory location that does not exist, or Truncate=1 and Pointer is outside the EPC specified in the length field in the StoredPC, in which case the Tag is not-matching.
- Truncate indicates whether a Tag's backscattered reply shall be truncated to those EPC bits that follow Mask. If an Interrogator asserts Truncate, and if a subsequent *Query* specifies Sel=10 or Sel=11, then a matching Tag shall truncate its *ACK* reply to the portion of the EPC immediately following Mask, followed by a PacketCRC. If an Interrogator asserts Truncate then it shall assert it:
 - in the last *Select* that the Interrogator issues prior to sending a *Query*,
 - only if the *Select* has Target=100₂, and
 - only if Mask ends in the EPC.

These constraints *do not* preclude an Interrogator from issuing multiple *Select* commands that target the **SL** and/or **inventoried** flags. They *do* require that an Interrogator that is requesting Tags to truncate their replies assert Truncate in the last *Select*, and that this last *Select* targets the **SL** flag. A Tag shall decide whether to truncate its backscattered EPC on the basis of the most recently received valid *Select* (i.e. not ignored and matching or not-matching).

If a Tag receives a *Select* with Truncate=1 and

- Target<>100₂ or MemBank<>01₂ then the Tag shall ignore the *Select*.
- MemBank=01₂ but Mask ends outside the EPC specified by the **L** bits in the StoredPC then the Tag shall be not-matching.

A Tag shall preface a truncated reply with five leading zeros (00000₂) inserted between the preamble and the truncated reply. Specifically, when truncating its replies a Tag backscatters 00000₂, then the portion of its EPC following Mask, and then a PacketCRC. See [Table 6-17](#).

A Tag shall power-up with Truncate=0.

Mask may end at the last bit of the EPC, in which case a truncating Tag shall backscatter 00000₂ followed by a PacketCRC.

Truncated replies never include an XPC_W1 or an XPC_W2, because Mask must end in the EPC.

Because a Tag stores its StoredPC and StoredCRC in EPC memory, a *Select* command may select on them. Because a Tag computes its PacketPC and PacketCRC dynamically and does not store them in memory, a *Select* command is unable to select on them.

A *Select* whose Pointer, Length, and Mask include the StoredPC may produce unexpected behavior. Specifically, if a Tag's *ACK* reply uses a PacketPC then the reply may appear to not match Mask even though the Tag's behavior indicates matching, and vice versa. For example, suppose that an Interrogator sends a *Select* to match a 00100₂ length field in the StoredPC. Further assume that the Tag is matching, but has an asserted **XI**. The Tag will increment its length field to 00101₂ when replying to the *ACK*. The Tag was matching, but the backscattered length field in the PacketPC appears to be not-matching.

An Interrogator shall prepend a *Select* command with a frame-sync (see [6.3.1.2.8](#)). The CRC-16 that protects a *Select* is calculated over the first command-code bit to the Truncate bit.

A Tag shall not reply to a *Select*.

Table 6-29: Select command

| | Command | Target | Action | MemBank | Pointer | Length | Mask | Truncate | CRC |
|-------------|---------|---|--------------------------------|--|-------------------------|--------------------|------------|---|--------|
| # of bits | 4 | 3 | 3 | 2 | EBV | 8 | Variable | 1 | 16 |
| description | 1010 | 000: Inventoried (S0) 001: Inventoried (S1) 010: Inventoried (S2) 011: Inventoried (S3) 100: SL 101: RFU 110: RFU 111: RFU | See Table 6-30 | 00: FileType 01: EPC 10: TID 11: File_0 | Starting Mask addresses | Mask length (bits) | Mask value | 0: Disable truncation 1: Enable truncation | CRC-16 |

Table 6-30: Tag response to Action parameter

| Action | Tag Matching | Tag Not-Matching |
|--------|--------------------------------|--------------------------------|
| 000 | assert SL or inventoried → A | deassert SL or inventoried → B |
| 001 | assert SL or inventoried → A | do nothing |
| 010 | do nothing | deassert SL or inventoried → B |
| 011 | negate SL or (A → B, B → A) | do nothing |
| 100 | deassert SL or inventoried → B | assert SL or inventoried → A |
| 101 | deassert SL or inventoried → B | do nothing |
| 110 | do nothing | assert SL or inventoried → A |
| 111 | do nothing | negate SL or (A → B, B → A) |

6.3.2.12.1.2 Challenge (optional)

Interrogators and Tags may implement the *Challenge* command; if they do then they shall implement it as shown in [Table 6-31](#). A *Challenge* allows an Interrogator to instruct multiple Tags to simultaneously yet independently precompute and store a cryptographic value or values for use in a subsequent authentication. The generic nature of the *Challenge* command allows it to support a wide variety of cryptographic suites. A Tag executes a *Challenge* from any state except **killed**. *Challenge* has the following fields:

- IncRepLen specifies whether the Tag omits or includes length in its stored reply. If IncRepLen=0 then the Tag omits length from its stored reply; if IncRepLen=1 then the Tag includes length in its stored reply.
- Immed specifies whether a Tag concatenates response to its EPC when replying to an *ACK*. If immed=0 then the Tag does not concatenate response to its EPC when replying to an *ACK*; if immed=1 then the Tag backscatters EPC + response when replying to an *ACK*.
- CSI selects the cryptographic suite that Tag and Interrogator use for the *Challenge*.
- Length is the message length in bits.
- Message includes parameters for the authentication.

Upon receiving a *Challenge* a Tag that supports the command shall return to the **ready** state and deassert its **C** flag. If the Tag supports the CSI and can execute message then it shall perform the requested action(s); otherwise the Tag shall not execute message. A Tag shall not reply to a *Challenge*.

A *Challenge* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. If a Tag receives a *Challenge* containing nonzero RFU bits then it shall return to the **ready** state and deassert its **C** flag but not execute message. Future protocols may use these RFU bits to expand the functionality of the *Challenge* command.

An Interrogator shall prepend a *Challenge* command with a frame-sync (see [6.3.1.2.8](#)). The CRC-16 that protects a *Challenge* is calculated over the first command-code bit to the last Message bit.

If a Tag supports the *Challenge* command then it shall implement the security (**S**) indicator (see [6.3.2.1.3](#)).

The cryptographic suite specifies message formatting, what value or values the Tag precomputes, and the format in which the Tag stores the value(s). It specifies Tag behavior if a Tag cannot compute one or more values. It may contain additional information such as how Tag and Interrogator perform a subsequent authentication from values precomputed by a *Challenge*. It specifies the formatting of the computed result for both a successful and an unsuccessful *Challenge*. It may contain information about how Tag and Interrogator derive session keys for subsequent communications. It may include parameters, such as a key, that affect pre- and post-authenticated communications. See [Annex M](#) for the parameters specified by a cryptographic suite.

After executing a *Challenge* a Tag shall store its response (result or error code) in its ResponseBuffer. If IncRepLen=1 then the Tag also stores the length of the response (in bits) as shown in [Figure 6-17](#). An Interrogator may subsequently read the response using a *ReadBuffer* command.

After executing and storing a response a Tag shall assert its **C** flag. A Tag shall not assert its **C** flag until after it has computed and stored the entire response. A Tag shall deassert its **C** flag upon (a) receiving a subsequent *Challenge*, or (b) exceeding the **C** flag persistence time in [Table 6-20](#). As described above the ResponseBuffer contents may include a length field and may be a cryptographic response or an error code. A Tag does not permit an Interrogator to read its ResponseBuffer when **C**=0.

If the most recent *Challenge* received and executable by a Tag asserts immed, and if the Tag's **C** flag is asserted when it receives a subsequent *ACK*, then when replying to the *ACK* the Tag shall concatenate its ResponseBuffer contents to its EPC and backscatter the concatenated reply. See [Table 6-17](#). See also [Figure 6-23](#).

A *Challenge* may precede a *Query*. Tags that hear a *Challenge* and support the command, the CSI, and message compute their results simultaneously. An Interrogator may select on the **C** flag to preferentially inventory Tags that have successfully stored a response.

If an Interrogator sends a command while a Tag is processing a *Challenge* then the Tag may abort its processing (leaving **C**=0) and evaluate the command or, in environments with limited power availability, may undergo a power-on reset. This protocol recommends that Interrogators send CW for a sufficient period of time after sending a *Challenge* for all Tags to compute and store their result and assert their **C** flag.

If a Tag observes a properly formatted *Challenge* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall return to **ready** and enforce a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for a *Challenge* receives a *Challenge* during a timeout then it shall return to **ready** but not act on or otherwise execute any portion of the *Challenge*.

Table 6-31: Challenge Command

| | Command | RFU | IncRepLen | Immed | CSI | Length | Message | CRC |
|-------------|----------|-----|---|---|------------|---------------------------------|---|--------|
| # of bits | 8 | 2 | 1 | 1 | 8 | 12 | Variable | 16 |
| description | 11010100 | 00 | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | 0: Do not transmit result with EPC 1: Transmit result with EPC | <u>CSI</u> | <u>length</u> of <u>message</u> | <u>message</u> (depends on <u>CSI</u>) | CRC-16 |

6.3.2.12.2 Inventory commands

The inventory command set comprises *Query*, *QueryAdjust*, *QueryRep*, *ACK*, and *NAK*.

6.3.2.12.2.1 Query (mandatory)

Interrogators and Tags shall implement the *Query* command shown in [Table 6-32](#). *Query* initiates and specifies an inventory round. *Query* includes the following fields:

- **DR** (TRcal divide ratio) sets the T=>R link frequency as described in [6.3.1.2.8](#) and [Table 6-9](#).
- **M** (cycles per symbol) sets the T=>R data rate and modulation format as shown in [Table 6-10](#).
- **TRext** chooses whether a Tag prepends the T=>R preamble with a pilot tone as described in [6.3.1.3.2.2](#) and [6.3.1.3.2.4](#). A Tag's reply to a command that uses a *delayed* or an *in-process* reply (see [6.3.1.6](#)) always uses an extended preamble regardless of the **TRext** value.
- **Sel** chooses which Tags respond to the *Query* (see [6.3.2.12.1.1](#) and [6.3.2.10](#)).
- **Session** chooses a session for the inventory round (see [6.3.2.10](#)).
- **Target** selects whether Tags whose **inventoried** flag is *A* or *B* participate in the inventory round. Tags may change their inventoried flag from *A* to *B* (or vice versa) as a result of being singulated.
- **Q** sets the number of slots in the round (see [6.3.2.10](#)).

An Interrogator shall prepend a *Query* with a preamble (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate a *Query* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

The CRC-5 that protects a *Query* is calculated over the first command-code bit to the last **Q** bit. If a Tag receives a *Query* with a CRC-5 error then it shall treat the command as invalid (see [Table C-30](#)).

Upon receiving a *Query*, Tags with matching **Sel** and **Target** shall pick a random value in the range $(0, 2^Q - 1)$, inclusive, and shall load this value into their slot counter. If a Tag, in response to the *Query*, loads its slot counter with zero, then its reply to a *Query* shall be as shown in [Table 6-33](#) using the *immediate* reply type specified in [6.3.1.6.1](#); otherwise the Tag shall remain silent.

A *Query* may initiate an inventory round in a new session or in the prior session. If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose **session** parameter matches the prior session it shall invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) for the session before it evaluates whether to transition to **ready**, **arbitrate**, or **reply**. If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose **session** parameter does not match the prior session it shall leave its **inventoried** flag for the prior session unchanged when beginning the new round.

A Tag shall support all **DR** and **M** values specified in [Table 6-9](#) and [Table 6-10](#), respectively.

A Tag in any state other than **killed** shall execute a *Query* command, starting a new round in the specified session and transitioning to **ready**, **arbitrate**, or **reply**, as appropriate (see [Figure 6-21](#)).

A Tag in the **killed** state shall ignore a *Query*.

Table 6-32: *Query* command

| | Command | DR | M | TRext | Sel | Session | Target | Q | CRC |
|-------------|---------|-----------------------|--|---------------------------------------|---|--------------------------------------|--------------|------|-------|
| # of bits | 4 | 1 | 2 | 1 | 2 | 2 | 1 | 4 | 5 |
| description | 1000 | 0: DR=8 1: DR=64/3 | 00: M=1 01: M=2 10: M=4 11: M=8 | 0: No pilot tone 1: Use pilot tone | 00: All 01: All 10: ~SL 11: SL | 00: S0 01: S1 10: S2 11: S3 | 0: A 1: B | 0–15 | CRC-5 |

Table 6-33: Tag reply to a *Query* command

| | Reply |
|-------------|-------|
| # of bits | 16 |
| description | RN16 |

6.3.2.12.2.2 QueryAdjust (mandatory)

Interrogators and Tags shall implement the *QueryAdjust* command shown in [Table 6-34](#). *QueryAdjust* adjusts Q (i.e. the number of slots in an inventory round – see [6.3.2.10](#)) without changing any other round parameters.

QueryAdjust includes the following fields:

- Session corroborates the session number for the inventory round (see [6.3.2.10](#) and [6.3.2.12.2.1](#)). If a Tag receives a *QueryAdjust* whose session number is different from the session number in the *Query* that initiated the round it shall ignore the command.
- UpDn determines whether and how the Tag adjusts Q , as follows:
 - 110: Increment Q (i.e. $Q = Q + 1$).
 - 000: No change to Q .
 - 011: Decrement Q (i.e. $Q = Q - 1$).

If a Tag receives a *QueryAdjust* with an UpDn value different from those specified above then it shall treat the command as invalid (see [Table C-30](#)). If a Tag whose Q value is 15 receives a *QueryAdjust* with UpDn=110 then it shall change UpDn to 000 prior to executing the command; likewise, if a Tag whose Q value is 0 receives a *QueryAdjust* with UpDn=011 then it shall change UpDn to 000 prior to executing the command.

A Tag shall maintain a running count of the current Q value. The initial Q value is specified in the *Query* command that started the inventory round; one or more subsequent *QueryAdjust* commands may modify Q .

An Interrogator shall prepend a *QueryAdjust* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate a *QueryAdjust* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Upon receiving a *QueryAdjust* Tags first update Q , then pick a random value in the range $(0, 2^Q - 1)$, inclusive, and load this value into their slot counter. If a Tag, in response to the *QueryAdjust*, loads its slot counter with zero, then its reply to a *QueryAdjust* shall be shown in [Table 6-35](#) using the *immediate* reply type specified in [6.3.1.6.1](#); otherwise, the Tag shall remain silent. A Tag shall respond to a *QueryAdjust* only if it received a prior *Query*.

A Tag in any state except **ready** or **killed** shall execute a *QueryAdjust* command if, and only if, (i) the session parameter in the command matches the session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see [6.3.2.12.3.4](#) or [6.3.2.12.3.6](#), respectively).

A Tag in the **acknowledged**, **open**, or **secured** state that receives a *QueryAdjust* whose session parameter matches the session parameter in the prior *Query*, and that is not in the middle of a *Kill* or *Access* command sequence (see [6.3.2.12.3.4](#) or [6.3.2.12.3.6](#), respectively), shall invert its **inventoried** flag (i.e. $A \rightarrow B$ or $B \rightarrow A$, as appropriate) for the current session and transition to **ready**.

Table 6-34: *QueryAdjust* command

| | Command | Session | UpDn |
|-------------|---------|--------------------------------------|---|
| # of bits | 4 | 2 | 3 |
| description | 1001 | 00: S0 01: S1 10: S2 11: S3 | 110: $Q = Q + 1$ 000: No change to Q 011: $Q = Q - 1$ |

Table 6-35: Tag reply to a *QueryAdjust* command

| | Reply |
|-------------|-------|
| # of bits | 16 |
| description | RN16 |

6.3.2.12.2.3 QueryRep (mandatory)

Interrogators and Tags shall implement the *QueryRep* command shown in [Table 6-36](#). *QueryRep* instructs Tags to decrement their slot counters and, if slot=0 after decrementing, to backscatter an RN16 to the Interrogator.

QueryRep includes the following field:

- Session corroborates the session number for the inventory round (see [6.3.2.10](#) and [6.3.2.12.2.1](#)). If a Tag receives a *QueryRep* whose session number is different from the session number in the *Query* that initiated the round it shall ignore the command.

An Interrogator shall prepend a *QueryRep* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate a *QueryRep* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

If a Tag, in response to the *QueryRep*, decrements its slot counter and the decremented slot value is zero, then its reply to a *QueryRep* shall be as shown in [Table 6-37](#) using the *immediate* reply type specified in [6.3.1.6.1](#); otherwise the Tag shall remain silent. A Tag shall respond to a *QueryRep* only if it received a prior *Query*.

A Tag in any state except **ready** or **killed** shall execute a *QueryRep* command if, and only if, (i) the session parameter in the command matches the session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see [6.3.2.12.3.4](#) or [6.3.2.12.3.6](#), respectively).

A Tag in the **acknowledged**, **open**, or **secured** state that receives a *QueryRep* whose session parameter matches the session parameter in the prior *Query*, and that is not in the middle of a *Kill* or *Access* command sequence (see [6.3.2.12.3.4](#) or [6.3.2.12.3.6](#), respectively), shall invert its **inventoried** flag (i.e. $A \rightarrow B$ or $B \rightarrow A$, as appropriate) for the current session and transition to **ready**.

Table 6-36: *QueryRep* command

| | Command | Session |
|-------------|---------|--------------------------------------|
| # of bits | 2 | 2 |
| description | 00 | 00: S0 01: S1 10: S2 11: S3 |

Table 6-37: Tag reply to a *QueryRep* command

| | Reply |
|-------------|-------|
| # of bits | 16 |
| description | RN16 |

6.3.2.12.2.4 ACK (mandatory)

Interrogators and Tags shall implement the *ACK* command shown in [Table 6-38](#). An Interrogator sends an *ACK* to acknowledge a single Tag. *ACK* echoes the Tag's backscattered RN16.

If an Interrogator issues an *ACK* to a Tag in the **reply** or **acknowledged** state then the echoed RN16 shall be the RN16 that the Tag previously backscattered as it transitioned from the **arbitrate** state to the **reply** state. If an Interrogator issues an *ACK* to a Tag in the **open** or **secured** state then the echoed RN16 shall be the Tag's handle (see [6.3.2.12.3.1](#)).

An Interrogator shall prepend an *ACK* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator may encapsulate an *ACK* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

The Tag reply to a successful *ACK* shall be as shown in [Table 6-39](#), using the *immediate* reply type specified in [6.3.1.6.1](#). As described in [6.3.2.1.2](#) and shown in [Table 6-17](#), the reply may be truncated or include a concatenated response. A Tag that receives an *ACK* with an incorrect RN16 or an incorrect handle (as appropriate) shall return to **arbitrate** without responding, unless the Tag is in **ready** or **killed**, in which case it shall ignore the *ACK* and remain in its current state.

If a Tag does not support XPC functionality then the maximum length of its backscattered EPC is 496 bits. If a Tag supports XPC functionality then the maximum length of its backscattered EPC is reduced by two words to accommodate the optional XPC_W1 and XPC_W2, so is 464 bits (see [6.3.2.1.2.2](#)). In either case a Tag's reply to an *ACK* shall not exceed 528 bits for the PC + EPC + PacketCRC, optionally followed by a response field and its associated CRC-16 (see [Table 6-17](#)).

Table 6-38: *ACK* command

| | Reply |
|-------------|--------------------------------|
| # of bits | 21 to 33,328 |
| description | See Table 6-17 |

Table 6-39: Tag reply to a successful *ACK* command

| | Command | RN |
|-------------|---------|------------------------------|
| # of bits | 2 | 16 |
| description | 01 | Echoed RN16 or <u>handle</u> |

6.3.2.12.2.5 *NAK* (mandatory)

Interrogators and Tags shall implement the *NAK* command shown in [Table 6-40](#). A Tag that receives a *NAK* shall return to the **arbitrate** state without changing its **inventoried** flag, unless the Tag is in **ready** or **killed**, in which case it shall ignore the *NAK* and remain in its current state.

An Interrogator shall prepend a *NAK* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate a *NAK* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

A Tag shall not reply to a *NAK*.

Table 6-40: *NAK* command

| | Command |
|-------------|----------|
| # of bits | 8 |
| description | 11000000 |

6.3.2.12.3 Access commands

The access command set comprises Req_RN, Read, Write, Lock, Kill, Access, BlockWrite, BlockErase, BlockPermalock, Authenticate, ReadBuffer, SecureComm, AuthComm, KeyUpdate, Untraceable, FileOpen, FileList, FilePrivilege, FileSetup, and TagPrivilege.

All access commands include the Tag's handle and a CRC-16. The CRC-16 is calculated over the first command-code bit to the last handle bit. A Tag in the **open** or **secured** state that receives an access command with an incorrect handle but a correct CRC-16 shall behave as specified in [Table C-30](#).

6.3.2.12.3.1 *Req_RN* (mandatory)

Interrogators and Tags shall implement the *Req_RN* command shown in [Table 6-41](#). *Req_RN* instructs a Tag to backscatter a new RN16. Both the Interrogator's command and the Tag's reply depend on the Tag's state:

- Acknowledged** state: When issuing a *Req_RN* to a Tag in the **acknowledged** state an Interrogator shall include the Tag's last backscattered RN16 as a parameter in the *Req_RN*. The *Req_RN* is protected by a CRC-16 calculated over the command code and the RN16. If a Tag receives a *Req_RN* with a correct RN16 and a correct CRC-16 then it shall generate and store a new RN16 (denoted handle), backscatter this handle, and transition to the **open** or **secured** state. The choice of ending state depends on the Tag's access password, as follows:

- Access password <> 0: Tag transitions to **open** state.
- Access password = 0: Tag transitions to **secured** state.

A Tag in the **acknowledged** state that receives a *Req_RN* with an incorrect RN16 but a correct CRC-16 shall ignore the *Req_RN* and remain in the **acknowledged** state.

- **Open or secured** state: When issuing a *Req_RN* to a Tag in the **open** or **secured** state an Interrogator shall include the Tag's handle as a parameter in the *Req_RN*. If a Tag receives the *Req_RN* with a correct handle and a correct CRC-16 then it shall generate and backscatter a new RN16, remaining in its current state (**open** or **secured**, as appropriate).

If an Interrogator wants to ensure that only one Tag is in the acknowledged state then it may issue a *Req_RN*, causing the Tag or Tags to each backscatter a handle and transition to the open or secured state (as appropriate). The Interrogator may then issue an *ACK* with handle as a parameter in the command. The Tag that receives the *ACK* with a correct handle replies as specified in [Table 6-39](#), whereas those that receive it with an incorrect handle shall return to **arbitrate**. (Note: If a Tag receives an *ACK* with an incorrect handle it returns to **arbitrate**, whereas if it receives an access command with an incorrect handle it behaves as specified in [Table C-30](#)).

The first bit of the backscattered RN16 shall be denoted the MSB; the last bit shall be denoted the LSB.

An Interrogator shall prepend a *Req_RN* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator may encapsulate a *Req_RN* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

A Tag's reply to a *Req_RN* shall be as shown in [Table 6-42](#), using the *immediate* reply type specified in [6.3.1.6.1](#). The RN16 or handle are protected by a CRC-16.

Table 6-41: *Req_RN* command

| | Command | RN | CRC |
|-------------|----------|-----------------------------|--------|
| # of bits | 8 | 16 | 16 |
| description | 11000001 | Prior RN16 or <u>handle</u> | CRC-16 |

Table 6-42: Tag reply to a *Req_RN* command

| | RN | CRC |
|-------------|---------------------------|--------|
| # of bits | 16 | 16 |
| description | <u>handle</u> or new RN16 | CRC-16 |

6.3.2.12.3.2 Read (mandatory)

Interrogators and Tags shall implement the *Read* command shown in [Table 6-44](#). A *Read* allows an Interrogator to read part or all of a Tag's Reserved memory, EPC memory, TID memory, or the currently open file in User memory. *Read* has the following fields:

- MemBank specifies whether the *Read* accesses Reserved, EPC, TID, or User memory. *Read* commands shall apply to a single memory bank. Successive *Reads* may apply to different banks.
- WordPtr specifies the starting word address for the memory read, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see [Annex A](#)).
- WordCount specifies the number of 16-bit words to read. If WordCount=00_h then a Tag shall backscatter the contents of the chosen memory bank or file starting at WordPtr and ending at the end of the memory bank or file, however:
 - if MemBank=01₂ then a Tag shall backscatter the memory contents specified in [Table 6-43](#).
 - if MemBank=10₂, and part of TID memory is untraceably hidden (see [6.3.2.12.3.16](#)), and the Interrogator has a deasserted Untraceable privilege, and the memory address specified by WordPtr is in the traceable part of TID memory, then a Tag may either (i) backscatter the

traceable part of TID memory starting at WordPtr, or (ii) treat the command's parameters as unsupported (see [Table C-30](#)), depending on the Tag manufacturer's implementation.

Table 6-43: Tag *Read* reply when WordCount=00_h and MemBank=01₂

| <u>WordPtr</u> Memory Address | Tag Implements XPC_W1? | Tag Implements XPC_W2? | What the Tag Backscatters |
|---|------------------------|------------------------|--|
| Within the StoredCRC, StoredPC, or the EPC specified by bits 10 _h –14 _h of the StoredPC | Don't care | Don't care | EPC memory starting at <u>WordPtr</u> and ending at the EPC length specified by StoredPC bits 10 _h –14 _h . |
| Within physical EPC memory but above the EPC specified by bits 10 _h –14 _h of the StoredPC | No | N/A. See note 1 | EPC memory starting at <u>WordPtr</u> and ending at the end of physical EPC memory, unless the Interrogator has a deasserted <u>Untraceable</u> privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). |
| | Yes | No | EPC memory starting at <u>WordPtr</u> and ending at the end of physical EPC memory, unless the Interrogator has a deasserted <u>Untraceable</u> privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 if <u>WordPtr</u> is less than or equal to 210 _h , physical EPC memory extends to or above 210 _h , and no error code. |
| | Yes | Yes | EPC memory starting at <u>WordPtr</u> and ending at the end of physical EPC memory, unless the Interrogator has a deasserted <u>Untraceable</u> privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 and XPC_W2 if <u>WordPtr</u> is less than or equal to 210 _h , physical EPC memory extends to or above 210 _h , and no error code. Includes XPC_W2 if <u>WordPtr</u> is equal to 220 _h and physical EPC memory extends to or above 220 _h . |
| 210 _h . Above physical EPC memory | No | N/A. See note 1 | Error code. |
| | Yes | No | XPC_W1. |
| | Yes | Yes | XPC_W1 and XPC_W2. |
| 220 _h . Above physical EPC memory | No | N/A. See note 1 | Error code. |
| | Yes | No | Error code. |
| | Yes | Yes | XPC_W2. |
| Not 210 _h or 220 _h . Above physical EPC memory. | Don't care | Don't care | Error code. |

Note 1: If a Tag does not implement an XPC_W1 then it does not implement an XPC_W2. See [6.3.2.1.2.5](#).

Note 2: Untraceably hidden memory is not readable except by an Interrogator with an asserted Untraceable privilege. See [6.3.2.12.3.16](#).

An Interrogator shall prepend a *Read* with a frame-sync (see [6.3.1.2.8](#)).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *Read* command in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

A Tag shall reply to a *Read* using the *immediate* reply type specified in [6.3.1.6.1](#). If all memory words specified in a *Read* exist, none are read-locked, all are traceable or the Interrogator has an asserted Untraceable privilege, and for User memory the Interrogator has read privileges to the currently open file (see [6.3.2.11.3](#)), then a Tag's reply to a *Read* shall be as shown in [Table 6-45](#) comprising a header (a 0-bit), the requested memory words, and the Tag's handle. The reply includes a CRC-16 calculated over the 0-bit, memory words, and handle. Otherwise the Tag shall

not execute the *Read* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Table 6-44: *Read* command

| | Command | MemBank | WordPtr | WordCount | RN | CRC |
|-------------|----------|--|--------------------------|-------------------------|---------------|--------|
| # of bits | 8 | 2 | EBV | 8 | 16 | 16 |
| description | 11000010 | 00: Reserved 01: EPC 10: TID 11: User | Starting address pointer | Number of words to read | <u>handle</u> | CRC-16 |

Table 6-45: Tag reply to a successful *Read* command

| | Header | Memory Words | RN | CRC |
|-------------|--------|--------------|---------------|--------|
| # of bits | 1 | Variable | 16 | 16 |
| description | 0 | Data | <u>handle</u> | CRC-16 |

6.3.2.12.3.3 Write (mandatory)

Interrogators and Tags shall implement the *Write* command shown in [Table 6-46](#). *Write* allows an Interrogator to write a word in a Tag's Reserved memory, EPC memory, TID memory, or the currently open file in User memory. *Write* has the following fields:

- MemBank specifies whether the *Write* occurs in Reserved, EPC, TID, or User memory. *Write* commands shall apply to a single memory bank. Successive *Writes* may apply to different banks.
- WordPtr specifies the word address for the memory write, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see [Annex A](#)).
- Data contains a 16-bit word to be written. Before each and every *Write* the Interrogator shall first issue a *Req_RN* command; the Tag replies by backscattering a new RN16. The Interrogator shall cover code the data by EXORing it with this new RN16 prior to transmission.

A Tag shall only execute a *Write* in the **open** or **secured** state. If a Tag in the **open** or **secured** state receives a *Write* before which the immediately preceding command was not a *Req_RN* then it shall not execute the *Write* and instead treat the command as invalid (see [Table C-30](#)).

If an Interrogator attempts to write to the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are permalocked; or to the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to a permalocked block in File_N, N_≥0 of User memory; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or to a file for which the Interrogator does not have sufficient privileges; then the Tag shall not execute the *Write* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend a *Write* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate a *Write* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Upon receiving an executable *Write* a Tag shall write the commanded data into memory.

A Tag shall reply to a *Write* using the *delayed* reply specified in [6.3.1.6.2](#).

Table 6-46: *Write* command

| | Command | MemBank | WordPtr | Data | RN | CRC |
|-----------|---------|---------|---------|------|----|-----|
| # of bits | 8 | 2 | EBV | 16 | 16 | 16 |

| | Command | MemBank | WordPtr | Data | RN | CRC |
|-------------|----------|--|-----------------|---------------------------|---------------|--------|
| description | 11000011 | 00: Reserved 01: EPC 10: TID 11: User | Address pointer | RN16 ⊗ word to be written | <u>handle</u> | CRC-16 |

6.3.2.12.3.4 Kill (mandatory)

Interrogators and Tags shall implement the *Kill* command shown in [Table 6-47](#). *Kill* allows an Interrogator to permanently disable a Tag. To kill a Tag, an Interrogator shall follow the kill procedure shown in [Figure 6-24](#). A Tag shall implement the password-based kill sequence shown in the left-side branch of the kill procedure in [Figure 6-24](#). A Tag that implements Interrogator or mutual authentication and *SecureComm* or *AuthComm* may also implement the authenticated-kill sequence shown in the right-side branch of the kill procedure in [Figure 6-24](#). *Kill* has the following fields:

- Password specifies half of the kill password EXORed with an RN16.

A *Kill* contains 3 RFU bits. An Interrogator shall set these bits to 000₂. A Tag shall ignore these bits. Future protocols may use these bits to expand the functionality of the *Kill* command.

An Interrogator shall prepend an unencapsulated *Kill* command with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator may encapsulate a *Kill* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Password-based kill (mandatory)

A Tag may execute a password-based kill from the **open** or **secured** state. A Tag is not required to authenticate an Interrogator for a password-based kill. To perform the kill, an Interrogator issues two successive *Kill* commands, the first containing the 16 MSBs of the Tag's kill password EXORed with an RN16, and the second containing the 16 LSBs of the Tag's kill password EXORed with a different RN16. Each EXOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each *Kill* command the Interrogator first issues a *Req_RN* to obtain a new RN16.

A Tag shall be capable of successively accepting two 16-bit subportions of the 32-bit kill password. An Interrogator shall not intersperse commands other than a *Req_RN* between the two successive *Kill* commands. If a Tag, after receiving a first *Kill*, receives any valid command other than *Req_RN* before the second *Kill* then it shall not execute the command and instead treat it as improper (see [Table C-30](#)), unless the intervening command is a *Query*, in which case the Tag shall execute the *Query* and invert its **inventoried** flag if the session parameter in the *Query* matches that in the prior session.

A Tag with a zero-valued kill password shall disallow itself from being killed by a password-based kill operation. A Tag with a zero-valued kill password shall respond to a password-based kill by not executing the kill operation and backscattering an error code, remaining in its current state. See [Figure 6-24](#).

A Tag shall reply to a first *Kill* using the *immediate* reply specified in [6.3.1.6.1](#). The Tag's first reply shall be as shown in [Table 6-48](#). The reply shall use the TRext value specified in the *Query* command that initiated the round.

A Tag shall reply to the second *Kill* using the *delayed* reply specified in [6.3.1.6.2](#). If the kill succeeds then the Tag, after sending the final reply shown in [Table 6-13](#), shall render itself silent and shall not respond to an Interrogator thereafter. If the kill does not succeed then the Interrogator may issue a *Req_RN* containing the Tag's handle to verify that the Tag is in the Interrogator's field, and may again attempt the multi-step kill procedure in [Figure 6-24](#).

If a Tag observes a properly formatted password-based *Kill*-command sequence but the kill fails (as will happen if the Interrogator sends an incorrect kill password) then the Tag shall return to **arbitrate** and may enforce a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for a password-based *Kill*-command sequence receives such a sequence during a timeout then it shall behave as though it is not killable, backscatter an error code (see [Annex I](#)), and remain in its current state.

Authenticated kill (optional)

A Tag may execute an authenticated kill from the **secured** state. A Tag shall authenticate an Interrogator via an Interrogator or mutual authentication prior to executing an authenticated kill. To perform an authenticated kill an Interrogator issues a single *Kill* command encapsulated in a *SecureComm* or *AuthComm*. The Interrogator may use any 16-bit value in the password field of the *Kill* command because a Tag shall ignore the kill password for an authenticated kill. An Interrogator is not required to issue a *Req_RN* prior to sending an encapsulated *Kill*.

A Tag shall only execute an authenticated kill if the Interrogator possesses an asserted AuthKill privilege (see [Table 6-23](#)) and the Tag is in the **secured** state. A Tag shall reply to an authenticated kill using an in-process reply (as required by a *SecureComm* or *AuthComm*), but with SenRep=1 regardless of the SenRep value actually specified in the *SecureComm* or *AuthComm*. If the kill succeeds then the Tag, after sending the final reply shown in [Table 6-13](#), shall transition to the **killed** state and not respond to an Interrogator thereafter. If the kill fails then the Tag shall remain in its current state and backscatter an error code (see [Annex I](#)), unless the Tag is in the **open** state, the Interrogator is not authenticated, or the Interrogator does not have an asserted AuthKill privilege (see [Table 6-23](#)), in which case the Tag shall return to **arbitrate** and may enforce a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for an authenticated *Kill* command receives an authenticated *Kill* command during a timeout then it shall behave as though it is not killable, backscatter an error code (see [Annex I](#)), and remain in its current state.

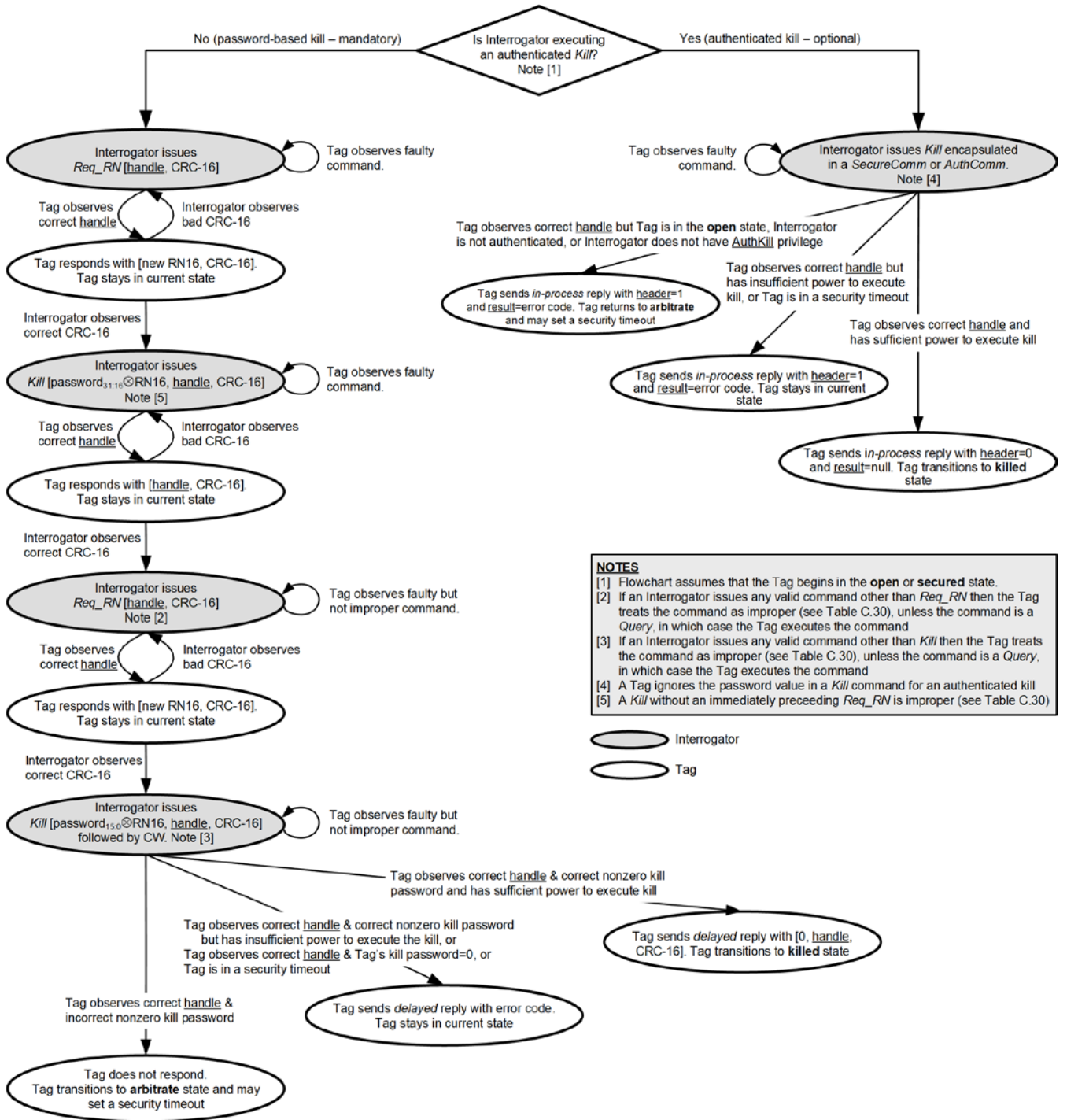
Table 6-47: *Kill* command

| | Command | Password | RFU | RN | CRC |
|-------------|----------|---|------------------|---------------|--------|
| # of bits | 8 | 16 | 3 | 16 | 16 |
| description | 11000100 | Password-based kill: (½ kill password) ⊗ RN16 Authenticated kill: any 16-bit value | 000 ₂ | <u>handle</u> | CRC-16 |

Table 6-48: Tag reply to the first *Kill* command

| | RN | CRC |
|-------------|---------------|--------|
| # of bits | 16 | 16 |
| description | <u>handle</u> | CRC-16 |

Figure 6-24: Kill procedure



6.3.2.12.3.5 Lock (mandatory)

Interrogators and Tags shall implement the *Lock* command shown in [Table 6-49](#) and [Figure 6-25](#). *Lock* allows an Interrogator to:

- Lock the kill and/or access passwords, thereby preventing or allowing subsequent reads and/or writes of those passwords,
- Lock the EPC and TID memory banks, thereby preventing or allowing subsequent writes to those banks,

- Lock File_0 of User memory, thereby preventing or allowing subsequent writes to File_0, and
- Make the lock status for the passwords, EPC memory, TID memory, and/or File_0 permanent.

Lock contains a 20-bit payload defined as follows:

- The first 10 payload bits are Mask bits. A Tag shall interpret these bit values as follows:
 - Mask=0: Ignore the associated Action field and retain the current lock setting.
 - Mask=1: Implement the associated Action field and overwrite the current lock setting.
- The last 10 payload bits are Action bits. A Tag shall interpret these bit values as follows:
 - Action=0: Deassert lock for the associated memory location.
 - Action=1: Assert lock or permalock for the associated memory location.

The functionality of the various Action fields is described in [Table 6-50](#).

The payload of a *Lock* command shall always be 20 bits in length.

If an Interrogator issues a *Lock* whose Mask and Action fields attempt to change the lock status of a nonexistent memory bank, nonexistent File_0, or nonexistent password then a Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Lock differs from *BlockPermalock* in that *Lock* reversibly or permanently locks the kill and/or access password, the EPC memory bank, the TID memory bank, and/or File_0 of User memory in a writable or unwritable state, whereas *BlockPermalock* permanently locks individual blocks of File_N, $N \geq 0$ of User memory in an unwritable state. [Table 6-55](#) specifies how a Tag reacts to a *Lock* targeting File_0 that follows a prior *BlockPermalock* (with Read/Lock=1), or vice versa.

Permalock bits, once asserted, cannot be deasserted. If a Tag receives a *Lock* whose payload attempts to deassert a previously asserted permalock bit then the Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). If a Tag receives a *Lock* whose payload attempts to reassert a previously asserted permalock bit then the Tag shall ignore this particular Action field and implement the remainder of the *Lock* payload.

An *Untraceable* command may change the values of the **L** and **U** bits in EPC memory regardless of the lock or permalock status of the EPC memory bank. See [6.3.2.12.3.16](#).

A Tag manufacturer may choose where a Tag stores its lock bits and may choose a readable portion of memory (if desired). Regardless of the location, a field-deployed Tag shall not permit an Interrogator to change its lock bits except by means of a *Lock* command.

A Tag shall implement memory locking and the *Lock* command. However, a Tag need not support all the Action fields in [Figure 6-25](#), depending on whether a Tag implements the memory location associated with the Action field and that memory location is lockable and/or unlockable. If a Tag receives a *Lock* it cannot execute because one or more memory locations do not exist, or one or more of the Action fields attempt to change a permalocked value, or one or more of the memory locations are either not lockable or not unlockable, then the Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). The only exception to this general rule is for a Tag that (a) does not support File_N, $N > 0$ and (b) whose only lock functionality is to permanently lock **all** memory (i.e. all memory banks and all passwords) at once; such a Tag shall execute a *Lock* whose payload is FFFF_h, and shall backscatter an error code for any payload other than FFFF_h.

A Tag in the **secured** state shall permit an Interrogator to write or erase memory locations with (pwd-write=1 AND permalock=0) or (pwd-read/write=1 AND permalock=0) without first issuing a *Lock* to change these fields.

An Interrogator shall prepend a *Lock* with a frame-sync (see [6.3.1.2.8](#)).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *Lock* command in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

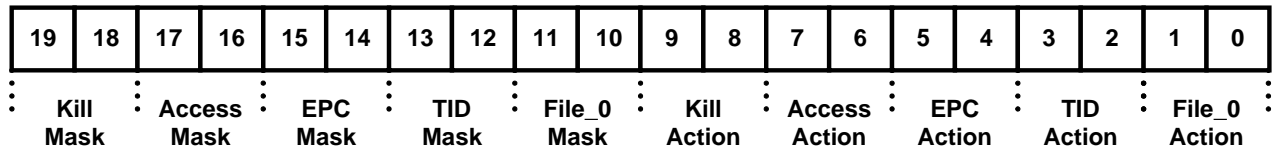
Upon receiving an executable *Lock* a Tag shall perform the commanded lock operation. A Tag shall reply to a *Lock* using the *delayed* reply specified in [6.3.1.6.2](#).

Table 6-49: Lock command

| | Command | Payload | RN | CRC |
|-------------|----------|------------------------|--------|--------|
| # of bits | 8 | 20 | 16 | 16 |
| description | 11000101 | Mask and Action Fields | handle | CRC-16 |

Figure 6-25: Lock payload and usage

Lock-Command Payload



Masks and Associated Action Fields

| | Kill pwd | | Access pwd | | EPC memory | | TID memory | | File_0 memory | |
|---------------|-----------------------|----------------|-----------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| <i>Mask</i> | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write | skip/ write |
| | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| <i>Action</i> | pwd read/ write | perma lock | pwd read/ write | perma lock | pwd write | perma lock | pwd write | perma lock | pwd write | perma lock |

Table 6-50: Lock Action-field functionality

| pwd-write | permalock | Description |
|----------------|-----------|--|
| 0 | 0 | Associated memory bank/file is writeable from either the open or secured states. |
| 0 | 1 | Associated memory bank/file is permanently writeable from either the open or secured states and may never be locked. |
| 1 | 0 | Associated memory bank/file is writeable from the secured state but not from the open state. |
| 1 | 1 | Associated memory bank/file is not writeable from any state. |
| pwd-read/write | permalock | Description |
| 0 | 0 | Associated password location is readable and writeable from either the open or secured states. |
| 0 | 1 | Associated password location is permanently readable and writeable from either the open or secured states and may never be locked. |
| 1 | 0 | Associated password location is readable and writeable from the secured state but not from the open state. |
| 1 | 1 | Associated password location is not readable or writeable from any state. |

6.3.2.12.3.6 Access (optional)

Interrogators and Tags may implement an Access command; if they do, they shall implement it as shown in [Table 6-51](#). Access allows an Interrogator to transition a Tag from the **open** to the **secured** state or, if the Tag is already in the **secured** state, to remain in **secured**. Access has the following fields:

- Password specifies half of the access password EXORed with an RN16.

To access a Tag, an Interrogator shall follow the multi-step procedure outlined in [Figure 6-26](#). Briefly, an Interrogator issues two *Access* commands, the first containing the 16 MSBs of the Tag's access password EXORed with an RN16, and the second containing the 16 LSBs of the Tag's access password EXORed with a different RN16. Each EXOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each *Access* the Interrogator first issues a *Req_RN* to obtain a new RN16.

A Tag shall be capable of successively accepting two 16-bit subportions of the 32-bit access password. An Interrogator shall not intersperse commands other than a *Req_RN* between the two successive *Access* commands. If a Tag, after receiving a first *Access*, receives any valid command other than *Req_RN* before the second *Access* then it shall not execute the command and instead treat it as improper (see [Table C-30](#)), unless the intervening command is a *Query*, in which case the Tag shall execute the *Query* and invert its **inventoried** flag if the session parameter in the *Query* matches that in the prior session.

An Interrogator shall prepend an *Access* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate an *Access* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

A Tag shall reply to an *Access* using the *immediate* reply specified in [6.3.1.6.1](#). The reply shall be as shown in [Table 6-52](#). If the *Access* is the first in the sequence then the Tag backscatters its handle to acknowledge that it received the command. If the *Access* is the second in the sequence and the received 32-bit access password is correct then the Tag backscatters its handle to acknowledge that it has executed the command successfully and has transitioned to the **secured** state; otherwise the Tag does not reply and returns to **arbitrate**. The Tag reply includes a CRC-16 calculated over the handle.

If a Tag observes a properly formatted *Access* sequence but the Interrogator sends an incorrect access password then the Tag shall return to **arbitrate** and may enforce a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for an *Access* command sequence receives such a sequence during a timeout then it shall behave as though Tag access was disallowed, backscatter an error code (see [Annex I](#)), and remain in its current state.

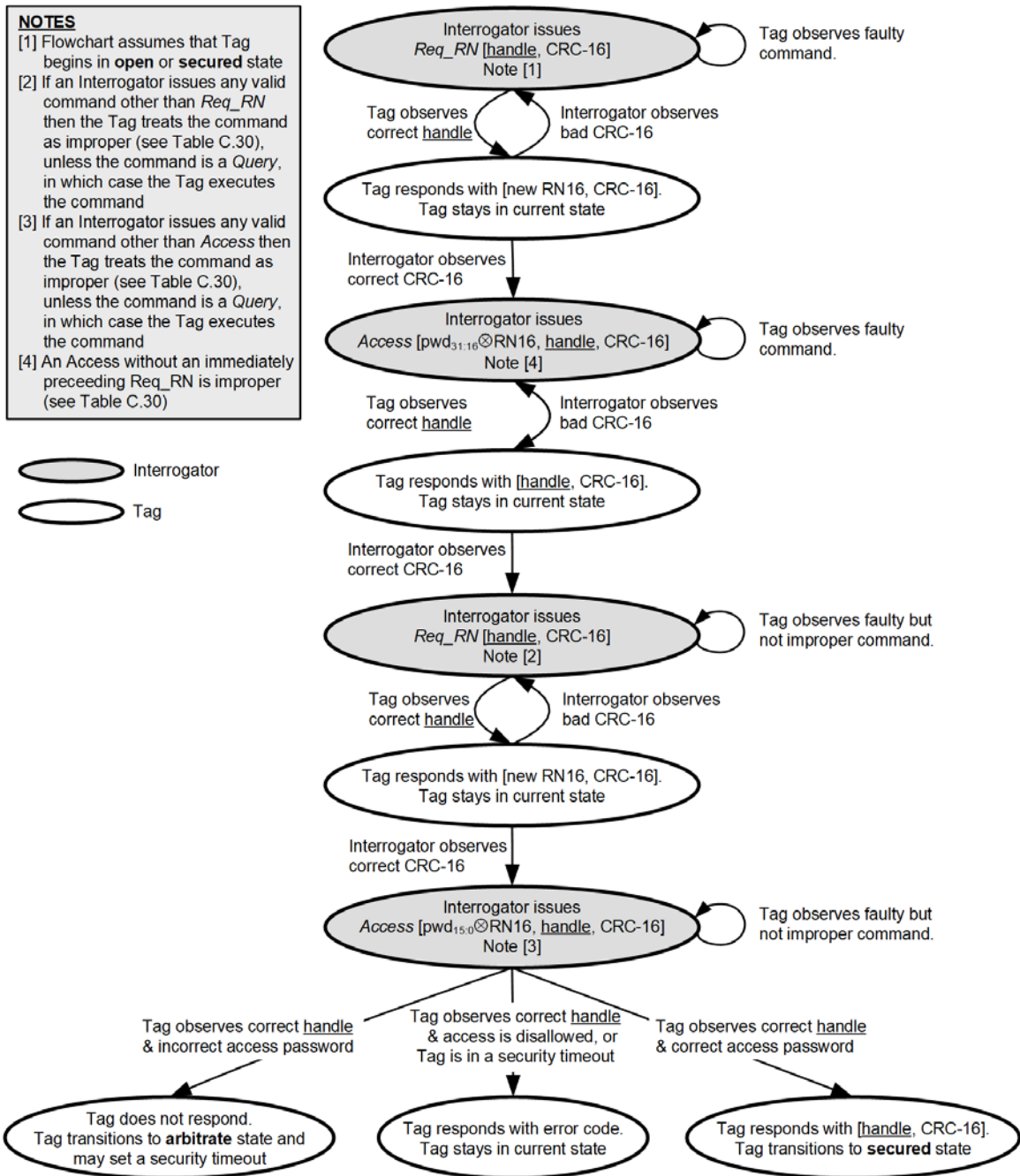
Table 6-51: *Access* command

| | Command | Password | RN | CRC |
|-------------|----------|----------------------------|---------------|--------|
| # of bits | 8 | 16 | 16 | 16 |
| description | 11000110 | (½ access password) ⊗ RN16 | <u>handle</u> | CRC-16 |

Table 6-52: Tag reply to an *Access* command

| | RN | CRC |
|-------------|---------------|--------|
| # of bits | 16 | 16 |
| description | <u>handle</u> | CRC-16 |

Figure 6-26: Access procedure



6.3.2.12.3.7 **BlockWrite (optional)**

Interrogators and Tags may implement a *BlockWrite* command; if they do, they shall implement it as shown in [Table 6-53](#). *BlockWrite* allows an Interrogator to write multiple words in a Tag’s Reserved memory, EPC memory, TID memory, or the currently open file in User memory. *BlockWrite* has the following fields:

- **MemBank** specifies whether the *BlockWrite* occurs in Reserved, EPC, TID, or User memory. *BlockWrite* commands shall apply to a single memory bank. Successive *BlockWrites* may apply to different banks.

- WordPtr specifies the starting word address for the memory write, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see [Annex A](#)).
- WordCount specifies the number of 16-bit words to be written. If WordCount=00_h then a Tag shall treat the *BlockWrite* as invalid. If WordCount=01_h then a Tag shall write a single data word.
- Data contains the 16-bit words to be written, and shall be 16 × WordCount bits in length. Unlike a *Write*, the data in a *BlockWrite* are not cover-coded, and an Interrogator need not issue a *Req_RN* before issuing a *BlockWrite*.

A Tag shall only execute a *BlockWrite* in the **open** or **secured** state.

If an Interrogator attempts to write to the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are permalocked; or to the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or to a file for which the Interrogator does not have sufficient privileges; or if WordPtr and WordCount include one or more permalocked blocks in File_N, N ≥ 0 of User memory; then the Tag shall not execute the *BlockWrite* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend a *BlockWrite* with a frame-sync (see [6.3.1.2.8](#)).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockWrite* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Upon receiving an executable *BlockWrite* a Tag shall write the commanded data into memory.

A Tag shall reply to a *BlockWrite* using the *delayed* reply specified in [6.3.1.6.2](#).

Table 6-53: *BlockWrite* command

| | Command | MemBank | WordPtr | WordCount | Data | RN | CRC |
|-------------|----------|--|--------------------------|--------------------------|--------------------|---------------|--------|
| # of bits | 8 | 2 | EBV | 8 | Variable | 16 | 16 |
| description | 11000111 | 00: Reserved 01: EPC 10: TID 11: User | Starting address pointer | Number of words to write | Data to be written | <u>handle</u> | CRC-16 |

6.3.2.12.3.8 *BlockErase* (optional)

Interrogators and Tags may implement a *BlockErase* command; if they do, they shall implement it as shown in [Table 6-54](#). *BlockErase* allows an Interrogator to erase multiple words in a Tag's Reserved memory, EPC memory, TID memory, or the currently open file in User memory. *BlockErase* has the following fields:

- MemBank specifies whether the *BlockErase* occurs in Reserved, EPC, TID, or User memory. *BlockErase* commands shall apply to a single memory bank. Successive *BlockErases* may apply to different banks.
- WordPtr specifies the starting word address for the memory erase, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see [Annex A](#)).
- WordCount specifies the number of 16-bit words to be erased. If WordCount=00_h then a Tag shall treat the *BlockErase* as invalid. If WordCount=01_h then a Tag shall erase a single data word.

A Tag shall only execute a *BlockErase* in the **open** or **secured** state.

If an Interrogator attempts to erase the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are permalocked; or the kill or access password, EPC or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or a file for which the Interrogator does not have sufficient privileges; or if

WordPtr and WordCount include one or more permalocked blocks in File_N, N_≥0 of User memory; then the Tag shall not execute the *BlockErase* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend a *BlockErase* with a frame-sync (see [6.3.1.2.8](#)).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockErase* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Upon receiving an executable *BlockErase* command a Tag shall erase the commanded memory words.

A Tag shall reply to a *BlockErase* using the *delayed* reply specified in [6.3.1.6.2](#).

Table 6-54: *BlockErase* command

| | Command | MemBank | WordPtr | WordCount | RN | CRC |
|-------------|----------|--|--------------------------|--------------------------|---------------|-----|
| # of bits | 8 | 2 | EBV | 8 | 16 | 16 |
| description | 11001000 | 00: Reserved 01: EPC 10: TID 11: User | Starting address pointer | Number of words to erase | <u>handle</u> | |

6.3.2.12.3.9 BlockPermalock (optional)

Interrogators and Tags may implement a *BlockPermalock* command; if they do, they shall implement it as shown in [Table 6-56](#). *BlockPermalock* allows an Interrogator to:

- Permalock one or more memory blocks in the currently open file of a Tag's User memory, or
- Read the permalock status of the memory blocks in the currently open file of a Tag's User memory.

A *BlockPermalock* may permalock between zero and 4080 memory blocks. The block size, which is predefined by the Tag manufacturer, is fixed at between one and 1024 words, is the same for all files, and is the same for block permalocking and file allocation. The memory blocks specified by a *BlockPermalock* need not be contiguous.

A Tag shall only execute a *BlockPermalock* in the **secured** state.

A *BlockPermalock* differs from a *Lock* in that *BlockPermalock* permanently locks individual blocks of File_N, N_≥0 of User memory in an unwriteable state whereas *Lock* reversibly or permanently locks the kill and/or access password, the EPC memory bank, the TID memory bank, and/or File_0 of User memory in a writeable or unwriteable state. [Table 6-55](#) specifies how a Tag shall behave upon receiving a *BlockPermalock* targeting File_0 that follows a prior *Lock*, or vice versa (assuming Read/Lock=1).

A *BlockPermalock* has the following fields:

- MemBank specifies whether the *BlockPermalock* applies to EPC, TID, or User memory. *BlockPermalock* commands shall apply to a single memory bank. Successive *BlockPermalocks* may apply to different memory banks. A Tag shall only execute a *BlockPermalock* command if MemBank=11 (User memory); if a Tag receives a *BlockPermalock* with MemBank<>11 then it shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these other MemBank values to expand the functionality of the *BlockPermalock* command.
- Read/Lock specifies whether a Tag backscatters the permalock status of, or permalocks, one or more blocks within the memory bank specified by MemBank. A Tag shall interpret the Read/Lock bit as follows:
 - Read/Lock=0: A Tag shall backscatter the permalock status of blocks in the specified memory bank, starting from the memory block located at BlockPtr and ending at the memory block located at BlockPtr+(16×BlockRange)-1. A Tag shall backscatter a "0" if the memory block corresponding to that bit is not permalocked and a "1" if the block is permalocked. An Interrogator omits Mask from the *BlockPermalock* when Read/Lock=0.

Table 6-55: Precedence for *Lock* and *BlockPermalock* targeting File_0

| First Command | | Second Command | | Tag Action and Response to 2 nd Command | |
|---|-----------|----------------|---|--|---|
| <i>Lock</i> | pwd-write | permalock | <i>BlockPermalock</i> (<u>Read/Lock</u> =1) | | |
| | 0 | 0 | | Permalock the blocks indicated by <u>Mask</u> ; respond as described in this section 6.3.2.12.3.9 | |
| | 0 | 1 | | Do not execute the <i>BlockPermalock</i> ; respond with an error code (Table C-30 , unsupported parameters) | |
| | 1 | 0 | | Permalock the blocks indicated by <u>Mask</u> ; respond as described in this section 6.3.2.12.3.9 | |
| | 1 | 1 | | Permalock the blocks indicated by <u>Mask</u> ; respond as described in this section 6.3.2.12.3.9 | |
| <i>BlockPermalock</i> (<u>Read/Lock</u> =1) | | <i>Lock</i> | pwd-write | permalock | |
| | | | 0 | 0 | Implement the <i>Lock</i> , but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5 |
| | | | 0 | 1 | Implement the <i>Lock</i> , but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5 |
| | | | 1 | 0 | Implement the <i>Lock</i> , but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5 |
| | | | 1 | 1 | Implement the <i>Lock</i> ; respond as described in 6.3.2.12.3.5 |

- Read/Lock=1: A Tag shall permalock those blocks in the specified memory bank that are specified by Mask, starting at BlockPtr and ending at BlockPtr+(16×BlockRange)-1.
- BlockPtr specifies the starting address for Mask, in units of 16 blocks. For example, BlockPtr=00_h indicates block 0, BlockPtr=01_h indicates block 16, BlockPtr=02_h indicates block 32. BlockPtr uses EBV formatting (see [Annex A](#)).
- BlockRange specifies the range of Mask, starting at BlockPtr and ending (16×BlockRange)-1 blocks later. If BlockRange=00_h then a Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see [Table C-30](#)).
- Mask specifies which memory blocks a Tag permalocks. Mask depends on the Read/Lock bit as follows:
 - Read/Lock=0: The Interrogator shall omit Mask from the *BlockPermalock*.
 - Read/Lock=1: The Interrogator shall include a Mask of length 16×BlockRange bits in the *BlockPermalock*. The Mask bits shall be ordered from lower-order block to higher (i.e. if BlockPtr=00_h then the leading Mask bit refers to block 0). The Tag shall interpret each bit of Mask as follows:
 - Mask bit=0: Retain the current permalock setting for the corresponding memory block.
 - Mask bit=1: Permalock the corresponding memory block. If a block is already permalocked then a Tag shall retain the current permalock setting. A memory block, once permalocked, cannot be un-permalocked.

The following example illustrates the usage of Read/Lock, BlockPtr, BlockRange, and Mask: If Read/Lock=1, BlockPtr=01_h, and BlockRange=01_h then the Tag operates on sixteen blocks starting at block 16 and ending at block 31, permalocking those blocks whose corresponding bits are asserted in Mask.

A *BlockPermalock* contains 8 RFU bits. An Interrogator shall set these bits to 00_h. A Tag in the **secured** state that receives a *BlockPermalock* with nonzero RFU bits shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *BlockPermalock* command's functionality.

If a Tag receives a *BlockPermalock* that it cannot execute because User memory does not exist, or User memory is untraceably hidden and the Interrogator has a deasserted *Untraceable* privilege, or in which one of the asserted *Mask* bits references a non-existent memory block, or because the Interrogator has insufficient file privileges (see [6.3.2.11.3](#)) then the Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). A Tag shall treat as invalid a *BlockPermalock* in which *Read/Lock*=0 but *Mask* is not omitted, or a *BlockPermalock* in which *Read/Lock*=1 but *Mask* has a length not equal to $16 \times \text{BlockRange}$ bits (see [Table C-30](#)).

Certain Tags, depending on the Tag manufacturer's implementation, may be unable to execute a *BlockPermalock* with certain *BlockPtr* and *BlockRange* values, in which case the Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Because a Tag contains information in its TID memory that an Interrogator can use to identify the optional features that the Tag supports (see [6.3.2.1.3](#)), this protocol recommends that an Interrogator read a Tag's TID memory prior to issuing a *BlockPermalock*.

If an Interrogator issues a *BlockPermalock* in which *BlockPtr* and *BlockRange* specify one or more nonexistent blocks, but *Mask* only asserts permalocking on existent blocks, then the Tag shall execute the *BlockPermalock*.

An Interrogator shall prepend a *BlockPermalock* with a frame-sync (see [6.3.1.2.8](#)).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockPermalock* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

Upon receiving an executable *BlockPermalock* a Tag shall perform the requested operation, unless the Tag does not support block permalocking in which case it shall treat the command as invalid (see [Table C-30](#)).

If *Read/Lock*=0 then a Tag shall reply to a *BlockPermalock* using the *immediate* reply type specified in [6.3.1.6.1](#). If the Tag is able to execute the *BlockPermalock* then its reply shall be as shown in [Table 6-57](#) comprising a header (a 0-bit), the requested permalock bits, and the Tag's *handle*. The reply includes a CRC-16 calculated over the 0-bit, permalock bits, and *handle*. If the Tag is unable to execute the *BlockPermalock* then it shall backscatter an error code (see [Table C-30](#), unsupported parameters) rather than the reply shown in [Table 6-57](#). The Tag's reply when *Read/Lock*=0 shall use the preamble specified by the *TRExt* value in the *Query* that initiated the inventory round.

If *Read/Lock*=1 then a Tag shall reply to a *BlockPermalock* using the *delayed* reply specified in [6.3.1.6.2](#).

Table 6-56: *BlockPermalock* command

| | Command | RFU | Read/Lock | MemBank | BlockPtr | BlockRange | Mask | RN | CRC |
|-------------|----------|-----------------|-------------------------|---|---|--|--|---------------|--------|
| # of bits | 8 | 8 | 1 | 2 | EBV | 8 | Variable | 16 | 16 |
| description | 11001001 | 00 _h | 0: Read 1: Permalock | 00: RFU 01: EPC 10: TID 11: User | <i>Mask</i> starting address, specified in units of 16 blocks | <i>Mask</i> range, specified in units of 16 blocks | 0: Retain current permalock setting 1: Assert permalock | <i>handle</i> | CRC-16 |

Table 6-57: Tag reply to a successful *BlockPermalock* command with *Read/Lock*=0

| | Header | Data | RN | CRC |
|-------------|--------|----------------|---------------|--------|
| # of bits | 1 | Variable | 16 | 16 |
| description | 0 | Permalock bits | <i>handle</i> | CRC-16 |

6.3.2.12.3.10 Authenticate (optional)

Interrogators and Tags may implement the *Authenticate* command; if they do, they shall implement it as shown in [Table 6-58](#). *Authenticate* allows an Interrogator to perform Tag, Interrogator, or mutual authentication. The generic nature of the *Authenticate* command allows it to support a variety of cryptographic suites. The *CSI* specified in an *Authenticate* selects one cryptographic suite from among those supported by the Tag. The number of *Authenticate* commands required to

implement an authentication depends on the authentication type and on the chosen cryptographic suite. A Tag only executes an *Authenticate* in the **open** or **secured** state. *Authenticate* has the following fields:

- **SenRep** specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- **IncRepLen** specifies whether a Tag omits or includes length in its reply. If **IncRepLen**=0 then a Tag omits length from its reply; if **IncRepLen**=1 then the Tag includes length in its reply.
- **CSI** selects the cryptographic suite that Tag and Interrogator use for the authentication as well as for all subsequent communications (until the Interrogator initiates another authentication with a different **CSI** or the Tag leaves the **open** or **secured** state).
- **Length** is the message length in bits.
- **Message** includes parameters for the authentication.

An *Authenticate* contains 2 RFU bits. An Interrogator shall set these bits to 00. A Tag in the **open** or **secured** states that receives an *Authenticate* with nonzero RFU bits shall not execute the *Authenticate* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *Authenticate* command's functionality.

An Interrogator shall prepend an *Authenticate* with a frame-sync (see [6.3.1.2.8](#)).

An Interrogator shall not encapsulate an *Authenticate* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)).

If a Tag supports the *Authenticate* command then it shall implement the security (**S**) indicator (see [6.3.2.1.3](#)).

The cryptographic suite specifies message formatting, the number of steps in an authentication, whether an authentication implements wait states, the behavior if Tag or Interrogator cannot complete a computation, and the behavior in the event of an incorrect cryptographic response. It specifies the formatting of the Tag's response for both a successful and an unsuccessful authentication. It may include parameters, such as a key, that affect pre- and post-authenticated communications. It may contain information about how Tag and Interrogator derive session keys for subsequent communications. See [Annex M](#) for the parameters specified by a cryptographic suite.

An *Authenticate* command shall use the *in-process* reply specified in [6.3.1.6.3](#). The parameters that a Tag includes in its response are specified by the cryptographic suite. See [Annex M](#).

If a Tag receives an *Authenticate* specifying an unsupported **CSI**, an improperly formatted or not-executable message, or an improper cryptographic parameter then the Tag shall not execute the *Authenticate* and instead treat the command's parameters as unsupported (see [Table C-30](#)). If a Tag in the **secured** state receives an *Authenticate* that begins a new authentication, such as if the *Authenticate* contains a changed **CSI**, then the Tag shall transition to the **open** state, discontinue using and reset the current cryptographic engine, and begin the new authentication.

If a Tag receives a properly formatted *Authenticate* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for the *Authenticate* command receives an *Authenticate* during a timeout then it shall reject the command, backscatter an error code (see [Annex I](#)), and remain in its current state.

Table 6-58: *Authenticate* command

| | Command | RFU | SenRep | IncRepLen | CSI | Length | Message | RN | CRC |
|--------------------|----------|-----|-------------------|---|------------|------------------------------------|---|---------------|--------|
| # of bits | 8 | 2 | 1 | 1 | 8 | 12 | Variable | 16 | 16 |
| description | 11010101 | 00 | 0:store 1:send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | <u>CSI</u> | <u>length</u> of <u>message</u> | <u>message</u> (depends on <u>CSI</u>) | <u>handle</u> | CRC-16 |

6.3.2.12.3.11 AuthComm (optional)

Interrogators and Tags may implement the *AuthComm* command; if they do, they shall implement it as shown in [Table 6-59](#). *AuthComm* allows authenticated communications from R=>T by encapsulating another command and typically also a MAC in the *AuthComm*'s message field. [Table 6-28](#) shows the commands that an *AuthComm* may encapsulate. The generic nature of an *AuthComm* allows it to support a wide variety of cryptographic suites. An *AuthComm* shall always be preceded by a Tag, Interrogator, or mutual authentication via an *Authenticate* or a *Challenge*. The cryptographic suite indicated by the CSI in the *Authenticate* or *Challenge* that preceded the *AuthComm* specifies message and reply formatting. A Tag may include a MAC in its reply, again as specified by the cryptographic suite. A Tag only executes an *AuthComm* in the **open** or **secured** state. *AuthComm* has the following fields:

- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- Message includes the encapsulated command and other parameters (such as a MAC) as specified by the cryptographic suite. An Interrogator shall remove the command's preamble, handle, and CRC before encapsulating it in an *AuthComm*. The encapsulated command shall not be encrypted or obscured.

An *AuthComm* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **open** or **secured** states that receives an *AuthComm* with nonzero RFU bits shall not execute the *AuthComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *AuthComm* command's functionality.

A Tag in the **open** or **secured** states that receives an *AuthComm* encapsulating a disallowed command, an unsupported command, or a command that does not support encapsulation (see [Table 6-28](#)) shall not execute the *AuthComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend an *AuthComm* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall only accept an *AuthComm* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute an *AuthComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

When processing an *AuthComm* a Tag shall first perform the functions/analysis/state-change/error-handling for the *AuthComm* itself and then, if the *AuthComm* is successful, the functions/analysis/state-change/error-handling for the command encapsulated in the *AuthComm*'s message field. In some instances, such as when an *AuthComm* encapsulates an authenticated *Kill*, the Tag may change state in response to the encapsulated command even though it did not change state in response to the *AuthComm* itself.

A Tag shall reply to an *AuthComm* using the *in-process* reply specified in [6.3.1.6.3](#). The cryptographic suite shall specify the parameters that a Tag includes in its response, including at least the reply for the encapsulated command minus preamble, handle, and CRC. For example, if the encapsulated command is a *Read* then the reply includes at least the read data or an error code as appropriate for a *Read*. Unlike other commands that use an *in-process* reply, *AuthComm* does not include a SenRep field because a Tag shall always send (i.e. never store) its reply to an *AuthComm*.

An *AuthComm* may exhibit behavior different from other commands because an *AuthComm* itself may succeed or fail or the encapsulated command, such as a *Lock*, may succeed or fail. Done and header in the reply of [Table 6-14](#) indicate success or failure of the *AuthComm*. Response in the reply of [Table 6-14](#) indicates success or failure of the encapsulated command. For example, suppose a Tag receives an *AuthComm* with IncRepLen=1 encapsulating a command whose reply type is *delayed* (such as a *Lock*). Upon successfully completing the *Lock* the Tag's reply will be as shown in [Table 6-14](#) with done=1 and header=0, indicating that the *AuthComm* executed successfully, and length=0003_n and result=0, indicating that the *Lock* completed successfully. Note that this example presumes that the Tag was in the **secured** state; if the Tag was in the **open** state then the Tag would not execute the *AuthComm* and instead treat the command's parameters as unsupported see [Table C-30](#).

If a Tag receives a properly formatted *AuthComm* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for the *AuthComm*

command receives an *AuthComm* during a timeout then it shall reject the command, backscatter an error code (see [Annex I](#)), and remain in its current state.

Table 6-59: *AuthComm* command

| | Command | RFU | IncRepLen | Message | RN | CRC |
|-------------|----------|-----------------|---|----------------|---------------|--------|
| # of bits | 8 | 2 | 1 | Variable | 16 | 16 |
| description | 11010111 | 00 ₂ | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | <u>message</u> | <u>handle</u> | CRC-16 |

6.3.2.12.3.12 *SecureComm* (optional)

Interrogators and Tags may implement the *SecureComm* command; if they do, they shall implement it as shown in [Table 6-60](#). *SecureComm* allows encrypted communications from R=>T by encapsulating another, encrypted command in the *SecureComm*'s message field. [Table 6-28](#) shows the commands that a *SecureComm* may encapsulate. The generic nature of a *SecureComm* allows it to support a wide variety of cryptographic suites. A *SecureComm* shall always be preceded by a Tag, Interrogator, or mutual authentication via an *Authenticate* or a *Challenge*. The cryptographic suite indicated by the CSI in the *Authenticate* or *Challenge* that preceded the *SecureComm* specifies message and reply formatting. A Tag may encrypt and/or include a MAC in its reply, again as specified by the cryptographic suite. A Tag only executes a *SecureComm* in the **open** or **secured** state. *SecureComm* has the following fields:

- SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- Length is the message length, in bits.
- Message includes the encapsulated command and other parameters (such as a MAC) as specified by the cryptographic suite. An Interrogator shall remove the command's preamble, handle, and CRC before encapsulating it in a *SecureComm*. The encapsulated command shall be encrypted.

A *SecureComm* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **open** or **secured** states that receives a *SecureComm* with nonzero RFU bits shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *SecureComm* command's functionality.

A Tag in the **open** or **secured** states that receives a *SecureComm* encapsulating a disallowed command, an unsupported command, or a command that does not support encapsulation (see [Table 6-28](#)) shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend a *SecureComm* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall only accept a *SecureComm* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute a *SecureComm* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

When processing a *SecureComm* a Tag shall first perform the functions/analysis/state-change/error-handling for the *SecureComm* itself and then, if the *SecureComm* is successful, the functions/analysis/state-change/error-handling for the command encapsulated in the *SecureComm*'s message field. In some instances, such as when a *SecureComm* encapsulates an authenticated *Kill*, the Tag may change state in response to the encapsulated command even though it did not change state in response to the *SecureComm* itself.

A Tag shall reply to a *SecureComm* using the *in-process* reply specified in [6.3.1.6.3](#). The cryptographic suite shall specify the parameters that a Tag includes in its response, including at least the reply for the encapsulated command minus preamble, handle, and CRC. For example, if

the encapsulated command is a *Read* then the reply includes at least the read data or an error code as appropriate for a *Read*.

A *SecureComm* may exhibit behavior different from other commands because a *SecureComm* itself may succeed or fail or the encapsulated command, such as a *Lock*, may succeed or fail. Done and header in the reply of [Table 6-14](#) indicate success or failure of the *SecureComm*. Response in the reply of [Table 6-14](#) indicates success or failure of the encapsulated command. For example, suppose a Tag receives a *SecureComm* with IncRepLen=1 and SenRep=1 encapsulating a command whose reply type is *delayed* (such as a *Lock*). Upon successfully completing the *Lock* the Tag's reply will be as in [Table 6-14](#) with done=1 and header=0, indicating the *SecureComm* executed successfully, and length=0003_h and result=0, indicating that the *Lock* completed successfully. Alternatively, if SenRep=0 then the reply will be as shown in [Table 6-14](#) with done=1 and header=0, indicating the *SecureComm* executed successfully, and length=0003_h and result=null, indicating that the *Lock* completed successfully and result (0₂) is in the ResponseBuffer. In this latter case the Tag asserts **C** in XPC_W1 to indicate that the ResponseBuffer contains a computed result. This example presumes that the Tag was in the **secured** state; if it was in the **open** state then the *SecureComm* would succeed but the reply to the *Lock* would be an error code.

If a Tag receives a properly formatted *SecureComm* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for the *SecureComm* command receives a *SecureComm* during a timeout then it shall reject the command, backscatter an error code (see [Annex I](#)), and remain in its current state.

Table 6-60: *SecureComm* command

| | Command | RFU | SenRep | IncRepLen | Length | Message | RN | CRC |
|-------------|----------|-----|---------------------|---|---------------------------------------|----------------|---------------|------------|
| # of bits | 8 | 2 | 1 | 1 | 12 | Variable | 16 | 16 |
| description | 11010110 | 00 | 0: store 1: send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | <u>length</u> of <u>message</u> | <u>message</u> | <u>handle</u> | CRC- 16 |

6.3.2.12.3.13 *KeyUpdate* (optional)

Interrogators and Tags may implement the *KeyUpdate* command; if they do, they shall implement it as shown in [Table 6-61](#). *KeyUpdate* allows an Interrogator to write or overwrite a key stored in a Tag. The generic nature of a *KeyUpdate* allows it to support a wide variety of cryptographic suites. A *KeyUpdate* shall always be preceded by an Interrogator or mutual authentication via an *Authenticate*. The cryptographic suite indicated by the CSI in the *Authenticate* that preceded the *KeyUpdate* specifies message and reply formatting. A Tag only executes a *KeyUpdate* in the **secured** state. *KeyUpdate* has the following fields:

- SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- Length is the message length, in bits.
- KeyID specifies the key to be written or updated.
- Message is or contains the key. Message may contain other parameters (such as a MAC) as specified by the cryptographic suite. Message may be encrypted.

A *KeyUpdate* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **secured** state that receives a *KeyUpdate* with nonzero RFU bits shall not execute the *KeyUpdate* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *KeyUpdate* command's functionality.

An Interrogator may encapsulate a *KeyUpdate* in a *SecureComm* or an *AuthComm* (see [Table 6-28](#)). If a cryptographic suite requires that *KeyUpdate* be encapsulated in a *SecureComm* then message in the *KeyUpdate* need not be encrypted. If a cryptographic suite allows sending a *KeyUpdate* in an *AuthComm* or without encapsulation then message in the *KeyUpdate* shall be encrypted.

A Tag in the **secured** state shall only write a key if (a) the Interrogator authenticated itself as a crypto superuser and KeyID is assigned to the same cryptographic suite as that specified by CSI in the *Authenticate* command that preceded the *KeyUpdate*, or (b) KeyID is the same as that used by the Interrogator to authenticate itself. In all other instances the Tag shall not execute the *KeyUpdate* and instead treat the command's parameters as unsupported (see [Table C-30](#)). See [6.3.2.11.2](#) for a description of Tag privileges and the crypto superuser privilege.

Upon receiving an executable *KeyUpdate* 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 to the prior stored key. A Tag may meet this latter requirement by, for example, double-buffering the write operation.

An Interrogator shall prepend an unencapsulated *KeyUpdate* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall only accept a *KeyUpdate* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute a *KeyUpdate* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag shall reply to a *KeyUpdate* using the *in-process* reply specified in [6.3.1.6.3](#). The cryptographic suite shall specify the parameters that a Tag includes in its response.

If a Tag receives a properly formatted *KeyUpdate* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in [6.3.2.5](#). If a Tag that supports security timeouts for the *KeyUpdate* command receives a *KeyUpdate* during a timeout then it shall reject the command, backscatter an error code (see [Annex I](#)), and remain in its current state.

Table 6-61: KeyUpdate command

| | Command | RFU | SenRep | IncRepLen | KeyID | Length | Message | RN | CRC |
|-------------|----------------------|-----|---------------------|---|--------------|---------------------------------------|----------------|---------------|------------|
| # of bits | 16 | 2 | 1 | 1 | 8 | 12 | Variable | 16 | 16 |
| description | 11100010 00000010 | 00 | 0: store 1: send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | <u>KeyID</u> | <u>length</u> of <u>message</u> | <u>message</u> | <u>handle</u> | CRC- 16 |

6.3.2.12.3.14 TagPrivilege (optional)

Interrogators and Tags may implement the *TagPrivilege* command; if they do, they shall implement it as shown in [Table 6-63](#). *TagPrivilege* allows an Interrogator to read or modify the Tag privileges in [Table 6-22](#) or [Table 6-23](#) for the access password or for a key, respectively. A Tag only executes a *TagPrivilege* in the **secured** state. *TagPrivilege* has the following fields:

- SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- Action specifies whether the Interrogator is reading privileges or modifying them. Action=0 indicates read; Action=1 indicates modify.
- Target specifies whether the Interrogator is targeting the access password or a key. If Target=0 then the Tag reads or modifies the access-password privileges; if Target=1 then the Tag reads or modifies the Tag privileges for the key indicated by KeyID.
- KeyID specifies the key for the privileges being read or written.
- Privilege specifies values for each of the 16 Tag privileges in [Table 6-22](#) or [Table 6-23](#) when an Interrogator is modifying a privilege (i.e. when Action=1).

A *TagPrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **secured** state that receives a *TagPrivilege* containing nonzero RFU bits shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *TagPrivilege* command's functionality.

An unauthenticated Interrogator may issue a *TagPrivilege*; if it does then it shall issue the *TagPrivilege* without encapsulation and with Target=0 (i.e. specifying the access password).

An authenticated Interrogator shall encapsulate a *TagPrivilege* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *TagPrivilege* from an authenticated Interrogator then it shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag in the **secured** state shall only read or modify the access-password privileges if the Interrogator supplied the correct access password and is not attempting to assert a deasserted privilege. In all other instances the Tag shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag in the **secured** state shall only read or modify a key's privileges if (a) the Interrogator authenticated itself as a crypto superuser and KeyID is assigned to the same cryptographic suite as that specified by CSI in the *Authenticate* command that preceded the *TagPrivilege*, or (b) KeyID is the same as that used by the Interrogator to authenticate itself and the Interrogator is not attempting to assert a deasserted privilege.

If an Interrogator specifies Action=0 in a *TagPrivilege* then it may use any value for privilege. A Tag shall ignore privilege when Action=0.

If an Interrogator specifies Target=0 in a *TagPrivilege* then it may use any value for the KeyID. If Tag receives a *TagPrivilege* with Target=0 then it shall ignore the value that the Interrogator supplies for KeyID.

Upon receiving an executable *TagPrivilege* with Action=1 a Tag shall overwrite the old privileges with the new privileges. If the Tag does not write the new privileges successfully then it shall revert to the prior stored privileges. A Tag may meet this latter requirement by, for example, double-buffering the write operation.

A Tag in the **secured** state that receives a *TagPrivilege* which attempts to assert one or more RFU privilege bits or to change an unchangeable privilege value shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An Interrogator shall prepend an unencapsulated *TagPrivilege* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall reply to a *TagPrivilege* using the *in-process* reply specified in [6.3.1.6.3](#). The Tag's response shall be as shown in [Table 6-62](#) for Action=0 or Action=1. The response includes Target, the Interrogator-supplied KeyID, and the current privileges (newly written if Action=1 and the Tag wrote the new privileges successfully).

Table 6-62: *TagPrivilege* command

| | Command | RFU | SenRep | IncRepLen | Action | Target | KeyID | Privileges | RN | CRC |
|-------------|----------------------|-----|---------------------|---|----------------------|-------------------------|--------------|------------------|---------------|--------|
| # of bits | 16 | 2 | 1 | 1 | 1 | 1 | 8 | 16 | 16 | 16 |
| description | 11100010 00000011 | 00 | 0: store 1: send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | 0: read 1: modify | 0: access pwd 1: key | <u>KeyID</u> | <u>privilege</u> | <u>handle</u> | CRC-16 |

Table 6-63: Tag reply to a successful *TagPrivilege* command

| | Target | KeyID | Privileges |
|-------------|-------------------------|--------------|------------------|
| # of bits | 1 | 8 | 16 |
| description | 0: access pwd 1: key | <u>KeyID</u> | <u>privilege</u> |

6.3.2.12.3.15 ReadBuffer (optional)

Interrogators and Tags may implement the *ReadBuffer* command; if they do, they shall implement it as shown in [Table 6-64](#). *ReadBuffer* allows an Interrogator to read data stored in a Tag's ResponseBuffer. A Tag only executes a *ReadBuffer* in the **open** or **secured** state and only if the Tag's **C** flag is asserted. *ReadBuffer* has the following fields:

- WordPtr specifies the starting word address for the read. For example, WordPtr=000_h specifies the first 16-bit memory word, WordPtr=001_h specifies the second 16-bit memory word, etc.
- BitCount specifies the number of bits to read. If BitCount=000_h then a Tag shall backscatter the contents of the ResponseBuffer starting at WordPtr and ending at the end of the allocated ResponseBuffer.

A *ReadBuffer* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **open** or **secured** states that receives a *ReadBuffer* with nonzero RFU bits shall not execute the *ReadBuffer* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *ReadBuffer* command's functionality. An Interrogator may encapsulate a *ReadBuffer* in an *AuthComm* but shall not encapsulate it in a *SecureComm* (see [Table 6-28](#)).

If a Tag implements a ResponseBuffer then that Tag shall implement the *ReadBuffer* command.

An Interrogator shall prepend an unencapsulated *ReadBuffer* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall reply to a *ReadBuffer* using the *immediate* reply specified in [6.3.1.6.1](#). If **C**=1 and the memory bits specified in the *ReadBuffer* exist then the Tag's reply shall be as shown in [Table 6-65](#) including a header (a 0-bit), the data bits, and the Tag's handle. The reply includes a CRC-16 calculated over the 0-bit, data bits, and handle. If one or more of the memory bits specified in the *ReadBuffer* do not exist, or if the **C** flag in XPC_W1 is zero-valued, then the Tag shall not execute the *ReadBuffer* and instead backscatter an error code (see [Table C-30](#), unsupported parameters) within time T₁ in [Table 6-16](#) rather than the reply shown in [Table 6-65](#).

Table 6-64: *ReadBuffer* command

| | Command | RFU | WordPtr | BitCount | RN | CRC |
|-------------|----------|-----|--------------------------|------------------------|---------------|--------|
| # of bits | 8 | 2 | 12 | 12 | 16 | 16 |
| description | 11010010 | 00 | Starting address pointer | Number of bits to read | <u>handle</u> | CRC-16 |

Table 6-65: Tag reply to a successful *ReadBuffer* command

| | Header | Data Bits | RN | CRC |
|-------------|--------|-------------|---------------|--------|
| # of bits | 1 | Variable | 16 | 16 |
| description | 0 | <u>data</u> | <u>handle</u> | CRC-16 |

6.3.2.12.3.16 *Untraceable* (optional)

Interrogators and Tags may implement the *Untraceable* command; if they do, they shall implement it as shown in [Table 6-66](#). *Untraceable* allows an Interrogator with an asserted Untraceable privilege to instruct a Tag to (a) alter the **L** and **U** bits in EPC memory, (b) hide memory from Interrogators with a deasserted Untraceable privilege, and/or (c) reduce its operating range for all Interrogators. The memory that a Tag may hide includes words of EPC memory, the Tag serialization in TID memory, all of TID memory, and/or User memory (File_0 and above). *Untraceable* and traceable Tags behave identically from a state-machine and command-response perspective; the difference between them is (a) the memory the Tag exposes to an Interrogator with a deasserted Untraceable privilege and/or (b) the Tag's operating range. A Tag only executes an *Untraceable* in the **secured** state. *Untraceable* has the following fields:

- U specifies a value for the **U** bit in XPC_W1 (see [6.3.2.1.2.2](#)). Upon receiving an *Untraceable* command a Tag that supports the **U** bit overwrites bit 21C_n of XPC_W1 with the provided U value regardless of the lock or permalock status of EPC memory. If the Tag does not support the **U** bit then the Tag shall ignore the provided U value but continue to process the remainder of the *Untraceable*.
- EPC includes a show/hide bit (MSB) and 5 length bits (5 LSBs). These fields operate independently.
 - Show/hide specifies whether a Tag untraceably hides part of EPC memory. If show/hide=0₂ then a Tag exposes EPC memory. If show/hide=1₂ then a Tag untraceably hides EPC

memory above that set by its EPC length field (i.e. StoredPC bits $10_h - 14_h$) to bit $20F_h$ (inclusive).

- Length specifies a new EPC length field (**L** bits). Upon receiving an *Untraceable* command a Tag overwrites its EPC length field (StoredPC bits $10_h - 14_h$) with the provided length bits regardless of the lock or permalock status of EPC memory. In response to subsequent *ACKs* the Tag backscatters an EPC whose length is set by the new length bits.
- TID specifies the TID memory that a Tag untraceably hides. If TID= 00_2 then a Tag exposes TID memory. If TID= 01_2 and a Tag's allocation class identifier (see [6.3.2.1.3](#)) is $E0_h$ then the Tag untraceably hides TID memory above 10_h , inclusive; if the Tag's allocation class identifier is $E2_h$ then the Tag untraceably hides TID memory above 20_h , inclusive. If TID= 10_2 then the Tag untraceably hides all of TID memory. TID= 11_2 is RFU.
- User specifies whether a Tag untraceably hides User memory. If User= 0_2 then the Tag exposes User memory. If User= 1_2 then the Tag untraceably hides User memory (i.e. hides File_0 and above).
- Range specifies a Tag's operating range. If range= 00_2 then the Tag persistently enables normal operating range. If range= 10_2 then the Tag persistently enables reduced operating range. If range= 01_2 then the Tag temporarily toggles its operating range (if normal then to reduced; if reduced then to normal) but reverts to its prior persistent operating range when the Tag loses power. Temporary toggling allows an Interrogator to confirm that a Tag is still readable before committing range-reduced untraceability to the Tag's nonvolatile memory (by sending a subsequent *Untraceable* with range= 10_2). Range= 11_2 is RFU. A Tag shall execute a range change prior to replying to the *Untraceable*. The range-reduction details, including its magnitude and the commands to which it applies, are manufacturer-defined. If a Tag does not support range reduction then it shall ignore range but continue to process the remainder of the *Untraceable*.

A Tag is not required to support all of these field values in an *Untraceable*. If a Tag receives an *Untraceable* containing field values it does not support then the Tag treats the command's parameters as unsupported (see [Table C.30](#)), unless the unsupported fields are U or range, in which case the Tag behaves as described above.

An *Untraceable* contains 2 RFU bits. An Interrogator shall set these bits to 00_2 . A Tag in the **secured** state that receives an *Untraceable* with nonzero RFU bits, TID= 11_2 , or range= 11_2 shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *Untraceable* command's functionality.

If a Tag in the **secured** state receives an *Untraceable* from an Interrogator with an asserted Untraceable privilege then it shall execute the command; if the Interrogator has a deasserted Untraceable privilege then the Tag shall not execute the command and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An unauthenticated Interrogator may issue an *Untraceable* without encapsulation. An authenticated Interrogator shall encapsulate an *Untraceable* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *Untraceable* from an authenticated Interrogator then it shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Untraceable commands shall be atomic, meaning that a Tag, upon receiving an executable *Untraceable*, shall discard its prior memory and range settings and implement the new ones.

If an *Untraceable* command modifies a Tag's EPC length field and the Tag computes its StoredCRC at powerup then the StoredCRC is likely to be incorrect until the Interrogator power-cycles the Tag. See [6.3.2.1.2.1](#).

If a Tag supports only **XI**= 0_2 then the length bits in an *Untraceable* may have any 5-bit value. If the Tag supports **XI**= 1_2 then the maximum length-bit value is 11101_2 . A Tag that supports **XI**= 1_2 shall not execute an *Untraceable* that specifies length bits greater than 11101_2 and shall instead treat the command's parameters as unsupported (see [Table C-30](#)). Regardless of these absolute bounds on length, if an *Untraceable* specifies a length value that a Tag does not support then the Tag shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

A Tag that is operating with reduced range shall do so for all commands regardless of whether an Interrogator has an asserted or a deasserted Untraceable privilege.

A Tag shall execute supported access commands that operate on untraceably hidden memory if the commanding Interrogator has an asserted Untraceable privilege, but shall not execute these commands if the Interrogator has a deasserted Untraceable privilege. In the latter case a Tag shall behave as though untraceably hidden memory does not exist and treat the commands' parameters as unsupported (see [Table C-30](#)). As an example, suppose that a Tag's User memory is untraceably hidden. The Tag may execute a *FileOpen* from an Interrogator with an asserted Untraceable privilege but not from an Interrogator with a deasserted Untraceable privilege.

A Tag that is untraceably hiding EPC memory shall not include any of the untraceably hidden EPC memory bits when replying to an *ACK*.

This protocol recommends that an Interrogator permalock the EPC memory bank prior to untraceably hiding part or all of EPC memory. Absent such permalocking, an Interrogator without the Untraceable privilege may subsequently alter the Tag's EPC length field and expose untraceably hidden memory.

A Tag treats as not-matching a *Select* command whose Mask includes untraceably hidden memory.

If a Tag computes its **UMI** then the untraceability status of User memory does not change the **UMI** value.

An Interrogator shall prepend an unencapsulated *Untraceable* with a frame-sync (see [6.3.1.2.8](#)).

This protocol allows a Tag manufacturer to implement irreversible untraceability whereby a memory region, once untraceably hidden, cannot be re-exposed and/or the Tag's operating range, once reduced, cannot be restored to normal. The details of this irreversible untraceability, including whether a Tag with irreversibly hidden memory will still alter its operating range, and vice versa, shall be Tag-manufacturer defined.

This protocol allows a Tag manufacturer to configure a Tag to only execute an *Untraceable* at short range. This protocol also allows a Tag manufacturer to configure such a Tag with a zero-valued access password and an asserted Untraceable privilege for the access password, in which case the short-range feature provides the only protection against illicit use of the *Untraceable* command.

A Tag shall reply to an *Untraceable* using the *delayed* reply specified in [6.3.1.6.2](#). Upon receiving an executable *Untraceable* a Tag shall perform the specified actions. If a Tag receives an *Untraceable* (i) with field values it supports but nonetheless cannot execute, such as if the *Untraceable* instructs the Tag to expose an irreversibly hidden portion of Tag memory or the Interrogator has a deasserted Untraceable privilege, or (ii) with field values it does not support, unless the unsupported fields are U or range, then the Tag shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Table 6-66: *Untraceable* command

| | Command | RF U | U | EPC | TID | User | Range | RN | CRC |
|-------------|------------------------------|------|--|--|---|--------------------|--|-------------------|--------|
| # of bits | 16 | 2 | 1 | 6 | 2 | 1 | 2 | 16 | 16 |
| description | 1110001 0 0000000 0 | 00 | 0: Deassert U in XPC_W1 1: Assert U in XPC_W1 | MSB (<u>show/hide</u>): 0: show memory above EPC 1: hide memory above EPC 5 LSBs (<u>length</u>): New EPC length field (new L bits) | 00: hide none 01: hide some 10: hide all 11: RFU | 0: view 1: hide | 00: normal 01: toggle temporarily 10: reduced 11: RFU | <u>handl</u> e | CRC-16 |

6.3.2.12.3.17 FileOpen (optional)

Interrogators and Tags may implement the *FileOpen* command; if they do, they shall implement it as shown in [Table 6-67](#). *FileOpen* allows an Interrogator to instruct a Tag to close the currently open file and open a new file. A Tag only executes a *FileOpen* in the **open** or **secured** state. *FileOpen* has the following fields:

- FileNum specifies the file to be opened.

A *FileOpen* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **open** or **secured** states that receives a *FileOpen* with nonzero RFU bits or that specifies FileNum=111111111₂ (RFU FileNum) shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *FileOpen* command's functionality

An authenticated Interrogator shall encapsulate a *FileOpen* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *FileOpen* from an authenticated Interrogator then it shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An unauthenticated Interrogator may issue a *FileOpen* without encapsulation to open files accessible from a Tag (a) in the **open** state, or (b) in the **secured** state by an Interrogator that supplied the access password.

If an Interrogator or a Tag support File_N, N>0 then that Interrogator or Tag shall implement a *FileOpen*.

If an Interrogator issues a *FileOpen* specifying a File_N, N>0 for which the Interrogator has a 0000₂ file privilege value then the Tag will open the file, but the Interrogator will not be able to read any data in or otherwise modify the file (see [Table 6-24](#) and [Table 6-25](#)).

An Interrogator shall prepend an unencapsulated *FileOpen* with a frame-sync (see [6.3.1.2.8](#)).

If a Tag supports the *FileOpen* command then it shall implement the file (F) indicator (see [6.3.2.1.3](#)).

A Tag shall reply to a *FileOpen* using the *immediate* reply specified in [6.3.1.6.1](#). If the Tag has an allocated file at FileNum then it shall close the currently open file, open the specified file, and reply as shown in [Table 6-68](#). The reply includes a header (a 0-bit), FileNum, FileType, FileSize, BlockSize, IntPriv, LastFile, and the Tag's handle. FileNum, FileType, FileSize, BlockSize are defined in [6.3.2.11.3](#). IntPriv is the Interrogator's 4-bit privilege to the file (see [Table 6-24](#) and [Table 6-25](#)). LastFile indicates whether the just-opened file has the largest assigned FileNum; if a Tag has a FileNum larger than that of the just-opened file then it shall set LastFile to 0, otherwise it shall set LastFile to 1. The reply includes a CRC-16 calculated over the 0-bit to the last handle bit. If a Tag receives a *FileOpen* specifying the currently open file then it shall leave the file open and reply as specified in [Table 6-68](#). If a Tag receives a *FileOpen* but does not have an allocated file at FileNum, or if User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or if the Tag is otherwise unable to execute the *FileOpen*, then the Tag shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see [Table C-30](#)), reverting to the currently open file (or to no file if the Tag doesn't have any allocated files or if User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege).

Table 6-67: *FileOpen* command

| | Command | RFU | FileNum | RN | CRC |
|-------------|----------|-----|--------------------|---------------|--------|
| # of bits | 8 | 2 | 10 | 16 | 16 |
| description | 11010011 | 00 | Which file to open | <u>handle</u> | CRC-16 |

Table 6-68: Tag reply to a successful *FileOpen* command

| | Header | FileNum | FileType | FileSize | BlockSize | IntPriv | LastFile | RN | CRC |
|-------------|--------|-----------|-----------------|---------------------|---------------------|-------------------------------|--|---------------|--------|
| # of bits | 1 | 10 | 8 | 10 | 10 | 4 | 1 | 16 | 16 |
| description | 0 | Open file | <u>FileType</u> | File size in blocks | Block size in words | Interrogator's file privilege | 0: Not max <u>FileNum</u> 1: Max <u>FileNum</u> | <u>handle</u> | CRC-16 |

6.3.2.12.3.18 FileList (optional)

Interrogators and Tags may implement the *FileList* command; if they do, they shall implement it as shown in [Table 6-69](#). *FileList* allows an Interrogator to obtain information about a Tag's files and the

Interrogator’s privileges to those files. A Tag only executes a *FileList* in the **open** or **secured** state. *FileList* has the following fields:

- SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- FileNum identifies the starting file for which the Interrogator is requesting information, inclusive.
- AddFiles identifies the number of additional files for which the Interrogator is requesting information. For example, if FileNum=4 and AddFiles=2 then the Tag shall provide information for File_4 and for the next two higher-numbered files (which may be File_5 and File_6 if the Tag manufacturer assigned file numbers sequentially or may be other files if the numbering is not sequential).

A *FileList* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **open** or **secured** states that receives a *FileList* with nonzero RFU bits or that specifies FileNum=11111111₂ (RFU FileNum) shall not execute the *FileList* and instead treat the command’s parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *FileList* command’s functionality.

An authenticated Interrogator shall encapsulate a *FileList* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *FileList* from an authenticated Interrogator then it shall not execute the *FileList* and instead treat the command’s parameters as unsupported (see [Table C-30](#)).

An unauthenticated Interrogator may issue a *FileList* without encapsulation to a Tag in the **open** or **secured** state.

An Interrogator shall not specify AddFiles=FF_h. If a Tag receives a *FileList* with AddFiles=FF_h then the Tag shall behave as though it had received a *FileList* with AddFiles=FD_h.

An Interrogator shall prepend an unencapsulated *FileList* with a frame-sync (see [6.3.1.2.8](#)).

A Tag shall reply to a *FileList* using the *in-process* reply specified in [6.3.1.6.3](#). A Tag’s response shall be as shown in [Table 6-70](#) and includes a message for each file for which the Interrogator requested information. The response includes the number of messages, the message contents (10-bit FileNum, 8-bit FileType, 10-bit FileSize, 4-bit IntPriv), the BlockSize, and the free memory available for file resizing (AvailFileSize). IntPriv is the Interrogator’s 4-bit privilege to the file (see [Table 6-24](#) and [Table 6-25](#)). If a Tag is *static* then AvailFileSize shall be zero. If a Tag has more than 1022 blocks of free memory then AvailFileSize shall be 11111111₂. If a Tag receives a *FileList* with an unsupported FileNum, or AddFiles exceeds the number of files above FileNum, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the *FileList*, then the Tag shall not execute the *FileList* and instead treat the command’s parameters as unsupported (see [Table C-30](#)).

Table 6-69: *FileList* command

| | Command | RFU | SenRep | IncRepLen | FileNum | AddFiles | RN16 | CRC |
|-------------|----------------------|-----|---------------------|---|----------------------|--|---------------|------------|
| # of bits | 16 | 2 | 1 | 1 | 10 | 8 | 16 | 16 |
| Description | 11100010 00000001 | 00 | 0: store 1: send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | First file number | Total number of additional files | <u>handle</u> | CRC- 16 |

Table 6-70: Tag reply to a successful *FileList* command

| | NumMessages | Message 1 | ... | Message N | BlockSize | AvailFileSize |
|-------------|--|---|-----|---|------------------------|----------------------------------|
| # of bits | 8 | 32 | ... | 32 | 10 | 10 |
| description | Number of messages in this reply | [FileNum, FileType, FileSize, IntPriv] | ... | [FileNum, FileType, FileSize, IntPriv] | Block size in words | Allocateable memory in blocks |

6.3.2.12.3.19 FilePrivilege (optional)

Interrogators and Tags may implement the *FilePrivilege* command; if they do, they shall implement it as shown in [Table 6-72](#). *FilePrivilege* allows an Interrogator to read or modify file privileges ([Table 6-24](#) or [Table 6-25](#)) for the currently open file. A Tag only executes a *FilePrivilege* in the **secured** state. *FilePrivilege* has the following fields:

- **SenRep** specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- **IncRepLen** specifies whether the Tag omits or includes length in its reply. If **IncRepLen**=0 then the Tag omits length from its reply; if **IncRepLen**=1 then the Tag includes length in its reply.
- **Action** specifies whether the Interrogator is reading or modifying a privilege for the currently open file, and if modifying whether the change applies to the **open** state, access password, a single key, or all keys.
- **KeyID** specifies a key.
- **Privilege** specifies the file privilege. See [Table 6-24](#) and [Table 6-25](#).

A *FilePrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **secured** state that receives a *FilePrivilege* with nonzero RFU bits shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *FilePrivilege* command's functionality.

An authenticated Interrogator shall encapsulate a *FilePrivilege* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *FilePrivilege* from an authenticated Interrogator then it shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An unauthenticated Interrogator may issue a *FilePrivilege* to a Tag in the **secured** state without encapsulation.

A Tag shall execute a *TagPrivilege* according to [Table 6-71](#) which specifies, for each **Action** value, the privilege assignment that the Tag makes (if any), the fields in the *FilePrivilege* that the Tag ignores, the required Tag or file privilege to perform the requested operation, and the reply that the Tag backscatters. An Interrogator may set an ignored field in a *FilePrivilege* to any value.

Table 6-71: Action field behavior for a *FilePrivilege*

| Action | Privilege a Tag Assigns to the Currently Open File | Tag Ignores | Required Privilege | Tag Reply to the <i>FilePrivilege</i> | Reference |
|------------------|--|-----------------------------------|--|---|--|
| 000 ₂ | – | <u>KeyID</u> and <u>privilege</u> | Any | <u>FileNum</u> and the open-state file <u>privilege</u> | Table 6-73 , <u>Action</u> =000 ₂ or 001 ₂ |
| 001 ₂ | <u>privilege</u> to the open state for <u>FileNum</u> | <u>KeyID</u> | File superuser | – | |
| 010 ₂ | – | <u>KeyID</u> and <u>privilege</u> | Any | <u>FileNum</u> and the access-password file <u>privilege</u> | Table 6-73 , <u>Action</u> =010 ₂ or 011 ₂ |
| 011 ₂ | <u>privilege</u> to the access password for <u>FileNum</u> | <u>KeyID</u> | File superuser to assign <u>privilege</u> for the access password <u>DecFilePriv</u> to decrement <u>privilege</u> for supplied access password | – | |
| 100 ₂ | – | <u>privilege</u> | Any | <u>FileNum</u> , <u>KeyID</u> , and <u>KeyID</u> 's file <u>privilege</u> | Table 6-73 , <u>Action</u> =100 ₂ or 101 ₂ |
| 101 ₂ | <u>privilege</u> to <u>KeyID</u> for <u>FileNum</u> | – | File superuser to assign <u>privilege</u> for any <u>Key_N</u> <u>DecFilePriv</u> to decrement <u>privilege</u> for supplied key | – | |

| Action | Privilege a Tag Assigns to the Currently Open File | Tag Ignores | Required Privilege | Tag Reply to the FilePrivilege | Reference |
|------------------|--|-----------------------------------|--------------------|--|--|
| 110 ₂ | – | <u>KeyID</u> and <u>privilege</u> | Any | FileNum, NumKeys, and a <u>KeyID/privilege</u> pair for each key | Table 6-73 , <u>Action</u> =110 ₂ or 111 ₂ |
| 111 ₂ | <u>privilege</u> to all <u>KeyID</u> for FileNum | <u>KeyID</u> | File superuser | – | |

As shown in [Table 6-71](#), a Tag permits an Interrogator that is a file superuser to modify a file privilege for the **open** state, access password, or any key regardless of the cryptographic suite to which the key is assigned, for the currently open file. [Table 6-71](#) also shows that a Tag permits an Interrogator that is not a file superuser to decrement the file privilege for the access password or key that it used to most recently enter the **secured** state if the access password or key has an asserted DecFilePriv, but only for the currently open file and not for the **open** state or for any other password or key. Finally, [Table 6-71](#) shows that a Tag permits any Interrogator to read the privileges for the currently open file, for the **open** state, access password, or for any key.

Upon receiving an executable *FilePrivilege* with Action=001₂, 011₂, 101₂, or 111₂ a Tag shall overwrite the current file privilege(s) with the new privilege. If the Tag does not write the new privilege successfully then it shall revert to the prior stored privilege. A Tag may meet this latter requirement by, for example, double-buffering the write operation. Note that, if Action=111₂ then a Tag may complete the write operation for some keys but not for others, in which case an Interrogator can read the privilege values and, if necessary, re-issue a *FilePrivilege*.

An Interrogator shall prepend an unencapsulated *FilePrivilege* with a frame-sync (see [6.3.1.2.8](#)).

A Tag's response to the *FilePrivilege*, for incorporation into the *in-process* reply specified in [6.3.1.6.3](#), shall be as shown in [Table 6-73](#).

If a Tag receives a *FilePrivilege* that it cannot execute because the access password or key the Interrogator supplied has insufficient privileges, or the *FilePrivilege* contains an unsupported KeyID, or privilege is an RFU value, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the *FilePrivilege*, then the Tag shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Table 6-72: *FilePrivilege* command

| | Command | RFU | SenRep | IncRepLen | Action | KeyID | Privilege | RN | CRC |
|-------------|----------------------|-----|-------------------|---|--|--------------|------------------|---------------|--------|
| # of bits | 16 | 2 | 1 | 1 | 3 | 8 | 4 | 16 | 16 |
| description | 11100010 00000100 | 00 | 0:store 1:send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | 000: Read open state 001: Modify open state 010: Read access pwd 011: Modify access pwd 100: Read <u>KeyID</u> 101: Modify <u>KeyID</u> 110: Read all keys 111: Modify all keys | <u>KeyID</u> | <u>privilege</u> | <u>handle</u> | CRC-16 |

Table 6-73: Tag reply to a successful *FilePrivilege* with indicated Action fields Action=000₂ or 001₂

| | FileNum | Privilege |
|-------------|---------------------|-----------------------------|
| # of bits | 10 | 4 |
| description | Currently open file | open state <u>privilege</u> |

Action=010₂ or 011₂

| | FileNum | Privilege |
|-------------|---------------------|----------------------------------|
| # of bits | 10 | 4 |
| description | Currently open file | access password <u>privilege</u> |

Action=100₂ or 101₂

| | FileNum | KeyID | Privilege |
|-------------|---------------------|--------------|------------------------|
| # of bits | 10 | 8 | 4 |
| description | Currently open file | <u>KeyID</u> | <u>KeyID privilege</u> |

Action=110₂ or 111₂

| | FileNum | NumKeys | Key_0 / Privilege_0 pair | ... | Key_N / Privilege_N pair |
|-------------|---------------------|----------------|--------------------------|-----|--------------------------|
| # of bits | 10 | 8 | 12 | ... | 12 |
| description | Currently open file | Number of keys | Key_0 Privilege_0 | ... | Key_N Privilege_N |

6.3.2.12.3.20 FileSetup (optional)

Interrogators and Tags may implement the *FileSetup* command; if they do, they shall implement it as shown in [Table 6-74](#). *FileSetup* allows an Interrogator to resize the currently open file, change its FileType, or both. A Tag only executes a *FileSetup* in the **secured** state. *FileSetup* has the following fields:

- SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.
- IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.
- FileType specifies the new file type.
- FileSize specifies the requested file size in blocks.

A *FileSetup* contains 2 RFU bits. An Interrogator shall set these bits to 00₂. A Tag in the **secured** state that receives a *FileSetup* with nonzero RFU bits shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see [Table C-30](#)). Future protocols may use these RFU bits to expand the *FileSetup* command's functionality.

A Tag shall only execute a *FileSetup* issued by an Interrogator with a file superuser privilege (see [6.3.2.11.3](#)).

An authenticated Interrogator shall encapsulate a *FileSetup* in a *SecureComm* or *AuthComm* (see [Table 6-28](#)). If a Tag in the **secured** state receives an unencapsulated *FileSetup* from an authenticated Interrogator then it shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

An unauthenticated Interrogator may issue a *FileSetup* to a Tag in the **secured** state without encapsulation.

A *static* Tag that supports the *FileSetup* command shall permit an Interrogator with the file superuser privilege to modify a file's type but never its size. A *static* Tag shall write the FileType in a *FileSetup* as the file's new type and shall ignore FileSize. An Interrogator may set FileSize to any value when communicating with a *static* Tag.

A *dynamic* Tag shall permit an Interrogator with the file superuser privilege to modify a file's type and size. [Table 6-26](#) specifies the conditions under which a *dynamic* Tag may be able to resize a file. When increasing a file's size a *dynamic* Tag shall only allocate "free" memory (i.e. memory not currently allocated to another file) to the resized file. When reducing a file's size a *dynamic* Tag may or may not, depending on the Tag manufacturer's implementation, erase the excised memory. Consequently, this protocol recommends that Interrogators, before reducing a file's size, erase that portion of the file that will be excised by the resizing. Whether a *dynamic* Tag is able to recover memory freed by resizing a file's size downward depends on the Tag manufacturer's implementation and is not specified by this protocol.

Regardless of whether a Tag is *static* or *dynamic*, after executing a *FileSetup* a Tag's response shall include both FileType and FileSize (even if the Tag made no changes to either one). See [Table 6-75](#).

An Interrogator shall prepend an unencapsulated *FileSetup* with a frame-sync (see [6.3.1.2.8](#)).

A Tag's response to the *FileSetup*, for incorporation into the *in-process* reply specified in [6.3.1.6.3](#), shall be as shown in [Table 6-75](#). The response includes the FileNum, FileType, and FileSize. If a Tag receives a *FileSetup* that it cannot execute because the access password or key that the Interrogator most recently supplied does not have a file superuser privilege, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the *FileSetup*, then the Tag shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see [Table C-30](#)).

There are many reasons why a *dynamic* Tag may be unable to execute a *FileSetup* including (a) the Tag does not have free memory to increase the file size, (b) the Tag has free memory but is unable to allocate it to the file, (c) the file has a permalocked block, and (d) many others. If a *dynamic* Tag is unable to execute the FileSize in the *FileSetup* command then it shall not execute any portion of the *FileSetup* (i.e. it shall not change the FileType) and instead treat the command's parameters as unsupported (see [Table C-30](#)).

Table 6-74: *FileSetup* command

| | Command | RFU | SenRep | IncRepLen | FileType | FileSize | RN16 | CRC |
|-------------|----------------------|-----|---------------------|---|-----------------|--------------------------------------|---------------|------------|
| # of bits | 16 | 2 | 1 | 1 | 8 | 10 | 16 | 16 |
| description | 11100010 00000101 | 00 | 0: store 1: send | 0: Omit <u>length</u> from reply 1: Include <u>length</u> in reply | <u>FileType</u> | Requested file size, in blocks | <u>handle</u> | CRC- 16 |

Table 6-75: Tag reply to a successful *FileSetup* command¹

| | FileNum | FileType | FileSize |
|-------------|------------------------|-----------------|---------------------------------|
| # of bits | 10 | 8 | 10 |
| description | Currently open file | <u>FileType</u> | Current file size, in blocks |

¹ see also [Table 6-26](#)



7. Intellectual property rights policy and disclaimer

GS1, under its IP Policy, seeks to avoid uncertainty regarding intellectual property claims by requiring the participants in the Work Group that developed this Generation-2 UHF RFID V2.0.0 Protocol to agree to grant to GS1 members a royalty-free license or a RAND license to Necessary Claims, as that term is defined in the GS1/EPCglobal IP Policy. Furthermore, attention is drawn to the possibility that an implementation of one or more features of this Protocol may be the subject of a patent or other intellectual property right that does not involve a Necessary Claim. Any such patent or other intellectual property right is not subject to the licensing obligations of GS1. Moreover, the agreement to grant licenses provided under the GS1/EPCglobal IP Policy does not include IP rights and any claims of third parties who were not participants in the Work Group.

Accordingly, GS1 recommends that any organization developing an implementation designed to be in conformance with this Protocol should determine whether there are any patents that may encompass a specific implementation that the organization is developing in compliance with the Protocol and whether a license under any patent rights identified herein, or any patent rights of third parties, or any other intellectual property right is needed. Such a determination of a need for licensing should be made in view of the details of the specific system designed by the organization in consultation with their own patent counsel.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF THIS PROTOCOL. GS1 disclaims all liability for any damages arising from use or misuse of this Protocol, whether special, indirect, consequential, or compensatory damages, and including liability for infringement of any intellectual property rights, relating to use of information in or reliance upon this document.

GS1 retains the right to make changes to this document at any time, without notice. GS1 makes no warranty for the use of this document and assumes no responsibility for any errors which may appear in the document, nor does it make a commitment to update the information contained herein.

It is noted that the owners of the patents and patent applications listed below gave notice pursuant to Section 3.3 of the GS1 IP Policy that they do not intend to grant royalty-free licenses under Section 3.1 of the IP Policy. The owners have indicated a willingness to grant royalty-bearing licenses on RAND terms in connection with any Necessary Claims. In view of the number of declared claims, the inevitable differences of opinion on whether a particular declared claim is a Necessary Claim, as that term is defined in the GS1/EPCglobal IP Policy, whether the Protocol provides means for avoiding any declared patent claims, and whether any of the declared claims are subject to royalty-free licensing obligations resulting from prior involvement in the Working Group, GS1 has decided to provide this notice and disclaimer in the Protocol.

| U.S. Patent No. | U.S. Patent Application No. |
|-----------------|-----------------------------|
| 7,145,482 | 13/481,141 |
| 8,188,839 | 10/597,725 |
| 6,784,787 | |
| 6,480,143 | |
| 5,680,459 | |
| 5,519,381 | |
| 5,726,630 | |
| 7,987,405 | |

Annex A (normative) Extensible bit vectors (EBV)

An *extensible bit vector* (EBV) is a data structure with an extensible data range.

An EBV is an array of blocks. Each block contains a single extension bit followed by a specific number of data bits. If B represents the total number of bits in one block, then a block contains B – 1 data bits. Although a general EBV may contain blocks of varying lengths, Tags and Interrogators manufactured according to this protocol shall use blocks of length 8 bits (EBV-8).

The data value represented by an EBV is simply the bit string formed by the data bits as read from left-to-right, ignoring the extension bits.

Tags and Interrogators shall use the EBV-8 word format specified in [Table A-1](#).

Table A-1: EBV-8 word format

| | | | | | | |
|--------------|-------|-------------|-------------|---|-------------|------------------|
| 0 | 0 | 000000 0 | | | | |
| 1 | 0 | 000000 1 | | | | |
| $2^7 - 1$ | 127 | 0 | 111111 1 | | | |
| 2^7 | 128 | 1 | 000000 1 | 0 | 000000 0 | |
| $2^{14} - 1$ | 16383 | 1 | 111111 1 | 0 | 111111 1 | |
| 2^{14} | 16384 | 1 | 000000 1 | 1 | 000000 0 | 0 000000 0 |

Because each block has 7 available data bits, an EBV-8 can represent numeric values between 0 and 127 with a single block. To represent the value 128, set the extension bit to 1 in the first block, and append a second block to the EBV-8. In this manner, an EBV-8 can represent arbitrarily large data values. This protocol uses EBV-8 values to represent memory addresses and mask lengths.

Annex B (normative) State-transition tables

State-transition tables [B.1](#) to [B.7](#) shall define a Tag's response to Interrogator commands. The term "handle" used in the state-transition tables is defined in [6.3.2.6.5](#); error codes are defined in [Table I-2](#); "slot" is the slot-counter output shown in [Figure 6-21](#) and detailed in [Annex J](#); and "-" in the "Action" column means that a Tag neither executes the command nor backscatters a reply.

B.1 Present state: Ready

Table B-1: Ready state-transition table

| Command | Condition | Action | Next State |
|---------------------------|---|---|------------|
| <i>Query</i> ¹ | slot=0; matching inventoried & SL flags | backscatter new RN16 | reply |
| | slot<>0; matching inventoried & SL flags | – | arbitrate |
| | otherwise | – | ready |
| <i>QueryRep</i> | all | – | ready |
| <i>QueryAdjust</i> | all | – | ready |
| <i>ACK</i> | all | – | ready |
| <i>NAK</i> | all | – | ready |
| <i>Req_RN</i> | all | – | ready |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | ready |
| <i>Read</i> | all | – | ready |
| <i>Write</i> | all | – | ready |
| <i>Kill</i> | all | – | ready |
| <i>Lock</i> | all | – | ready |
| <i>Access</i> | all | – | ready |
| <i>BlockWrite</i> | all | – | ready |
| <i>BlockErase</i> | all | – | ready |
| <i>BlockPermalock</i> | all | – | ready |
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> & executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | all | – | ready |
| <i>AuthComm</i> | all | – | ready |
| <i>SecureComm</i> | all | – | ready |
| <i>ReadBuffer</i> | all | – | ready |
| <i>KeyUpdate</i> | all | – | ready |
| <i>Untraceable</i> | all | – | ready |
| <i>FileSetup</i> | all | – | ready |
| <i>FileOpen</i> | all | – | ready |

| Command | Condition | Action | Next State |
|----------------------|----------------------|--------|------------|
| <i>FilePrivilege</i> | all | – | ready |
| <i>TagPrivilege</i> | all | – | ready |
| <i>FileList</i> | all | – | ready |
| Faulty | invalid ² | – | ready |

1: *Query* starts a new round and may change the session. *Query* also instructs a Tag to load a new random value into its slot counter.

2: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.2 Present state: Arbitrate

Table B-2: Arbitrate state-transition table

| Command | Condition | Action | Next State |
|---------------------------------|---|---|------------|
| <i>Query</i> ^{1,2} | slot=0; matching inventoried & SL flags | backscatter new RN16 | reply |
| | slot<>0; matching inventoried & SL flags | – | arbitrate |
| | otherwise | – | ready |
| <i>QueryRep</i> | <u>session</u> matches inventory round & slot=0 after decrementing slot counter | decrement slot counter; backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot<>0 after decrementing slot counter | decrement slot counter | arbitrate |
| | <u>session</u> does not match inventory round | – | arbitrate |
| <i>QueryAdjust</i> ² | <u>session</u> matches inventory round & slot=0 | backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot<>0 | – | arbitrate |
| | <u>session</u> does not match inventory round | – | arbitrate |
| <i>ACK</i> | all | – | arbitrate |
| <i>NAK</i> | all | – | arbitrate |
| <i>Req_RN</i> | all | – | arbitrate |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | arbitrate |
| <i>Read</i> | all | – | arbitrate |
| <i>Write</i> | all | – | arbitrate |
| <i>Kill</i> | all | – | arbitrate |
| <i>Lock</i> | all | – | arbitrate |
| <i>Access</i> | all | – | arbitrate |
| <i>BlockWrite</i> | all | – | arbitrate |
| <i>BlockErase</i> | all | – | arbitrate |
| <i>BlockPermalock</i> | all | – | arbitrate |
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> & executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | all | – | arbitrate |
| <i>AuthComm</i> | all | – | arbitrate |
| <i>SecureComm</i> | all | – | arbitrate |
| <i>ReadBuffer</i> | all | – | arbitrate |
| <i>KeyUpdate</i> | all | – | arbitrate |
| <i>Untraceable</i> | all | – | arbitrate |



| Command | Condition | Action | Next State |
|----------------------|----------------------|--------|------------|
| <i>FileSetup</i> | all | – | arbitrate |
| <i>FileOpen</i> | all | – | arbitrate |
| <i>FilePrivilege</i> | all | – | arbitrate |
| <i>TagPrivilege</i> | all | – | arbitrate |
| <i>FileList</i> | all | – | arbitrate |
| Faulty | invalid ³ | – | arbitrate |

1: *Query* starts a new round and may change the session.

2: *Query* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.3 Present state: Reply

Table B-3: Reply state-transition table

| Command | Condition | Action | Next State |
|---------------------------------|---|---|--------------|
| <i>Query</i> ^{1,2} | slot=0; matching inventoried & SL flags | backscatter new RN16 | reply |
| | slot<>0; matching inventoried & SL flags | – | arbitrate |
| | otherwise | – | ready |
| <i>QueryRep</i> | <u>session</u> matches inventory round | – | arbitrate |
| | <u>session</u> does not match inventory round | – | reply |
| <i>QueryAdjust</i> ² | <u>session</u> matches inventory round & slot=0 | backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot<>0 | – | arbitrate |
| | <u>session</u> does not match inventory round | – | reply |
| <i>ACK</i> | correct RN16 | See Table 6-17 | acknowledged |
| | incorrect RN16 | – | arbitrate |
| <i>NAK</i> | all | – | arbitrate |
| <i>Req_RN</i> | all | – | arbitrate |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to A or B | ready |
| | incorrect parameters | – | reply |
| <i>Read</i> | all | – | arbitrate |
| <i>Write</i> | all | – | arbitrate |
| <i>Kill</i> | all | – | arbitrate |
| <i>Lock</i> | all | – | arbitrate |
| <i>Access</i> | all | – | arbitrate |
| <i>BlockWrite</i> | all | – | arbitrate |
| <i>BlockErase</i> | all | – | arbitrate |
| <i>BlockPermalock</i> | all | – | arbitrate |
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> & executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | all | – | arbitrate |
| <i>AuthComm</i> | all | – | arbitrate |
| <i>SecureComm</i> | all | – | arbitrate |
| <i>ReadBuffer</i> | all | – | arbitrate |
| <i>KeyUpdate</i> | all | – | arbitrate |
| <i>Untraceable</i> | all | – | arbitrate |
| <i>FileSetup</i> | all | – | arbitrate |
| <i>FileOpen</i> | all | – | arbitrate |

| Command | Condition | Action | Next State |
|------------------------|--|--------|------------|
| <i>FilePrivilege</i> | all | – | arbitrate |
| <i>TagPrivilege</i> | all | – | arbitrate |
| <i>FileList</i> | all | – | arbitrate |
| T ₂ timeout | See Figure 6-18 and Table 6-16 | – | arbitrate |
| Faulty | invalid ³ | – | reply |

1: *Query* starts a new round and may change the session.

2: *Query* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.4 Present state: Acknowledged

Table B-4: Acknowledged state-transition table

| Command | Condition | Action | Next State |
|---------------------------|---|--|--------------|
| <i>Query</i> ¹ | slot=0; matching inventoried ² & SL flags | backscatter new RN16; transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | reply |
| | slot<>0; matching inventoried ² & SL flags | transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | arbitrate |
| | otherwise | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | ready |
| <i>QueryRep</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | acknowledged |
| <i>QueryAdjust</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | acknowledged |
| <i>ACK</i> | correct RN16 | See Table 6-17 | acknowledged |
| | incorrect RN16 | – | arbitrate |
| <i>NAK</i> | all | – | arbitrate |
| <i>Req_RN</i> | correct RN16 & access password<>0 | backscatter <u>handle</u> | open |
| | correct RN16 & access password=0 | backscatter <u>handle</u> | secured |
| | incorrect RN16 | – | acknowledged |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | acknowledged |
| <i>Read</i> | all | – | arbitrate |
| <i>Write</i> | all | – | arbitrate |
| <i>Kill</i> | all | – | arbitrate |
| <i>Lock</i> | all | – | arbitrate |
| <i>Access</i> | all | – | arbitrate |
| <i>BlockWrite</i> | all | – | arbitrate |
| <i>BlockErase</i> | all | – | arbitrate |
| <i>BlockPermalock</i> | all | – | arbitrate |
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> & executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | all | – | arbitrate |
| <i>AuthComm</i> | all | – | arbitrate |
| <i>SecureComm</i> | all | – | arbitrate |
| <i>ReadBuffer</i> | all | – | arbitrate |
| <i>KeyUpdate</i> | all | – | arbitrate |
| <i>Untraceable</i> | all | – | arbitrate |



| Command | Condition | Action | Next State |
|------------------------|--|--------|--------------|
| <i>FileSetup</i> | all | – | arbitrate |
| <i>FileOpen</i> | all | – | arbitrate |
| <i>FilePrivilege</i> | all | – | arbitrate |
| <i>TagPrivilege</i> | all | – | arbitrate |
| <i>FileList</i> | all | – | arbitrate |
| T ₂ timeout | See Figure 6-18 and Table 6-16 | – | arbitrate |
| Faulty | invalid ³ | – | acknowledged |

1: *Query* starts a new round and may change the session. *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in [6.3.2.10](#), a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.5 Present state: Open

Table B-5: Open state-transition table

| Command | Condition | Action | Next State |
|--|---|--|------------|
| <i>Query</i> ¹ | slot=0; matching inventoried ² & SL flags | backscatter new RN16; transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | reply |
| | slot<>0; matching inventoried ² & SL flags | transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | arbitrate |
| | otherwise | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | ready |
| <i>QueryRep</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | open |
| <i>QueryAdjust</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | open |
| <i>ACK</i> | correct <u>handle</u> | See Table 6-17 | open |
| | incorrect <u>handle</u> | – | arbitrate |
| <i>NAK</i> | all | – | arbitrate |
| <i>Req_RN</i> | all | backscatter new RN16 | open |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | open |
| <i>Read</i> | all | backscatter data | open |
| <i>Write</i> | all | backscatter <u>header</u> when done | open |
| <i>Kill</i> (see also Figure 6-24) | supported security timeout | backscatter error code | open |
| | password-based kill & correct nonzero kill password | backscatter <u>header</u> when done | killed |
| | password-based kill & incorrect nonzero kill password | may set security timeout | arbitrate |
| | password-based kill & kill password=0 | backscatter error code | open |
| | authenticated kill | backscatter error code; may set security timeout | arbitrate |
| <i>Lock</i> | all | – | open |
| <i>Access</i> (see also Figure 6-26) | supported security timeout | backscatter error code | open |
| | correct access password | backscatter <u>handle</u> | secured |
| | incorrect access password | may set security timeout | arbitrate |
| <i>BlockWrite</i> | all | backscatter <u>header</u> when done | open |
| <i>BlockErase</i> | all | backscatter <u>header</u> when done | open |
| <i>BlockPermalock</i> | all | – | open |
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> and executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | supported security timeout | backscatter error code | open |

| Command | Condition | Action | Next State |
|----------------------|--|--|------------------------------|
| | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | open or secured ³ |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | open or secured ³ |
| | crypto error | see crypto suite | arbitrate |
| | new authentication | reset crypto engine | open |
| <i>AuthComm</i> | supported security timeout | backscatter error code | open |
| | prior Tag authentication & executable | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Tag authentication | backscatter error code | open |
| | crypto error | see crypto suite | arbitrate |
| <i>SecureComm</i> | supported security timeout | backscatter error code | open |
| | prior Tag authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | see encapsulated command |
| | prior Tag authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Tag authentication | backscatter error code | open |
| | crypto error | see crypto suite | arbitrate |
| <i>ReadBuffer</i> | C=1 | backscatter data | open |
| | C=0 | backscatter error code | open |
| <i>KeyUpdate</i> | all | – | open |
| <i>Untraceable</i> | all | – | open |
| <i>FileSetup</i> | all | – | open |
| <i>FileOpen</i> | executable | close current file; open requested file; backscatter file info | open |
| <i>FilePrivilege</i> | all | – | open |
| <i>TagPrivilege</i> | all | – | open |
| <i>FileList</i> | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | open |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | open |
| Faulty | unsupported parameters ⁴ | backscatter error code | open |
| | incorrect <u>handle</u> ⁵ | none unless specified by crypto suite | open |
| | improper ⁶ | – | arbitrate |
| | invalid ⁷ | none unless specified by crypto suite | open |

1: *Query* starts a new round and may change the session. *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in [6.3.2.10](#), a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: See cryptographic suite

4: “Unsupported parameters” shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) a command that requires encapsulation but is nonetheless unencapsulated (see [Table 6-28](#)), (8) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (9) an operation for which the Interrogator has insufficient privileges, (10) an unsupported cryptographic parameter, or (11) other parameters not supported by the Tag.

- 5: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving a security command with an incorrect handle.
- 6: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see [Figure 6-24](#) and [Figure 6-26](#)).
- 7: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite/BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving an invalid command.

B.6 Present state: Secured

Table B-6: Secured state-transition table

| Command | Condition | Action | Next State |
|--|---|--|------------|
| <i>Query</i> ¹ | slot=0; matching inventoried ² & SL flags | backscatter new RN16; transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | reply |
| | slot<>0; matching inventoried ² & SL flags | transition inventoried ² from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | arbitrate |
| | otherwise | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> if and only if new <u>session</u> matches prior <u>session</u> | ready |
| <i>QueryRep</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | secured |
| <i>QueryAdjust</i> | <u>session</u> matches inventory round | transition inventoried from <i>A</i> → <i>B</i> or <i>B</i> → <i>A</i> | ready |
| | <u>session</u> does not match inventory round | – | secured |
| <i>ACK</i> | correct <u>handle</u> | See Table 6-17 | secured |
| | incorrect <u>handle</u> | – | arbitrate |
| <i>NAK</i> | all | – | arbitrate |
| <i>Req_RN</i> | all | backscatter new RN16 | secured |
| <i>Select</i> | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | secured |
| <i>Read</i> | all | backscatter data | secured |
| <i>Write</i> | all | backscatter <u>header</u> when done | secured |
| <i>Kill</i> (see also Figure 6-24) | supported security timeout | backscatter error code | secured |
| | password-based kill & correct nonzero kill password | backscatter <u>header</u> when done | killed |
| | password-based kill & incorrect nonzero kill password | may set security timeout | arbitrate |
| | password-based kill & kill password=0 | backscatter error code | secured |
| | authenticated kill with prior Interrogator authentication & <u>AuthKill</u> privilege | backscatter <u>response</u> when done | killed |
| | authenticated kill but no prior Interrogator authentication or no <u>AuthKill</u> privilege | backscatter error code; may set security timeout | arbitrate |
| <i>Lock</i> | all | backscatter <u>header</u> when done | secured |
| <i>Access</i> (see also Figure 6-26) | supported security timeout | backscatter error code | secured |
| | correct access password | backscatter <u>handle</u> | secured |
| | incorrect access password | may set security timeout | arbitrate |
| <i>BlockWrite</i> | all | backscatter <u>header</u> when done | secured |
| <i>BlockErase</i> | all | backscatter <u>header</u> when done | secured |
| <i>BlockPermalock</i> | <u>Read/Lock</u> =0 | backscatter permalock bits | secured |
| | <u>Read/Lock</u> =1 | backscatter <u>header</u> when done | secured |



| Command | Condition | Action | Next State |
|----------------------|---|--|--------------------------|
| <i>Challenge</i> | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> and executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| <i>Authenticate</i> | supported security timeout | backscatter error code | secured |
| | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| | crypto error | see crypto suite | arbitrate |
| | new authentication | reset crypto engine | open |
| <i>AuthComm</i> | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication & executable | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| <i>SecureComm</i> | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | see encapsulated command |
| | prior Interrogator authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| <i>ReadBuffer</i> | C=1 | backscatter data | secured |
| | C=0 | backscatter error code | secured |
| <i>KeyUpdate</i> | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| <i>Untraceable</i> | executable | backscatter <u>header</u> when done | secured |
| <i>FileSetup</i> | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| <i>FileOpen</i> | executable | close current file; open requested file; backscatter file info | secured |
| <i>FilePrivilege</i> | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| <i>TagPrivilege</i> | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |

| Command | Condition | Action | Next State |
|----------|--------------------------------------|--|------------------------------|
| FileList | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| Faulty | unsupported parameters ³ | backscatter error code | secured |
| | incorrect <u>handle</u> ⁴ | none unless specified by crypto suite | secured or open ⁴ |
| | improper ⁵ | – | arbitrate |
| | invalid ⁶ | none unless specified by crypto suite | secured or open ⁶ |

1: *Query* starts a new round and may change the session. *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in [6.3.2.10](#), a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: “Unsupported parameters” shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, lock payload, blockpermalock payload, KeyID, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) insufficient or unallocateable memory, (8) an unencrypted message that requires encryption, (9) a command that requires encapsulation but is nonetheless unencapsulated (see [Table 6-28](#)), (10) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (11) an RFU privilege value, (12) an operation for which the Interrogator has insufficient privileges, (13) an unsupported cryptographic parameter, or (14) other parameters not supported by the Tag.

4: “Incorrect handle” shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its crypto engine and transition to the **open** state upon receiving a security command with an incorrect handle.

5: “Improper” shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see [Figure 6-24](#) and [Figure 6-26](#)).

6: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite/BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine and transition to the **open** state upon receiving an invalid command.

B.7 Present state: Killed

Table B-7: Killed state-transition table

| Command | Condition | Action | Next State |
|-----------------------|-----------|--------|------------|
| <i>Query</i> | all | – | killed |
| <i>QueryRep</i> | all | – | killed |
| <i>QueryAdjust</i> | all | – | killed |
| <i>ACK</i> | all | – | killed |
| <i>NAK</i> | all | – | killed |
| <i>Req_RN</i> | all | – | killed |
| <i>Select</i> | all | – | killed |
| <i>Read</i> | all | – | killed |
| <i>Write</i> | all | – | killed |
| <i>Kill</i> | all | – | killed |
| <i>Lock</i> | all | – | killed |
| <i>Access</i> | all | – | killed |
| <i>BlockWrite</i> | all | – | killed |
| <i>BlockErase</i> | all | – | killed |
| <i>BlockPermalock</i> | all | – | killed |
| <i>Challenge</i> | all | – | killed |
| <i>Authenticate</i> | all | – | killed |
| <i>AuthComm</i> | all | – | killed |
| <i>SecureComm</i> | all | – | killed |
| <i>ReadBuffer</i> | all | – | killed |
| <i>KeyUpdate</i> | all | – | killed |
| <i>Untraceable</i> | all | – | killed |
| <i>FileSetup</i> | all | – | killed |
| <i>FileOpen</i> | all | – | killed |
| <i>FilePrivilege</i> | all | – | killed |
| <i>TagPrivilege</i> | all | – | killed |
| <i>FileList</i> | all | – | killed |
| Faulty | all | – | killed |

Annex C (normative) Command-Response Tables

Command-response tables C.1 to C.30 shall define a Tag's response to Interrogator commands. The term "handle" used in the command-response tables is defined in [6.3.2.6.5](#); error codes are defined in [Table I-2](#); "slot" is the slot-counter output shown in [Figure 6-21](#) and detailed in [Annex J](#); "-" in the "Response" column means that a Tag neither executes the command nor backscatters a reply.

C.1 Command response: Power-up

Table C-1: Power-up command-response table

| Starting State | Condition | Response | Next State |
|--|-----------|----------|------------|
| ready, arbitrate, reply, acknowledged, open, secured | power-up | – | ready |
| killed | all | – | killed |

C.2 Command response: Query

Table C-2: Query¹ command-response table

| Starting State | Condition | Response | Next State |
|-----------------------------|---|--|------------|
| ready, arbitrate, reply | slot=0; matching inventoried & SL flags | backscatter new RN16 | reply |
| | slot<>0; matching inventoried & SL flags | – | arbitrate |
| | otherwise | – | ready |
| acknowledged, open, secured | slot=0; matching inventoried ² & SL flags | backscatter new RN16; transition inventoried ² from A→B or B→A if and only if new <u>session</u> matches prior <u>session</u> | reply |
| | slot<>0; matching inventoried ² & SL flags | transition inventoried ² from A→B or B→A if and only if new <u>session</u> matches prior <u>session</u> | arbitrate |
| | otherwise | transition inventoried from A→B or B→A if and only if new <u>session</u> matches prior <u>session</u> | ready |
| killed | all | – | killed |

1: *Query* (in any state other than **killed**) starts a new round and may change the session; *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in [6.3.2.10](#), a Tag transitions its **inventoried** flag prior to evaluating the condition.

C.3 Command response: *QueryRep*

Table C-3: *QueryRep* command-response table

| Starting State | Condition | Response | Next State |
|----------------|--|--|--------------|
| ready | all | – | ready |
| arbitrate | <u>session</u> matches inventory round & slot=0 after decrementing slot counter | decrement slot counter; backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot < >0 after decrementing slot counter | decrement slot counter | arbitrate |
| | <u>session</u> does not match inventory round | – | arbitrate |
| reply | <u>session</u> matches inventory round | – | arbitrate |
| | <u>session</u> does not match inventory round | – | reply |
| acknowledged | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | acknowledged |
| open | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | open |
| secured | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | secured |
| killed | all | – | killed |

C.4 Command response: *QueryAdjust*

Table C-4: *QueryAdjust*¹ command-response table

| Starting State | Condition | Response | Next State |
|----------------|--|--|--------------|
| ready | all | – | ready |
| arbitrate | <u>session</u> matches inventory round & slot=0 | backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot < >0 | – | arbitrate |
| | <u>session</u> does not match inventory round | – | arbitrate |
| reply | <u>session</u> matches inventory round & slot=0 | backscatter new RN16 | reply |
| | <u>session</u> matches inventory round & slot < >0 | – | arbitrate |
| | <u>session</u> does not match inventory round | – | reply |
| acknowledged | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | acknowledged |
| open | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | open |
| secured | <u>session</u> matches inventory round | transition inventoried from $A \rightarrow B$ or $B \rightarrow A$ | ready |
| | <u>session</u> does not match inventory round | – | secured |
| killed | all | – | ready |

1: *QueryAdjust*, in the **arbitrate** or **reply** states, instructs a Tag to load a new random value into its slot counter.

C.5 Command response: *ACK*

Table C-5: *ACK* command-response table

| Starting State | Condition | Response | Next State |
|------------------------|-------------------------|--------------------------------|--------------|
| ready | all | – | ready |
| arbitrate | all | – | arbitrate |
| reply, acknowledged | correct RN16 | see Table 6-17 | acknowledged |
| | incorrect RN16 | – | arbitrate |
| open | correct <u>handle</u> | see Table 6-17 | open |
| | incorrect <u>handle</u> | – | arbitrate |
| secured | correct <u>handle</u> | see Table 6-17 | secured |
| | incorrect <u>handle</u> | – | arbitrate |
| killed | all | – | killed |

C.6 Command response: *NAK*

Table C-6: *NAK* command-response table

| Starting State | Condition | Response | Next State |
|--|-----------|----------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged, open, secured | all | – | arbitrate |
| killed | all | – | killed |

C.7 Command response: *Req_RN*

Table C-7: *Req_RN* command-response table

| Starting State | Condition | Response | Next State |
|----------------------|--------------------------------------|---------------------------|--------------|
| ready | all | – | ready |
| arbitrate, reply | all | – | arbitrate |
| acknowledged | correct RN16 & access password < > 0 | backscatter <u>handle</u> | open |
| | correct RN16 & access password = 0 | backscatter <u>handle</u> | secured |
| | incorrect RN16 | – | acknowledged |
| open ¹ | all | backscatter new RN16 | open |
| secured ¹ | all | backscatter new RN16 | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle.

C.8 Command response: *Select*

Table C-8: *Select* command-response table

| Starting State | Condition | Response | Next State |
|----------------|----------------------|---|--------------|
| ready | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | ready |
| arbitrate | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | arbitrate |
| reply | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | reply |
| acknowledged | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | acknowledged |
| open | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | open |
| secured | correct parameters | assert or deassert SL, or set inventoried to <i>A</i> or <i>B</i> | ready |
| | incorrect parameters | – | secured |
| killed | all | – | killed |

C.9 Command response: *Read*

Table C-9: *Read* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | all | backscatter data | open |
| secured ¹ | all | backscatter data | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.10 Command response: *Write*

Table C-10: *Write* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | all | backscatter <u>header</u> when done | open |
| secured ¹ | all | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.11 Command response: *Kill*

Table C-11: *Kill*¹ command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|---|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ² | supported security timeout | backscatter error code | open |
| | password-based kill & correct nonzero kill password | backscatter <u>header</u> when done | killed |
| | password-based kill & incorrect nonzero kill password | may set security timeout | arbitrate |
| | password-based kill & kill password=0 | backscatter error code | open |
| | authenticated kill | backscatter error code; may set security timeout | arbitrate |
| secured ² | supported security timeout | backscatter error code | secured |
| | password-based kill & correct nonzero kill password | backscatter <u>header</u> when done | killed |
| | password-based kill & incorrect nonzero kill password | may set security timeout | arbitrate |
| | password-based kill & kill password=0 | backscatter error code | secured |
| | authenticated kill with prior Interrogator authentication & <u>AuthKill</u> privilege | backscatter <u>response</u> when done | killed |
| | authenticated kill but no prior Interrogator authentication or no <u>AuthKill</u> privilege | backscatter error code; may set security timeout | arbitrate |
| killed | all | – | killed |

1: See also [Figure 6-24](#).

2: See [Table C-30](#) for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.12 Command response: *Lock*

Table C-12: *Lock* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | all | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.13 Command response: *Access*

Table C-13: *Access*¹ command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|----------------------------|---------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ² | supported security timeout | backscatter error code | open |
| | correct access password | backscatter <u>handle</u> | secured |
| | incorrect access password | may set security timeout | arbitrate |
| secured ² | supported security timeout | backscatter error code | secured |
| | correct access password | backscatter <u>handle</u> | secured |
| | incorrect access password | may set security timeout | arbitrate |
| killed | all | – | killed |

1: See also [Figure 6-26](#).

2: See [Table C-30](#) for the Tag response to an incorrect handle or an improper command.

C.14 Command response: *BlockWrite*

Table C-14: *BlockWrite* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | all | backscatter <u>header</u> when done | open |
| secured ¹ | all | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.15 Command response: *BlockErase*

Table C-15: *BlockErase* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | all | backscatter <u>header</u> when done | open |
| secured ¹ | all | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.16 Command response: *BlockPermalock*

Table C-16: *BlockPermalock* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|---------------------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | <u>Read/Lock</u> =0 | backscatter permalock bits | secured |
| | <u>Read/Lock</u> =1 | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.17 Command response: *Challenge*

Table C-17: *Challenge* command-response table

| Starting State | Condition | Response | Next State |
|--|---|-------------------------------|------------|
| ready, arbitrate, reply, acknowledged, open, secured | supported security timeout, unsupported <u>CSI</u> , not-executable <u>message</u> , nonzero RFU bits | set C=0 | ready |
| | supported <u>CSI</u> and executable <u>message</u> | store <u>result</u> , set C=1 | ready |
| killed | all | – | killed |

C.18 Command response: *Authenticate*

Table C-18: *Authenticate* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-------------------------------|--|------------------------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | supported security timeout | backscatter error code | open |
| | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | open or secured ² |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | open or secured ² |
| | crypto error | see crypto suite | arbitrate |
| | new authentication | reset crypto engine | open |
| secured ¹ | supported security timeout | backscatter error code | secured |
| | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| | crypto error | see crypto suite | arbitrate |
| | new authentication | reset crypto engine | open |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

2: See cryptographic suite.

C.19 Command response: *AuthComm*

Table C-19: *AuthComm* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|--|---------------------------------------|--------------------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | supported security timeout | backscatter error code | open |
| | prior Tag authentication & executable | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Tag authentication | backscatter error code | open |
| | crypto error | see crypto suite | arbitrate |
| secured ¹ | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication & executable | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.20 Command response: *SecureComm*

Table C-20: *SecureComm* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|---|--|--------------------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | supported security timeout | backscatter error code | open |
| | prior Tag authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | see encapsulated command |
| | prior Tag authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Tag authentication | backscatter error code | open |
| | crypto error | see crypto suite | arbitrate |
| secured ¹ | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | see encapsulated command |
| | prior Interrogator authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | see encapsulated command |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.21 Command response: *ReadBuffer*

Table C-21: *ReadBuffer* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-----------|------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | C=1 | backscatter data | open |
| | C=0 | backscatter error code | open |
| secured ¹ | C=1 | backscatter data | secured |
| | C=0 | backscatter error code | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.22 Command response: *KeyUpdate*

Table C-22: *KeyUpdate* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|---|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | supported security timeout | backscatter error code | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | prior Interrogator authentication, executable, & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| | no prior Interrogator authentication | backscatter error code | secured |
| | crypto error | see crypto suite | arbitrate |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.23 Command response: *Untraceable*

Table C-23: *Untraceable* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|------------|-------------------------------------|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | executable | backscatter <u>header</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.24 Command response: *FileSetup*

Table C-24: *FileSetup* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-------------------------------|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.25 Command response: *FileOpen*

Table C-25: *FileOpen* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|------------|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | executable | close current file; open requested file; backscatter file info | open |
| secured ¹ | executable | close current file; open requested file; backscatter file info | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.26 Command response: *FilePrivilege*

Table C-26: *FilePrivilege* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-------------------------------|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.27 Command response: *TagPrivilege*

Table C-27: *TagPrivilege* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-------------------------------|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open | all | – | open |
| secured ¹ | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.28 Command response: *FileList*

Table C-28: *FileList* command-response table

| Starting State | Condition | Response | Next State |
|--------------------------------|-------------------------------|--|------------|
| ready | all | – | ready |
| arbitrate, reply, acknowledged | all | – | arbitrate |
| open ¹ | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | open |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | open |
| secured ¹ | executable & <u>senrep</u> =0 | store <u>result</u> ; set C=1, backscatter <u>response</u> when done | secured |
| | executable & <u>senrep</u> =1 | backscatter <u>response</u> when done | secured |
| killed | all | – | killed |

1: See [Table C-30](#) for the Tag response to an incorrect handle or unsupported parameters.

C.29 Command response: T_2 timeout

Table C-29: T_2 timeout command-response table

| Starting State | Condition | Response | Next State |
|---------------------|--|----------|------------|
| ready | all | – | ready |
| arbitrate | all | – | arbitrate |
| reply, acknowledged | See Figure 6-18 and Table 6-16 | – | arbitrate |
| open | all | – | open |
| secured | all | – | secured |
| killed | all | – | killed |

C.30 Command response: Faulty command

Table C-30: Faulty command-response table

| Starting State | Condition | Response | Next State |
|----------------|--------------------------------------|---------------------------------------|------------------------------|
| ready | invalid ¹ | – | ready |
| arbitrate | invalid ¹ | – | arbitrate |
| reply | invalid ¹ | – | reply |
| acknowledged | invalid ¹ | – | acknowledged |
| open | unsupported parameters ² | backscatter error code | open |
| | incorrect <u>handle</u> ⁴ | none unless specified by crypto suite | open |
| | improper ⁶ | – | arbitrate |
| | invalid ⁷ | none unless specified by crypto suite | open |
| secured | unsupported parameters ³ | backscatter error code | secured |
| | incorrect <u>handle</u> ⁵ | none unless specified by crypto suite | secured or open ⁹ |
| | improper ⁶ | – | arbitrate |
| | invalid ⁸ | none unless specified by crypto suite | secured or open ⁹ |
| killed | all | – | killed |

- 1: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.
- 2: “Unsupported parameters” shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) a command that requires encapsulation but is nonetheless unencapsulated (see [Table 6-28](#)), (8) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (9) an operation for which the Interrogator has insufficient privileges, (10) an unsupported cryptographic parameter, or (11) other parameters not supported by the Tag.
- 3: “Unsupported parameters” shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, lock payload, blockpermalock payload, KeyID, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) insufficient or unallocateable memory, (8) an unencrypted message that requires encryption, (9) a command that requires encapsulation but is nonetheless unencapsulated (see [Table 6-28](#)), (10) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (11) an RFU privilege value, (12) an operation for which the Interrogator has insufficient privileges, (13) an unsupported cryptographic parameter, or (14) other parameters not supported by the Tag.
- 4: “Incorrect handle” shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving a security command with an incorrect handle.
- 5: “Incorrect handle” shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its crypto engine and transition to the **open** state upon receiving a security command with an incorrect handle.
- 6: “Improper” shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see [Figure 6-24](#) and [Figure 6-26](#)).
- 7: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite/BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving an invalid command.
- 8: “Invalid” shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite/BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine and transition to the **open** state upon



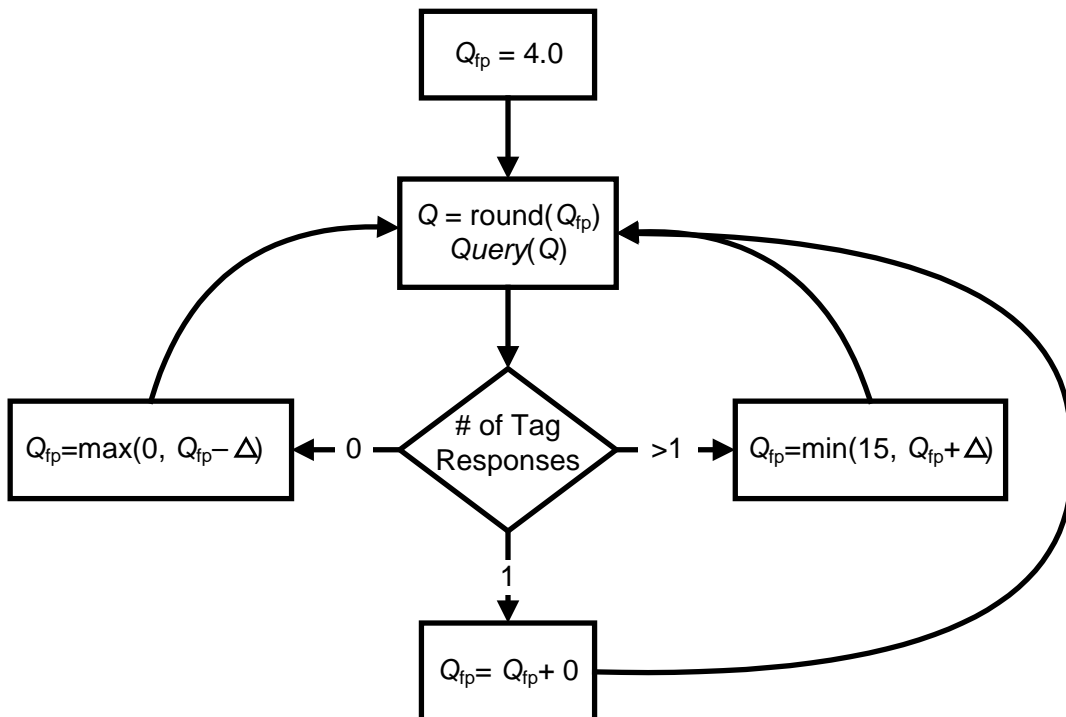
receiving an invalid command.
9: See cryptographic suite.

Annex D (informative) Example slot-count (Q) selection algorithm

D.1 Example algorithm an Interrogator might use to choose Q

[Figure D-1](#) shows an algorithm an Interrogator might use for setting the slot-count parameter Q in a *Query* command. Q_{fp} is a floating-point representation of Q ; an Interrogator rounds Q_{fp} to an integer value and substitutes this integer value for Q in the *Query*. Typical values for Δ are $0.1 < \Delta < 0.5$. An Interrogator typically uses small values of Δ when Q is large, and larger values of Δ when Q is small.

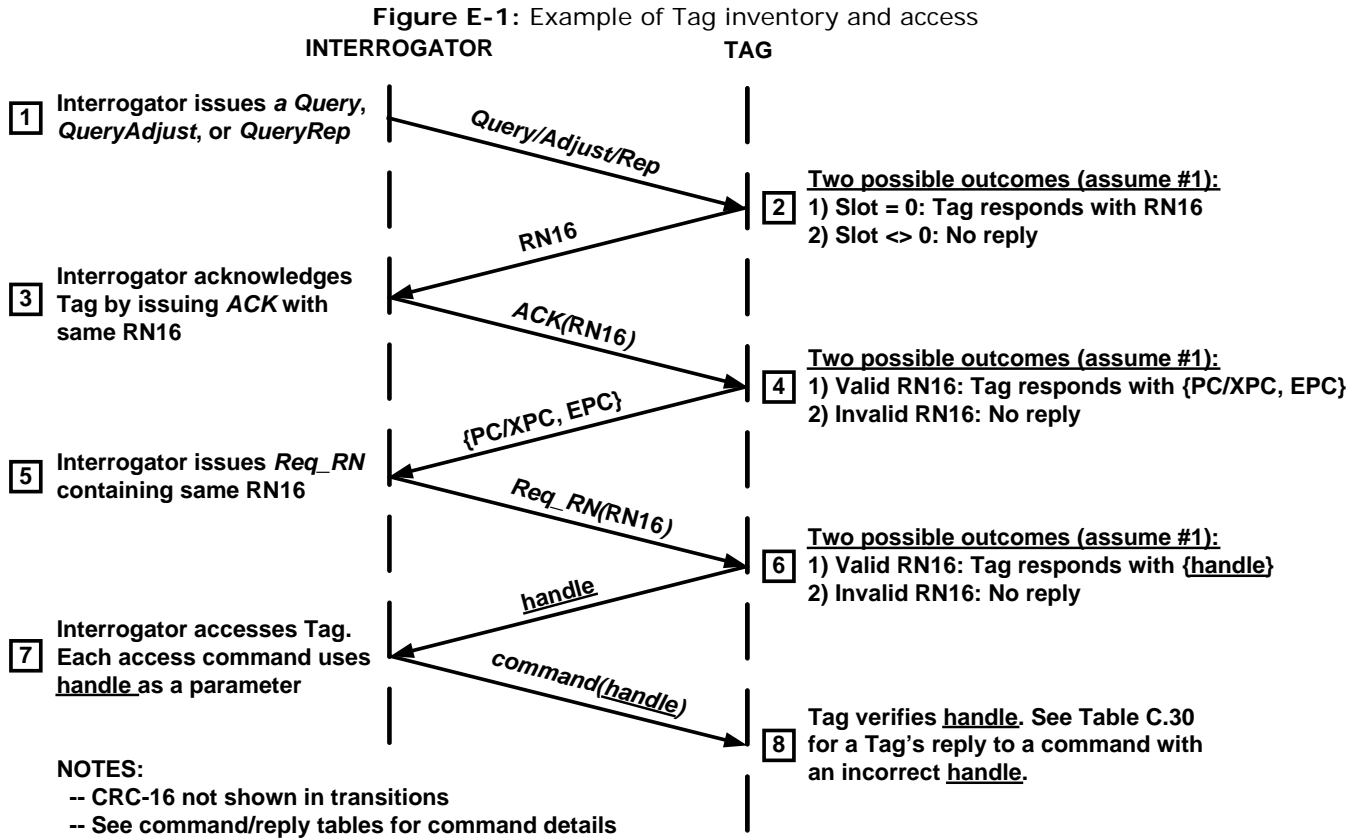
Figure D-1: Example algorithm for choosing the slot-count parameter Q



Annex E (informative) Example Tag inventory and access

E.1 Example inventory and access of a single Tag

[Figure E-1](#) shows the steps by which an Interrogator inventories and accesses a single Tag.



Annex F (informative)

Calculation of 5-bit and 16-bit cyclic redundancy checks

F.1 Example CRC-5 encoder/decoder

An exemplary schematic diagram for a CRC-5 encoder/decoder is shown in [Figure F-1](#), using the polynomial and preset defined in [Table 6-12](#).

To check a CRC-5, first preload the entire CRC register (Q[4:0]) with the value 01001₂, then clock the received data and CRC-5 {data, CRC-5} bits into the input labeled DATA, MSB first. The CRC-5 check passes if the value in Q[4:0] = 00000₂.

Figure F-1: Example CRC-5 circuit

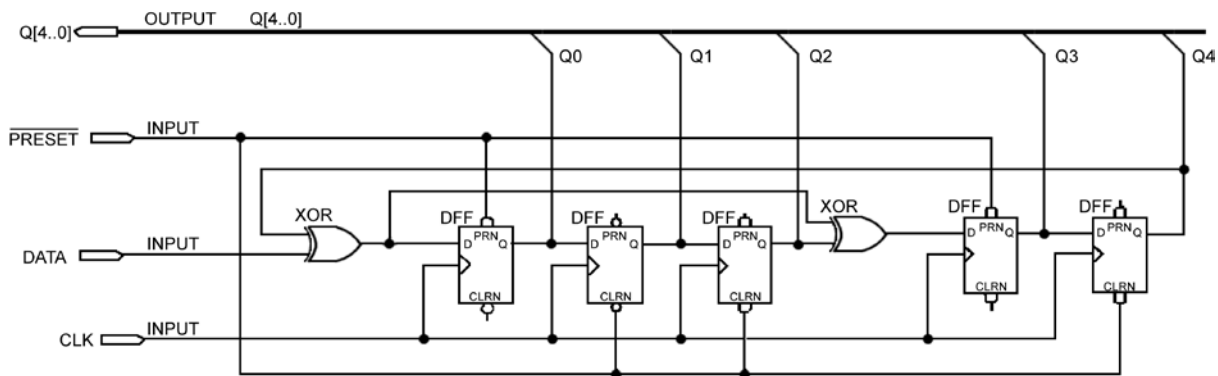


Table F-1: CRC-5 register preload values

| Register | Preload value |
|----------|---------------|
| Q0 | 1 |
| Q1 | 0 |
| Q2 | 0 |
| Q3 | 1 |
| Q4 | 0 |

F.2 Example CRC-16 encoder/decoder

An exemplary schematic diagram for a CRC-16 encoder/decoder is shown in [Figure F-2](#), using the polynomial and preset defined in [Table 6-11](#) (the polynomial used to calculate the CRC-16, $x^{16} + x^{12} + x^5 + 1$, is the CRC-CCITT International Standard, ITU Recommendation X.25).

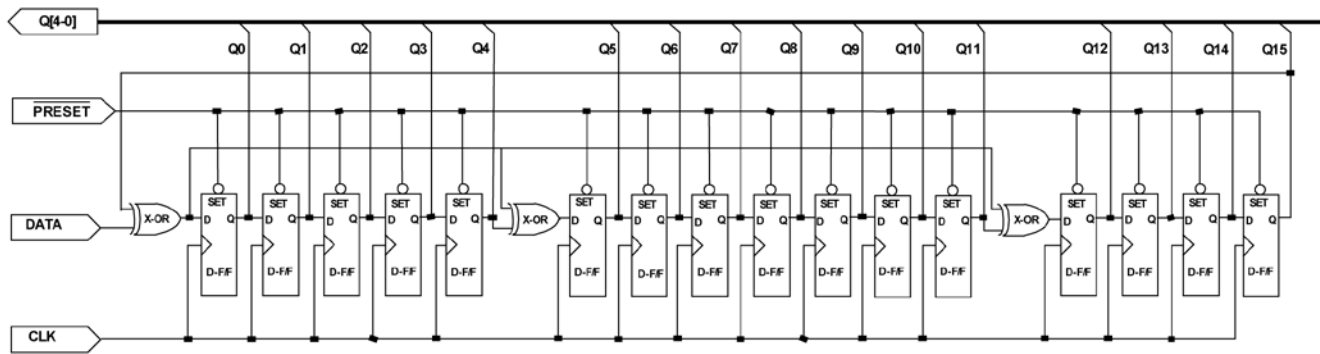
To calculate a CRC-16, first preload the entire CRC register (i.e. Q[15:0], Q15 being the MSB and Q0 the LSB) with the value FFFF_h. Second, clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, Q[15:0] holds the ones-complement of the CRC-16. Third, invert all the bits of Q[15:0] to produce the CRC-16.

There are two methods to check a CRC-16:

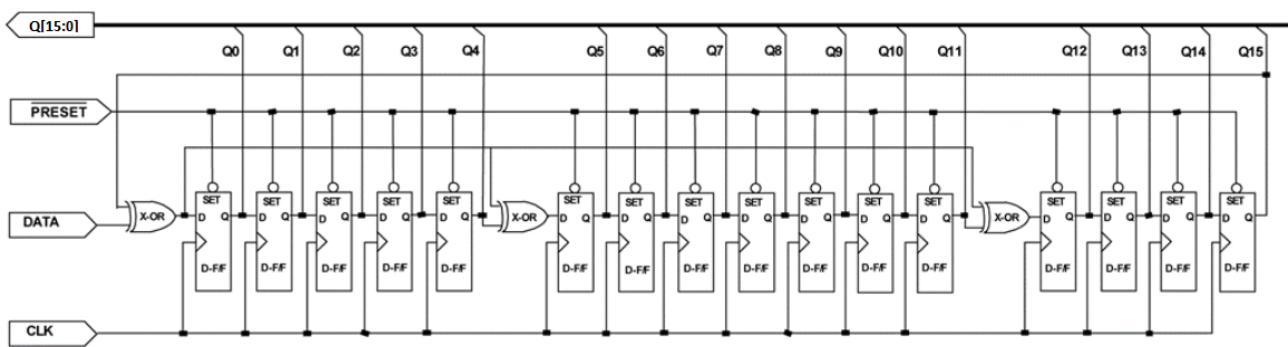
Method 1: First preload the entire CRC register (Q[15:0]) with the value FFFF_h, then clock the received data and CRC-16 {data, CRC-16} bits into the input labeled DATA, MSB first. The CRC-16 check passes if the value in Q[15:0]=1D0F_h.

Method 2: First preload the entire CRC register (Q[15:0]) with the value FFFF_h. Second, clock the received data bits into the input labeled DATA, MSB first. Third, invert all bits of the received CRC-16, and clock the inverted CRC-16 bits into the input labeled DATA, MSB first. The CRC-16 check passes if the value in Q[15:0]=0000_h.

Figure F-2: Example CRC-16 circuit



Delete above



Updated version

F.3 Example CRC-16 calculations

This example shows the StoredCRC (a CRC-16) that a Tag may calculate at power-up.

As shown in [Figure 6-19](#), EPC memory contains a StoredCRC starting at address 00_h, a StoredPC starting at address 10_h, zero or more EPC words starting at address 20_h, an optional XPC_W1 starting at address 210_h, and an optional XPC_W2 starting at address 220_h. As described in [6.3.2.1.2.1](#), a Tag calculates its StoredCRC over its StoredPC and EPC. [Table F-2](#) shows the StoredCRC that a Tag may calculate and logically map into EPC memory at power-up, for the indicated example StoredPC and EPC word values. In each successive column, one more word of EPC memory is written, with the entire EPC memory written in the rightmost column. The indicated StoredPC values correspond to the number of EPC words written, with StoredPC bits 15_h–1F_h set to zero. Entries marked N/A mean that that word of EPC memory is not included as part of the CRC calculation.

Table F-2: EPC memory contents for an example Tag

| EPC word starting address | EPC word contents | EPC word values | | | | | | |
|---------------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| | | 00 _h | 01 _h | 02 _h | 03 _h | 04 _h | 05 _h | 06 _h |
| 00 _h | StoredCRC | E2F0 _h | CCAE _h | 968F _h | 78F6 _h | C241 _h | 2A91 _h | 1835 _h |
| 10 _h | StoredPC | 0000 _h | 0800 _h | 1000 _h | 1800 _h | 2000 _h | 2800 _h | 3000 _h |
| 20 _h | EPC word 1 | N/A | 1111 _h | 1111 | 1111 _h | 1111 _h | 1111 _h | 1111 _h |
| 30 _h | EPC word 2 | N/A | N/A | 2222 _h | 2222 _h | 2222 _h | 2222 _h | 2222 _h |
| 40 _h | EPC word 3 | N/A | N/A | N/A | 3333 _h | 3333 _h | 3333 _h | 3333 _h |
| 50 _h | EPC word 4 | N/A | N/A | N/A | N/A | 4444 _h | 4444 _h | 4444 _h |
| 60 _h | EPC word 5 | N/A | N/A | N/A | N/A | N/A | 5555 _h | 5555 _h |



| EPC word starting address | EPC word contents | EPC word values | | | | | | |
|---------------------------|-------------------|-----------------|-----|-----|-----|-----|-----|-------------------|
| 60 _h | EPC word 6 | N/A | N/A | N/A | N/A | N/A | N/A | 6666 _h |

Annex G (Normative)

Multiple- and dense-Interrogator channelized signaling

This Annex describes channelized signaling in the optional multiple- and dense-Interrogator operating modes. It provides methods that Interrogators may use, as permitted by local authorities, to maximize the spectral efficiency and performance of RFID systems while minimizing the interference to non-RFID systems.

Because regulatory requirements vary worldwide, and even within a given regulatory region are prone to ongoing reinterpretation and revision, this Annex does not specify multiple- or dense-Interrogator operating requirements for any given regulatory region. Instead, this Annex merely outlines the goals of channelized signaling, and defers specification of the Interrogator operating requirements for each individual regulatory region to the channel plans located at <http://www.gs1.org/epcglobal/implementation>.

When an Interrogator in a multiple- or dense-Interrogator environment instructs Tags to use subcarrier backscatter, the Interrogator shall adopt the channel plan found at the above-referenced link for the regulatory region in which it is operating. When an Interrogator in a multiple- and dense-Interrogator environment instructs Tags to use FMO backscatter, the Interrogator shall adopt a channel plan in accordance with local regulations.

Regardless of the regulatory region and the choice of Tag backscatter data encoding,

- Interrogator signaling (both modulated and CW) shall be centered in a channel with the frequency accuracy specified in [6.3.1.2.1](#), unless local regulations specify tighter frequency accuracy, in which case the Interrogator shall meet the local regulations, and
- Interrogator transmissions shall satisfy the multiple- or dense-Interrogator transmit mask in [6.3.1.2.11](#) (as appropriate), unless local regulations specify a tighter mask, in which case the Interrogator shall meet the local regulations.

If an Interrogator uses SSB-ASK modulation, the transmit spectrum shall be centered in the channel during R=>T signaling, and the CW shall be centered in the channel during Tag backscatter.

G.1 Overview of dense-Interrogator channelized signaling (informative)

In environments containing two or more Interrogators, the range and rate at which Interrogators singulate Tags can be improved by preventing Interrogator transmissions from colliding spectrally with Tag responses. This section describes three frequency-division multiplexing (FDM) methods that minimize such Interrogator-on-Tag collisions. In each of these methods, Interrogator transmissions and Tag responses are separated spectrally.

1. *Channel-boundary backscatter*: Interrogator transmissions are constrained to occupy only a small portion of the center of each channel, and Tag backscatter is situated at the channel boundaries.
2. *Alternative-channel backscatter*: Interrogator transmissions are located in a subset of the channels, and Tag backscatter is located in a different subset of the channels.
3. *In-channel backscatter*: Interrogator transmissions are constrained to occupy only a small portion of the center of each channel, and Tag backscatter is situated near but within the channel boundaries.

[Figure G-1](#), shows examples of these FDM dense-Interrogator methods. For optimum performance, the operating requirements located at <http://www.gs1.org/epcglobal/implementation> suggest (but do not require) choosing values for BLF and M that allow a guardband between Interrogator signaling and Tag responses.

Example 1: Channel-boundary backscatter

At the time of this protocol FCC 15.247 authorizes frequency-hopping operation in the ISM band from 902–928 MHz with 500 kHz maximum channel width, and does not prohibit channel-boundary backscatter. In such an environment Interrogators will use 500 kHz channels with channel-boundary backscatter. Example 1 of [Figure G-1](#) shows Interrogator transmissions using PR-ASK modulation with $T_{\text{ari}} = 25 \mu\text{s}$, and 62.5 kbps Tag data backscatter on a 250 kHz

subcarrier (BLF = 250 kHz; $M = 4$). Interrogators center their R=>T signaling in the channels, with transmissions unsynchronized in time, hopping among channels.

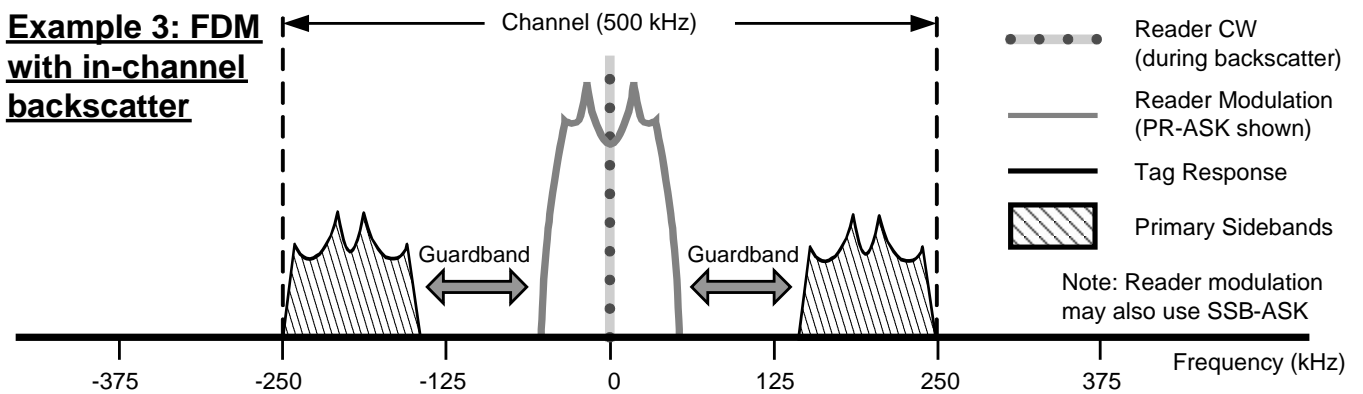
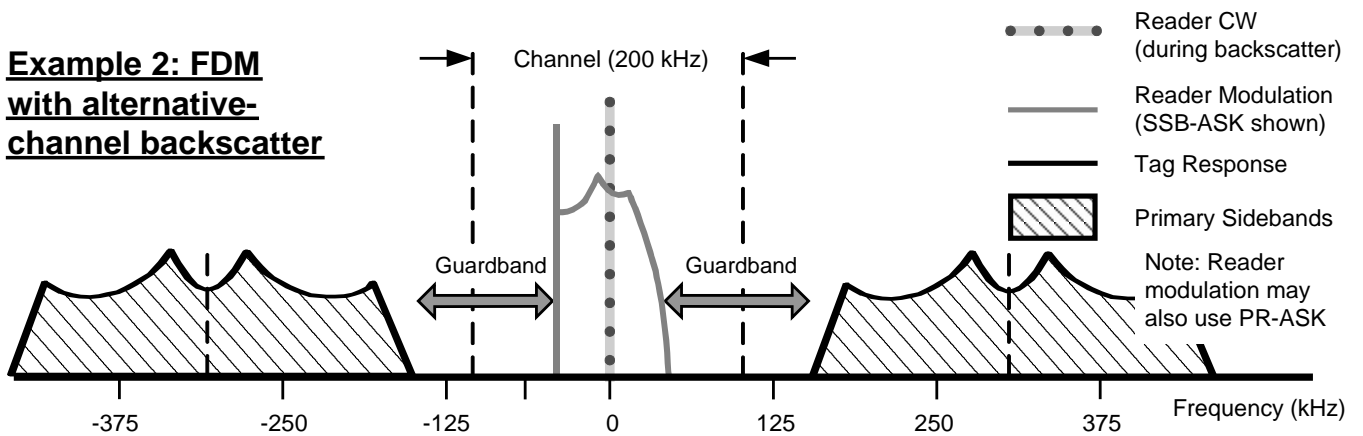
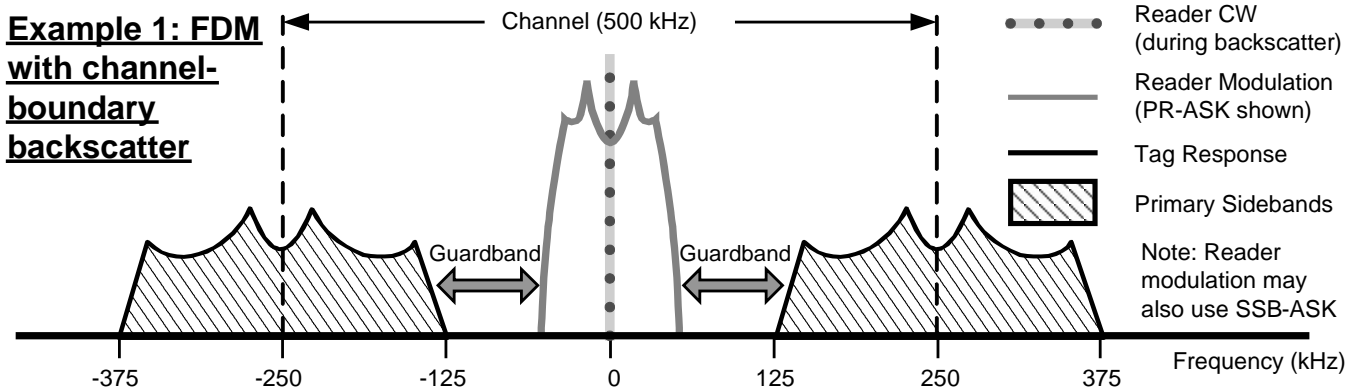
Example 2: Alternative-channel backscatter

At the time of this protocol ETSI EN 302 208 allows four high-power 200 kHz channels, each spaced 600 kHz apart, in the 865–868 MHz frequency range, with adjacent-channel Tag backscatter. In such an environment Interrogators will use alternative-channel backscatter. Example 2 of [Figure G-1](#) shows Interrogator transmissions using SSB-ASK modulation with $T_{\text{ari}} = 25 \mu\text{s}$, and 75 kbps Tag data backscatter on a 300 kHz subcarrier (BLF = 300 kHz, $M = 4$).

Example 3: FDM in-channel backscatter

A hypothetical regulatory region allocates four 500 kHz channels and disallows adjacent-channel and channel-boundary backscatter. In such an environment Interrogators will use in-channel backscatter. Example 3 of [Figure G-1](#) shows Interrogator transmissions using PR-ASK modulation with $T_{\text{ari}} = 25 \mu\text{s}$, and 25 kbps Tag data backscatter on a 200 kHz subcarrier (BLF = 200 kHz, $M = 8$).

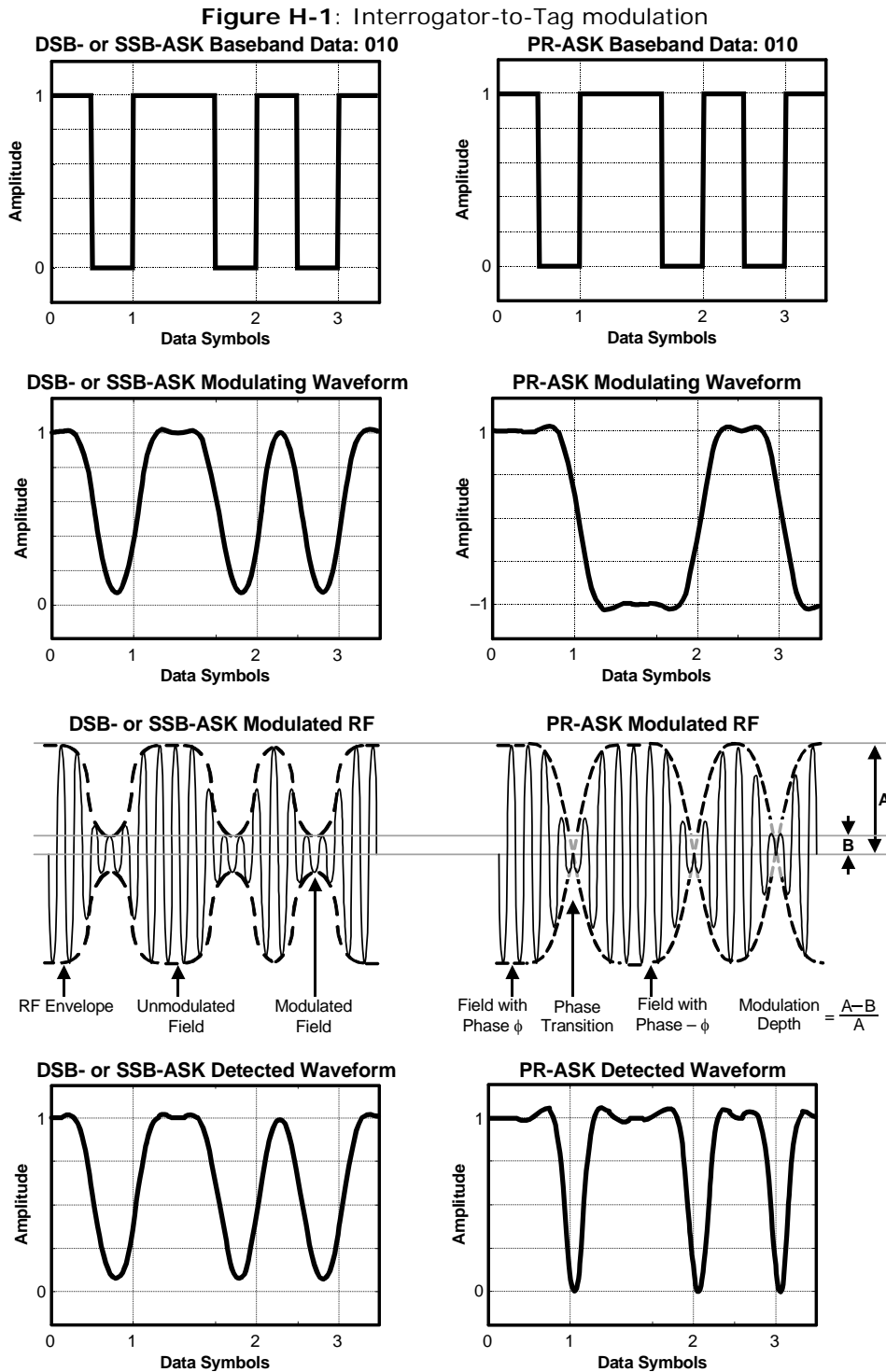
Figure G-1: Examples of dense-Interrogator-mode operation



Annex H (informative) Interrogator-to-Tag link modulation

H.1 Baseband waveforms, modulated RF, and detected waveforms

[Figure H-1](#) shows R=>T baseband and modulated waveforms as generated by an Interrogator, and the corresponding waveforms envelope-detected by a Tag, for DSB- or SSB-ASK modulation, and for PR-ASK modulation.



Annex I (Normative) Error codes

I.1 Tag error codes and their usage

If a Tag is required to backscatter an error code then the Tag shall use one of the error codes shown in [Table I-2](#). See [Annex C](#) for the conditions in which a Tag is required to backscatter an error code.

- If a Tag supports error-specific codes then it shall use the error-specific codes shown in [Table I-2](#).
- If a Tag does not support error-specific codes then it shall backscatter error code 00001111₂ (indicating a non-specific error) as shown in [Table I-2](#), regardless of the error type, including cryptographic errors.
- A Tag shall backscatter error codes only from the **open** or **secured** states.
- A Tag shall not backscatter an error code if it receives an invalid or improper access command, or an access command with an incorrect [handle](#).
- If an error is described by more than one error code then the more specific error code shall take precedence and shall be the code that the Tag backscatters.
- The [header](#) for an error code is a 1-bit, unlike the [header](#) for a success response which is a 0-bit.

Table I-1: Tag error-reply format

| | Header | Error Code | RN | CRC |
|-------------|--------|------------|------------------------|--------|
| # of bits | 1 | 8 | 16 | 16 |
| description | 1 | Error code | handle | CRC-16 |

Table I-2: Tag error codes

| Error-Code Support | Error Code | Error-Code Name | Error Description |
|--------------------|-----------------------|--------------------------|--|
| Error-specific | 00000000 ₂ | Other error | Catch-all for errors not covered by other codes |
| | 00000001 ₂ | Not supported | The Tag does not support the specified parameters or feature |
| | 00000010 ₂ | Insufficient privileges | The Interrogator did not authenticate itself with sufficient privileges for the Tag to perform the operation |
| | 00000011 ₂ | Memory overrun | The Tag memory location does not exist, is too small, or the Tag does not support the specified EPC length |
| | 00000100 ₂ | Memory locked | The Tag memory location is locked or permalocked and is either not writeable ¹ or not readable. |
| | 00000101 ₂ | Crypto suite error | Catch-all for errors specified by the cryptographic suite |
| | 00000110 ₂ | Command not encapsulated | The Interrogator did not encapsulate the command in an <i>AuthComm</i> or <i>SecureComm</i> as required |
| | 00000111 ₂ | ResponseBuffer overflow | The operation failed because the ResponseBuffer overflowed |
| | 00001000 ₂ | Security timeout | The command failed because the Tag is in a security timeout |
| | 00001011 ₂ | Insufficient power | The Tag has insufficient power to perform the operation |
| Non-specific | 00001111 ₂ | Non-specific error | The Tag does not support error-specific codes |

Note 1: Read-only memory is, by definition, not writeable

Annex J (normative) Slot counter

J.1 Slot-counter operation

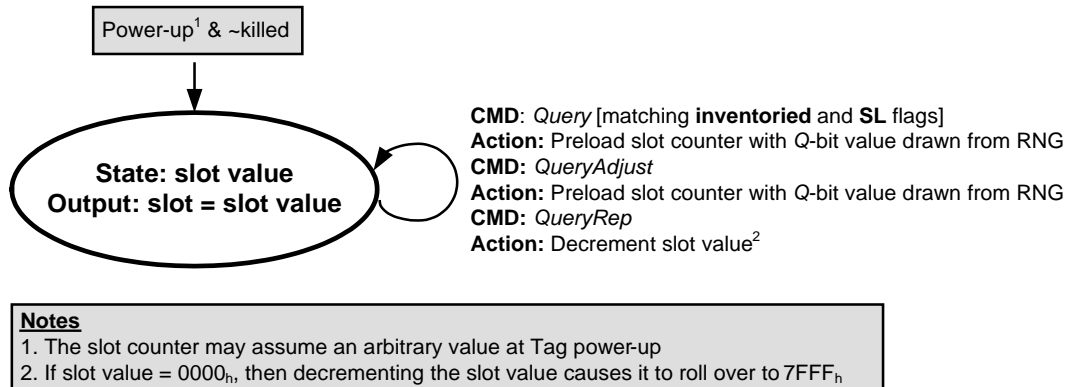
As described in [6.3.2.6.8](#), Tags implement a 15-bit slot counter. As described in [6.3.2.10](#), Interrogators use the slot counter to regulate the probability of a Tag responding to a *Query*, *QueryAdjust*, or *QueryRep* command. Upon receiving a *Query* or *QueryAdjust* a Tag preloads a *Q*-bit value, drawn from the Tag's RNG (see [6.3.2.7](#)), into its slot counter. *Q* is an integer in the range (0, 15). A *Query* specifies *Q*; a *QueryAdjust* may modify *Q* from the prior *Query*.

A Tag in the **arbitrate** state shall decrement its slot counter every time it receives a *QueryRep* command, transitioning to the **reply** state and backscattering an RN16 when its slot-counter value reaches 0000_h. A Tag whose slot-counter value reached 0000_h, who replied, and who was not acknowledged (including a Tag that responded to the original *Query* and was not acknowledged) returns to **arbitrate** with a slot-counter value of 0000_h.

A Tag that returns to **arbitrate** with a slot-counter value of 0000_h shall decrement its slot-counter from 0000_h to 7FFF_h (i.e. the slot counter rolls over) at the next *QueryRep* with matching session. Because the slot-counter value is now nonzero, the Tag remains in **arbitrate**. Slot counters implements continuous counting, meaning that, after a slot counter rolls over it begins counting down again from 7FFF_h, effectively preventing subsequent Tag replies until the Tag receives either a *Query* or a *QueryAdjust* and loads a new random value into its slot counter.

[Annex B](#) and [Annex C](#) contain tables describing a Tag's response to Interrogator commands; "slot" is a parameter in these tables.

Figure J-1: Slot-counter state diagram



Notes

1. The slot counter may assume an arbitrary value at Tag power-up
2. If slot value = 0000_h, then decrementing the slot value causes it to roll over to 7FFF_h

Annex K (informative) Example data-flow exchange

K.1 Overview of the data-flow exchange

The following example describes a data exchange, between an Interrogator and a single Tag, during which the Interrogator reads the kill password stored in the Tag's Reserved memory. This example assumes that:

- The Tag has been singulated and is in the **acknowledged** state.
- The Tag's Reserved memory is locked (but not permalocked), so the Interrogator issues the access password and transitions the Tag to the **secured** state before performing the read operation.
- The random numbers the Tag generates (listed in sequence, and not random for reasons of clarity) are:
 - RN16_0 1600_h (the RN16 the Tag backscattered prior to entering **acknowledged**)
 - RN16_1 1601_h (will become the handle for the entire access sequence)
 - RN16_2 1602_h
 - RN16_3 1603_h
- The Tag's EPC is 64 bits in length.
- The Tag's access password is ACCECODE_h.
- The Tag's kill password is DEADCODE_h.
- The 1st half of the access password EXORed with RN16_2 = ACCE_h ⊗ 1602_h = BACC_h.
- The 2nd half of the access password EXORed with RN16_3 = CODE_h ⊗ 1603_h = D6DD_h.

K.2 Tag memory contents and lock-field values

[Table K-1](#) and [Table K-2](#) show the example Tag memory contents and lock-field values, respectively.

Table K-1: Tag memory contents

| Memory Bank | Memory Contents | Memory Addresses | Memory Values |
|-------------|------------------------|----------------------------------|--|
| TID | TID[15:0] | 10 _h –1F _h | 54E2 _h |
| | TID[31:16] | 00 _h –0F _h | A986 _h |
| EPC | EPC[15:0] | 50 _h –5F _h | 3210 _h |
| | EPC[31:16] | 40 _h –4F _h | 7654 _h |
| | EPC[47:32] | 30 _h –3F _h | BA98 _h |
| | EPC[63:48] | 20 _h –2F _h | FEDC _h |
| | StoredPC[15:0] | 10 _h –1F _h | 2000 _h |
| | StoredCRC[15:0] | 00 _h –0F _h | as calculated (see Annex F) |
| Reserved | access password[15:0] | 30 _h –3F _h | CODE _h |
| | access password[31:16] | 20 _h –2F _h | ACCE _h |
| | kill password[15:0] | 10 _h –1F _h | CODE _h |
| | kill password[31:16] | 00 _h –0F _h | DEAD _h |

Table K-2: Lock-field values

| Kill Password | | Access Password | | EPC Memory | | TID Memory | | User Memory | |
|---------------|---|-----------------|---|------------|---|------------|---|-------------|-----|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | N/A | N/A |

K.3 Data-flow exchange and command sequence

The data-flow exchange follows the *Access* procedure outlined in [Figure 6-26](#) with a *Read* command added at the end. The sequence of Interrogator commands and Tag replies is:

- Step 1: *Req_RN*[RN16_0, CRC-16]
 Tag backscatters RN16_1, which becomes the handle for the entire access sequence
- Step 2: *Req_RN*[handle, CRC-16]
 Tag backscatters RN16_2
- Step 3: *Access*[access password[31:16] EXORed with RN16_2, handle, CRC-16]
 Tag backscatters handle
- Step 4: *Req_RN*[handle, CRC-16]
 Tag backscatters RN16_3
- Step 5: *Access*[access password[15:0] EXORed with RN16_3, handle, CRC-16]
 Tag backscatters handle
- Step 6: *Read*[MemBank=Reserved, WordPtr=00_h, WordCount=2, handle, CRC-16]
 Tag backscatters kill password

[Table K-3](#) shows the detailed Interrogator commands and Tag replies. For reasons of clarity, the CRC-16 has been omitted from all commands and replies.

Table K-3: Interrogator commands and Tag replies

| Step | Data Flow | Command | Parameter and/or Data | Tag State |
|---------------------------|-----------|----------|---|---------------------|
| 1a: <i>Req_RN</i> command | R => T | 11000001 | 0001 0110 0000 0000 (RN16_0=1600 _h) | acknowledged → open |
| 1b: Tag response | T => R | | 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | |
| 2a: <i>Req_RN</i> command | R => T | 11000001 | 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | open → open |
| 2b: Tag response | T => R | | 0001 0110 0000 0010 (RN16_2=1602 _h) | |
| 3a: <i>Access</i> command | R => T | 11000110 | 1011 1010 1100 1100 (BACC _h) 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | open → open |
| 3b: Tag response | T => R | | 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | |
| 4a: <i>Req_RN</i> command | R => T | 11000001 | 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | open → open |
| 4b: Tag response | T => R | | 0001 0110 0000 0011 (RN16_2=1603 _h) | |
| 5a: <i>Access</i> command | R => T | 11000110 | 1101 0110 1101 1101 (D6DD _h) 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | open → secured |
| 5b: Tag response | T => R | | 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | |
| 6a: <i>Read</i> command | R => T | 11000010 | 00 (MemBank=Reserved) 00000000 (WordPtr=kill password) 00000010 (WordCount=2) 0001 0110 0000 0001 (<u>handle</u> =1601 _h) | secured → secured |
| 6b: Tag response | T => R | | 0 (header) 1101 1110 1010 1101 (DEAD _h) 1100 0000 1101 1110 (CODE _h) | |

Annex L (informative) Optional Tag Features

The following options are available to Tags certified to this protocol.

L.1 Optional Tag memory banks, memory-bank sizes, and files

Reserved memory: Reserved memory is optional. If a Tag does not implement kill and access passwords then the Tag need not physically implement Reserved memory. Because a Tag with non-implemented passwords operates as if it has zero-valued password(s) that are permanently read/write locked, these passwords must still be logically addressable in Reserved memory at the memory locations specified in [6.3.2.1.1.1](#) and [6.3.2.1.1.2](#).

EPC memory: EPC memory is required, but its size is Tag-manufacturer defined. The minimum size is 32 bits, to contain a 16-bit StoredCRC and a 16-bit StoredPC. EPC memory may be larger than 32 bits, to contain an EPC whose length may be 16 to 496 bits (if a Tag does not support XPC functionality) or to 464 bits (if a Tag supports XPC functionality), as well as an optional XPC word or words. See [6.3.2.1.2](#).

TID memory: TID memory is required, but its size is Tag-manufacturer defined. The minimum-size TID memory contains an 8-bit ISO/IEC 15963 allocation class identifier (either E0_h or E2_h) in memory locations 00_h to 07_h, as well as sufficient identifying information for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. TID memory may optionally contain other data. See [6.3.2.1.3](#).

User memory: User memory is optional. A Tag may partition User memory into one or more files whose memory allocation may be *static* or *dynamic*. The Tag manufacturer chooses where a Tag stores its [FileType](#) and [FileNum](#) data. The Tag manufacturer also chooses the file-allocation block size (from one to 1024 words). User memory and the files in it may be encoded according to the GS1 EPC Tag Data Standard or to ISO/IEC 15961/15962. See [6.3.2.1.4](#), [6.3.2.1.4.1](#), [6.3.2.1.4.2](#), and [6.3.2.11.3](#).

L.2 Optional Tag commands

Proprietary: A Tag may support proprietary commands. See [2.3.3](#).

Custom: A Tag may support custom commands. See [2.3.4](#).

Challenge: A Tag may support the *Challenge* command. See [6.3.2.12.1.2](#).

Access: A Tag may support the *Access* command. See [6.3.2.12.3.6](#).

BlockWrite: A Tag may support the *BlockWrite* command. See [6.3.2.12.3.7](#).

BlockErase: A Tag may support the *BlockErase* command. See [6.3.2.12.3.8](#).

BlockPermalock: A Tag may support the *BlockPermalock* command. See [6.3.2.12.3.9](#).

Authenticate: A Tag may support the *Authenticate* command. See [6.3.2.12.3.10](#).

AuthComm: A Tag may support the *AuthComm* command. [6.3.2.12.3.11](#).

SecureComm: A Tag may support the *SecureComm* command. See [6.3.2.12.3.12](#).

KeyUpdate: A Tag may support the *KeyUpdate* command. See [6.3.2.12.3.13](#).

TagPrivilege: A Tag may support the *TagPrivilege* command. See [6.3.2.12.3.14](#).

ReadBuffer: A Tag may support the *ReadBuffer* command. See [6.3.2.12.3.15](#).

Untraceable: A Tag may support the *Untraceable* command. See [6.3.2.12.3.16](#).

FileOpen: A Tag may support the *FileOpen* command. See [6.3.2.12.3.17](#).

FileList: A Tag may support the *FileList* command. See [6.3.2.12.3.18](#).

FilePrivilege: A Tag may support the *FilePrivilege* command. See [6.3.2.12.3.19](#).

FileSetup: A Tag may support the *FileSetup* command. See [6.3.2.12.3.20](#).

L.3 Optional Tag passwords, security, and keys

Kill password: A Tag may implement a kill password. A Tag that does not implement a kill password operates as if it has a zero-valued kill password that is permanently read/write locked. See [6.3.2.1.1.1](#).

Access password: A Tag may implement an access password. A Tag that does not implement an access password operates as if it has a zero-valued access password that is permanently read/write locked. See [6.3.2.1.1.2](#).

Security timeout: A Tag may implement a security timeout. See [6.3.2.5](#).

Cryptographic security and keys: A Tag may support one or more cryptographic suites, each of which may implement Tag, Interrogator, and/or mutual authentication. A Tag may support up to 256 keys. A key may have a [CryptoSuperuser](#), [AuthKill](#), [Untraceable](#), or a [DecFilePriv](#) privilege. A key may also have a property defined by the cryptographic suite and/or a custom property. See [6.3.2.11.2](#).

Random-number generator: A cryptographic suite may specify different RNG requirements for cryptographic operations than for other Tag operations. See [6.3.2.7](#).

L.4 Optional Tag replies

A Tag may implement an *in-process* reply and a ResponseBuffer. See [6.3.1.6.3](#) and [6.3.1.6.4](#).

L.5 Optional Tag PC and XPC bit designations and values

PC: The **UMI** bit may be fixed by the Tag manufacturer or computed by the Tag. A Tag may be used in a GS1 EPCglobal™ or a non-GS1 EPCglobal™ Application (**T**=0 or **T**=1 respectively). If **T**=0 then the **XI** bit may be either (i) the logical OR of bits 210_h–217_h of XPC_W1 or (ii) the logical OR of bits 210_h–218_h of XPC_W1; the Tag manufacturer shall choose whether the Tag implements (i) or (ii). If **T**=1 then **XI** is the logical OR of the entirety of XPC_W1 (210_h–21F_h). Bits 18_h – 1F_h of the StoredPC may be RFU or an AFI. When forming a PacketPC, a Tag may substitute bits 218_h–21F_h of its XPC_W1 for bits 18_h–1F_h of its StoredPC. See [6.3.2.1.2.2](#).

XPC: A Tag may implement an XPC_W1. If the Tag implements an XPC_W1 then it may implement an XPC_W2. A Tag's XPC_W1 may support the **XEB**, **B**, **C**, **SLI**, **TN**, **U**, **K**, **NR**, and **H** flags, corresponding to whether the Tag has a nonzero XPC_W2, a battery, a computed response, an asserted **SL** flag, a Tag notification, an untraceability indicator, is killable, is not removable, or is attached to hazardous material. These XPC_W1 bits may be fixed, written, computed, or set based on the application. Bits 211_h–217_h of XPC_W1 may be zero or as defined in ISO/IEC 18000-63. All bits of XPC_W2 may be zero or as defined in ISO/IEC 18000-63. See [6.3.2.1.2.5](#).

L.6 Optional Tag error-code reporting format

A Tag may support error-specific or non-error-specific error-code reporting. See [Annex I](#).

L.7 Optional Tag backscatter modulation format

A Tag may support ASK and/or PSK backscatter modulation. See [6.3.1.3.1](#).

L.8 Optional Tag functionality

A Tag may use one of the methods described in [Annex F](#) to verify a CRC-5 or a CRC-16. See [6.3.1.5](#).

A Tag may compute its CRC-16 using one of two methods. A Tag may store the CRC-16 in volatile or nonvolatile memory. See [6.3.2.1.2.1](#).

A Tag may limit access to the **secured** state via one or more physical mechanisms. See [6.3.2.11](#).

A Tag may support authenticated kill. See [6.3.2.12.3.4](#).

A Tag may store its lock bits in a readable or an unreadable memory location. See [6.3.2.12.3.5](#).

Annex M (informative) Cryptographic-Suite Checklist

A cryptographic suite typically includes at a minimum the parameters, functionality, procedures, behavior, and error conditions shown in the checklist of [Table M-1](#).

Table M-1: Required elements of a cryptographic suite

| Required Element of a Cryptographic Suite | Included? |
|--|-----------|
| A <u>CSI</u> | |
| Message fields for commands supported by the cryptographic suite from the set <i>Challenge</i> , <i>Authenticate</i> , <i>AuthComm</i> , <i>SecureComm</i> , and <i>KeyUpdate</i> . | |
| Functional descriptions of and bit fields for all parameters, formatting, encoding, decoding, and integrity protection used in or by all <u>messages</u> and <u>responses</u> for the supported cryptographic commands. | |
| Conditions in which one or more security commands (see 6.3.2.11.2) cause a Tag to enforce a security timeout | |
| Conditions under which the Tag resets its cryptographic engine and the parameters of such a reset | |
| Randomness requirements including RN length, uniqueness, predictability, generation rate, nonce, etc. | |
| How an Interrogator specifies a <u>KeyID</u> in a <u>message</u> field. | |
| Whether <u>messages</u> and/or <u>responses</u> include an explicit step count. | |
| Whether the cryptographic suite requires a <i>KeyUpdate</i> to be encapsulated in a <i>SecureComm</i> or an <i>AuthComm</i> . | |
| The formatting of every <u>message</u> and every <u>response</u> for every cryptographic command | |
| Whether and when an Interrogator and/or a Tag use authenticated or secure communications | |
| Flow diagrams showing sequencing, timing, state transitions, and error conditions for every step in every operation supported by the cryptographic suite, including but not limited to: | |
| Whether the Tag or Interrogator require one or more wait states in the generation, execution, or receipt of any cryptographic commands or responses. | |
| Mechanism(s) for a multipass authentication (if any) | |
| How Tag and Interrogator derive session keys (if any) | |
| The beginning and ending Tag state for every step in a cryptographic authentication | |
| Definitions for and locations in the flow diagrams for when (1) a Tag considers an Interrogator to be authenticated, and (2) an Interrogator considers a Tag to be authenticated | |
| An enumeration of every potential cryptographic error that an Interrogator or a Tag may encounter and the Interrogator and Tag behavior for each. A few representative examples of such errors include (1) one entity replies incorrectly or fails to reply part way thru a cryptographic operation, (2) one entity cannot complete a cryptographic computation, (3) one entity replies with an incorrect step count, and (4) many others. | |

Annex N (normative) Application Conformance

To be certified as alteration-EAS, Tag-alteration, and/or consumer-electronics conformant, Tags and Interrogators shall support the optional clauses or portions of optional clauses cited in the corresponding sections of [Table N-1](#), respectively, as mandatory. To be clear, those features in the cited optional clause or portion of optional clause specified with a “may” shall become a “shall”; those specified with a “shall” shall remain a “shall”.

Table N-1: Required clauses for certification, by type

| Alteration EAS | Tag Alteration (Core) | Consumer Electronics |
|--|--|--|
| 6.3.2.1.3 <ul style="list-style-type: none"> an E2_h class identifier and an XTID (see 4.1) are mandatory a nonzero XTID serialization field is mandatory | 6.3.2.1.3 <ul style="list-style-type: none"> an E2_h class identifier and an XTID (see 4.1) are mandatory a nonzero XTID serialization field is mandatory | 6.3.2.1.3 <ul style="list-style-type: none"> an E2_h class identifier and an XTID (see 4.1) are mandatory a nonzero XTID serialization field is mandatory |
| 6.3.2.1.2.5 SLI and K bits in XPC_W1 are mandatory | 6.3.2.1.2.5 SLI , K , and NR bits in XPC_W1 are mandatory | 6.3.2.1.2.5 SLI , K , NR , and H bits in XPC_W1 are mandatory |
| 6.3.2.11 a Tag shall implement the mechanisms in this protocol that prevent it from transitioning directly from the acknowledged to the secured state | 6.3.2.11 a Tag shall implement the mechanisms in this protocol that prevent it from transitioning directly from the acknowledged to the secured state | 6.3.2.11 a Tag shall implement the mechanisms in this protocol that prevent it from transitioning directly from the acknowledged to the secured state |
| 6.3.2.12.3.16 <i>Untraceable</i> command ¹ . A Tag may, but is not required to, support range reduction or the U bit. | 6.3.2.12.3.16 <i>Untraceable</i> command ¹ . A Tag may, but is not required to, support range reduction or the U bit. | 6.3.2.12.3.9 <i>BlockPermalock</i> command ¹ |
| 6.3.2.5 Security timeout for the <i>Access</i> command with a timeout range as specified in Table 6-20 is mandatory | 6.3.2.5 Security timeout for the <i>Access</i> command with a timeout range as specified in Table 6-20 is mandatory | 6.3.2.5 Security timeout for the <i>Access</i> command with a timeout range as specified in Table 6-20 is mandatory |
| 6.3.2.1.4.1 ≥32 bits User memory is mandatory | Intentionally left blank | 6.3.2.12.3.17 <i>FileOpen</i> command ¹ |
| Intentionally left blank | Intentionally left blank | 6.3.2.11.3 At least 2 files are mandatory |
| Tag Alteration (Challenge) | Tag Alteration (Authenticate) | Tag Alteration (Full) |
| All requirements of Tag Alteration (Core) | All requirements of Tag Alteration (Core) | All requirements of Tag Alteration (Core) |
| 6.3.2.12.1.2 <i>Challenge</i> command ¹ | 6.3.2.12.3.10 <i>Authenticate</i> command ¹ | 6.3.2.12.1.2 <i>Challenge</i> command ¹ |
| Intentionally left blank | Intentionally left blank | 6.3.2.12.3.7 <i>Blockwrite</i> command ¹ |
| Intentionally left blank | Intentionally left blank | 6.3.2.12.3.10 <i>Authenticate</i> command ¹ |
| Intentionally left blank | Intentionally left blank | 6.3.2.12.3.12 <i>SecureComm</i> command ¹ |
| Intentionally left blank | Intentionally left blank | 6.3.2.12.3.13 <i>KeyUpdate</i> command ¹ |

1: The command in its entirety is mandatory