The Global Language of Business

EPC® Radio-Frequency Identity Generation-2 UHF RFID Standard

Specification for RFID Air Interface Protocol for Communications at 860 MHz – 930 MHz

Release 3.0, Ratified, Jan 2024

Document Summary

Document Item	Current Value
Document Name	EPC® Radio-Frequency Identity Generation-2 UHF RFID Standard
Document Date	Jan 2024
Document Version	3.0
Document Issue	
Document Status	Ratified
Document Description	Specification for RFID Air Interface Protocol for Communications at 860 MHz – 930 MHz

Contributors

Name	Organisation
Franz Amtmann	NXP Semiconductors
Lucelena Angarita	GS1 US
Koji Asano	GS1 Japan
Benjamin Bekritsky	Zebra Technologies Corporation
Zaid Ben Hmad	DECATHLON
Kevin Berisso	BAIT Consulting
Klaus Brandau	Deutsche Bahn AG
Megan Brewster (co-chair)	**Impinj, Inc**
Chris Brown	Printronix Auto ID
Arnaud Carle	AXEM Technology
Tony Ceder	Charmingtrim
HJ Cha	Avery Dennison RFID
Jiraporn Chalermjirarat	GS1 Thailand
Jeffrey Chen	Printronix Auto ID
Jasper Compaijen	Nedap
Luiz Costa	GS1 Brasil
Deirdre Courtney	GS1 Global Office
Norma Crockett	GS1 US
Oscar Cruz	GS1 Mexico
Huipeng Deng	GS1 China
Vojtech Derbek	CISC Semiconductor GmbH
Herve d'Halluin	DECATHLON
François-Régis Dousset	DANONE SPA
Sanjiv Dua	RFID4U
Jeanne Duckett	Avery Dennison RFID
Fadi El-Turk	GS1 UK
Don Ferguson	Lyngsoe Systems Ltd.
Ted Firlit	EM Microelectronic

Name	Organisation
Thomas Frederick (co-chair)	**Clairvoyant Technology LLC**
James Goodland	NXP Semiconductors
Heinz Graf	GS1 Switzerland
Jonathan Gregory	GS1 US
Danny Hakk	Nedap
Falk Habrichs	Lambda ID GmbH
Dominik Halbeisen	GS1 Switzerland
Gary Hartley	GS1 New Zealand
Andrew Hearn	GS1 Global Office
Olle Hellman	GS1 Sweden
Sven Hofmann	race result AG
Shinichi Ike	Johnson & Johnson
Michael Isabell	eAgile Inc.
Olivier Joyez	DECATHLON
Steven Keddie	GS1 Global Office
Kazuna Kimura	GS1 Japan
Nikias Klohr	race result AG
Keiko Konishi	GS1 Japan
Akshay Koshti	Robert Bosch GmbH
Philippe Lallement	Manufacture francaise des Pneumatiques Michelin
Mauricio León	Lambda ID GmbH
Zhimin Li	GS1 China
Huiru Lou	GS1 China
Marisa Lu	GS1 Chinese Taipei
Wayne Luk	GS1 Hong Kong, China
Yan Luo	GS1 China
Vivek Mangal	WiLO Networks Inc.
Rene Martinez (editor)	**Impinj, Inc**
Ned Mears	GS1 US
Jan Merckx	GS1 Netherlands
Andrew Meyer	GS1 US
Jin Mitsugi	Auto-ID Labs at Keio University
Akane Mitsui	Avery Dennison RFID
Satoshi Mizuno	DENSO WAVE Incorporated
Gena Morgan	GS1 US
Thuy Nguyen	GS1 Vietnam
Masatoshi Oka	TOPPAN
Yousuke Okayama	DENSO WAVE Incorporated
Sergio Pastrana	GS1 Mexico
Gao Peng	GS1 China

Name	Organisation
Peter Phaneuf	eAgile Inc.
Sarina Pielaat	GS1 Netherlands
Josef Preishuber-Pflügl	innobir e.U.
Albertus Pretorius	Tonnjes ISI Patent Holding GmbH
Mo Ramzan	SML
Craig Alan Repec	GS1 Global Office
Norida Rios Pino	Logyca
Matt Robshaw	Impinj, Inc
J John Ryu (facilitator)	**GS1 Global Office**
Laura Sanchez Cabrera	EM Microelectronic
Bernhard Santer	CISC Semiconductor GmbH
Mayu Sasase	GS1 Japan
Yuki Sato	GS1 Japan
Fabian Moritz Schenk	Lambda ID GmbH
Eugen Sehorz	GS1 Austria
Karmik Shah	Kevision
Nicolas Sogoyan	STID
Jim Springer (editor)	**EM Microelectronic**
KK Suen	GS1 Hong Kong, China
Claude Tételin (lead editor)	**GS1 Global Office**
Jérôme Torro	SNCF Rolling Stock Department
Jesse Tuominen	Voyantic Ltd
Ricardo Verza Amaral Melo	GS1 Brasil
Taira Wakamiya	TOPPAN Edge
Amber Walls	GS1 US
Owen Williams	Clairvoyant Technology LLC
Roman Winter	GS1 Germany
Ruoyun Yan	GS1 China
Shi Yu	Beijing REN JU ZHI HUI Technology Co. Ltd.
Hermann Zach	NXP Semiconductors
Yibo Zhang	Alibaba Group

Log of Changes

Release	Date of Change	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.

Release	Date of Change	Summary of Change
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.
2.1.0	Jul 2018	Modified to harmonize with ISO 18000-63 and fixed multiple errata.
2.1	Jul 2018	Updated to new GS1 branding and WR (Work Request) 17-223. (*Briefly, and incorrectly, published as version 3.0*).
3.0	2023	WR (Work request) 21-320. Modified Gen2 protocol v2.1 to satisfy GS1 Gen2v3 Business Requirements Analysis Document and fixed multiple errata.

Disclaimer

Table of Contents

Index of figures

Index of tables

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 – 930 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 – 930 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. all clauses in this protocol (except those marked as optional), and
b. the conformance document associated with this protocol, and,
c. all local radio regulations.

To be certified as alteration-electronic article surveillance, 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 demodulate/receive 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 6-29* 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 is reserving these RFU values for future extensibility. Under some circumstances GS1 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 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:

  - 0000_2 – 0011_2: ISO/IEC 29167
  - 0100_2 – 1100_2: RFU
  - 1101_2: Tag manufacturer
  - 1110_2: GS1
  - 1111_2: RFU
- Four least-significant bits: One of 16 cryptographic suites that the assigning authority may assign.

Example: CSI=00000000_2 is the first and CSI=00000001_2 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: EPC Tag Data Standard (TDS)

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 — Radio frequency identification for item management — Part 63: Parameters for air interface communications at 860 MHz to 960 MHz Type C*

ISO/IEC 19762: *Information technology – Automatic identification and data capture (AIDC) techniques – Harmonized vocabulary*

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

4. Terms and definitions

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.

Backscatter link frequency

Frequency at which at Tag modulates the backscattered signal.

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 bitwise XOR (exclusive OR) 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 bitwise XOR 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 0011_2 **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.

Handle

16-bit Tag identifier

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 or a *QueryX* command and terminated by either a subsequent *Query* command, a subsequent *QueryX* command, 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.

Pivot

Decision threshold differentiating an R=>T data-0 symbol from a data-1 symbol

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*.

Snapshot sensor

Sensor that makes a measurement during power-up or on demand from an Interrogator

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.

Word

16-bit long data element.

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/TRcal)
C	Computed-response indicator
CSI	Cryptographic suite identifier
DBLF	Digital backscatter link frequency
DR	Divide ratio
f_0	Offset frequency
F	File-services indicator (whether a Tag supports the *FileOpen* command)
f_c	Center frequency
FrT	Frequency tolerance
FS	Full function sensor indicator
H	Hazmat indicator
K	Killable indicator
M	Number of subcarrier cycles per symbol
M_{adjd}	RF field adjust down
M_{adju}	RF field adjust up
M_h	RF envelope ripple (overshoot)
M_{adjh}	RF envelope ripple adjusted (overshoot)
M_{hp}	RF overshoot, power-up
M_l	RF envelope ripple (undershoot)
M_{adjl}	RF envelope ripple adjusted (undershoot)
M_{lp}	RF undershoot, power-up
M_s	RF signal level when OFF
NR	Nonremovable indicator
P	Integrated power in a channel
Q	Slot-count parameter
R=>T	Interrogator-to-Tag
RTcal	Interrogator-to-Tag calibration symbol
RUM	Read user memory indicator
S	Security-services indicator (whether a Tag supports the *Challenge* and/or *Authenticate* commands)
SA	Sensor alarm indicator
SLI	**SL** indicator
SN	Snapshot sensor indicator
SS	Simple sensor indicator
S_x	State label for T=>R encoding

T	Numbering system identifier
T_1	Time from Interrogator transmission to Tag response for an *immediate* Tag reply
T_2	Time from Tag response to Interrogator transmission
T_3	Time an Interrogator waits, after T_1, before it issues another command
T_4	Minimum time between Interrogator commands
T_5	Time from Interrogator transmission to Tag response for a *delayed* Tag reply
T_6	Time from Interrogator transmission to first Tag response for an *in-process* Tag reply
T_7	Time between Tag responses for an *in-process* Tag reply
T_8	Time from end of *QueryX* or *QueryY* command to beginning of a subsequent *QueryY*
T_{cp}	Time before command, power-up
$t_{f,10-90}$	RF envelope fall time
T_{fp}	Fall time, power-down
t_{lag}	RF adjust lag time
t_{lead}	RF adjust lead time
T_{pri}	Backscatter-link pulse-repetition interval (T_{pri} = 1/BLF = TRcal/DR)
$t_{r,10-90}$	RF envelope rise time
T_{REPLY}	Time between Interrogator command and Tag backscattered reply for delayed Tag reply
T_{rp}	Rise time, power-up
T_{sp}	Settling time, power-up
T=>R	Tag-to-Interrogator
TN	Tag-notification indicator
TRcal	Tag-to-Interrogator calibration symbol
U	Untraceability indicator
X	XTID indicator (whether a Tag implements an XTID)
XEB	XPC_W2 indicator
XI	XPC_W1 indicator
XPC	Extended protocol control

5.2 Abbreviated terms

AFI	Application family identifier
AM	Amplitude modulation
ASK	Amplitude shift keying
BAP	Battery assisted passive
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
FM0	A type of frequency modulation encoding
LSB	Least significant bit
MAC	Message authentication code
MDID	Mask Designer Identifier
MSB	Most significant bit
PC	Protocol control
PIE	Pulse-interval encoding
ppm	Parts-per-million
PR-ASK	Phase-reversal amplitude shift keying
PSK	Phase shift keying or phase shift keyed
PW	Pulse width

pwd	Password
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
UWC	User word count
XOR	Exclusive OR
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 and Tag reply parameters are underlined. Some flags are also command parameters; a flag used as a command parameter will be underlined. Example: <u>Pointer</u>.
- Command parameters are upper-case and Tag reply parameters are lower-case. Example: <u>Pointer</u> and <u>handle</u>.
- 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, ⊗, refers to XOR.
- The symbol, ||, refers to concatenation.
- The symbol, ≠, refers to not equal.
- 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).
- The numbering base is denoted with a subscript. Binary numbers are denoted with a subscript 2 ($xxxx_2$) and hexadecimal numbers are denoted with a subscript h ($xxxx_h$); the absence of a subscript denotes decimal numbering system.
- The symbol, &, refers to bitwise Boolean and operation.
- The capitalized AND refers to logical (true/false) and operation.
- The capitalized OR refers to logical (true/false) or operation.
- When used in conditions, three or more logical conditions separated the symbol, ;, share the same logical operation. For example, a condition represented as "condition1; condition2; AND condition3" is true if "condition1 AND condition2 AND condition3" is true. The symbol, ;, also represents three or more separate actions. For example, "action1; action2; and action3" represent invoking of action1, action2, and action3.
- The symbols, (and), are used to clarify the order of evaluation for more complex conditions or actions. For example, a condition represented as "(condition1 OR condition 2) AND condition3" is true if either "condition1 AND condition3" or "condition2 AND condition3" is true.

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 FM0 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). An Interrogator may also use *QueryX* command followed by zero or more *QueryY* commands to select a population of Tags based on a value or values in Tag memory.

b. **Inventory.** Identifying individual Tags. An Interrogator begins an inventory round by transmitting a *Query* command or a *QueryX* command followed by zero or more *QueryY* commands using one of four sessions to initialize an inventory round. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the Tag's EPC or TID. 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 – 930 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

Ref	Parameter Name	Description
Int:1d	Frequency Hop Rate (frequency-hopping [FHSS] systems)	In accordance with local regulations
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	MAX(RTcal,10T_{pri}) nominal
Int:6b	Receive-to-Transmit Turn-Around Time	3T_{pri} minimum; 20T_{pri} maximum when Tag is in **reply** and **acknowledged** states; no limit otherwise
Int:6c	Dwell Time or Interrogator Transmit Power-On Ramp	2500 µs, minimum time
Int:6d	Decay Time or Interrogator Transmit Power-Down Ramp	500 µs, maximum time
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

Ref	Parameter Name	Description
Int:12	Scrambling (spread-spectrum systems)	N/A
Int:13	Bit Transmission Order	MSB is transmitted first
Int:14	Wake-up Process	As specified
Int:15	Polarization	Not specified

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

Ref.	Parameter Name	Description
Tag:1	Operating Frequency Range	860 – 930 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** and **acknowledged** states; no limit otherwise
Tag:6b	Receive-to-Transmit Turn-Around Time	MAX(RTcal,$10T_{pri}$) nominal
Tag:6c	Dwell Time or Transmit Power-On Ramp	Receive commands 2500 µs 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

Ref.	Parameter Name	Description
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
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	Varies with R=>T and T=>R link rate and number of bits being read
P:7	Write Transaction Time	20 ms (maximum) after end of *Write* command
P:8	Error Detection	Interrogator-to-Tag: *Select, Challenge, QueryX* and *QueryY* commands: 16-bit CRC *Query* 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, 5-bit CRC, or 16-bit CRC (varies by command) handle: 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)
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, NlogN for Tags with unique EPCs where N=number of tags
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 – 930 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 – 930 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 6-2a*, *Figure 6-2b*, *Figure 6-3a*, *Figure 6-3b*, *Figure 6-5a*, *Figure 6-5b* and *Figure 6-4* 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 °C) and ± 20 ppm over the extended temperature range (−40 °C to +65 °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.

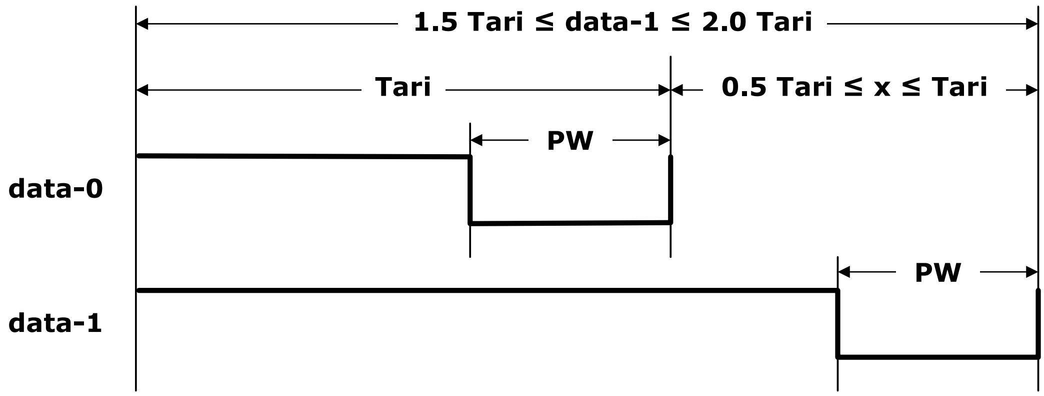
6.3.1.2.2 Modulation

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*. Tari 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 6-2a* and *Figure 6-2b*.

Figure 6-1: PIE symbols



6.3.1.2.4 Tari values

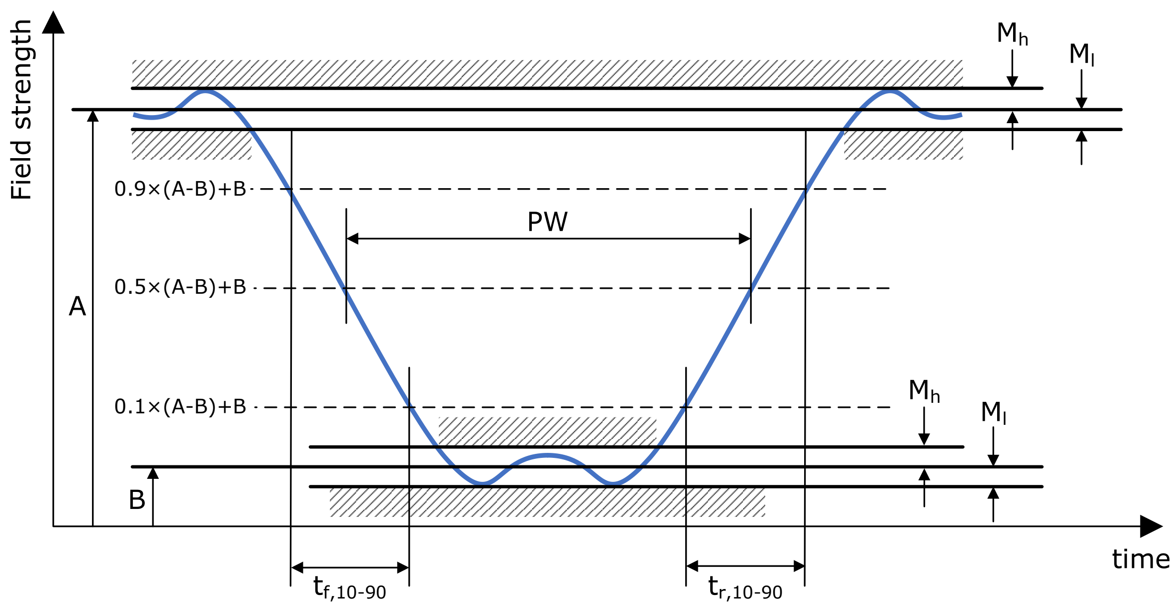
Interrogators shall communicate using Tari values in the range of 6.25 µs to 25 µs. Interrogator compliance shall be evaluated using at least one Tari value between 6.25 µs and 25 µs with at least one value of the parameter *x*. The tolerance on all parameters specified in units of Tari shall be ± 1%. 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 6-2a*, *Figure 6-2b* and *Table 6-5*. The amplitude of the RF envelope is an electric or magnetic field strength, and the field strength A is the nominal field strength of the RF envelope, measured in units of V/m or A/m. Tari is defined in *Figure 6-1*. The pulsewidth is measured at the 50% point on the pulse. Unless permitted by field strength adjustment (*6.3.1.2.9*), an Interrogator shall implement the R=>T RF envelope between modulating pulses in an inventory round with a single nominal field strength (A) and a magnitude between A-M_l and A+M_h (see *Figure 6-2a* and *Figure 6-2b*). An Interrogator with an R=>T RF envelope using PR-ASK modulation shall comply with the ASK modulation mask with B+M_h ≤ 0.1A when modulating the delimiter shown in *Figure 6-2*.

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

a) ASK Modulation



b) PR-ASK Modulation

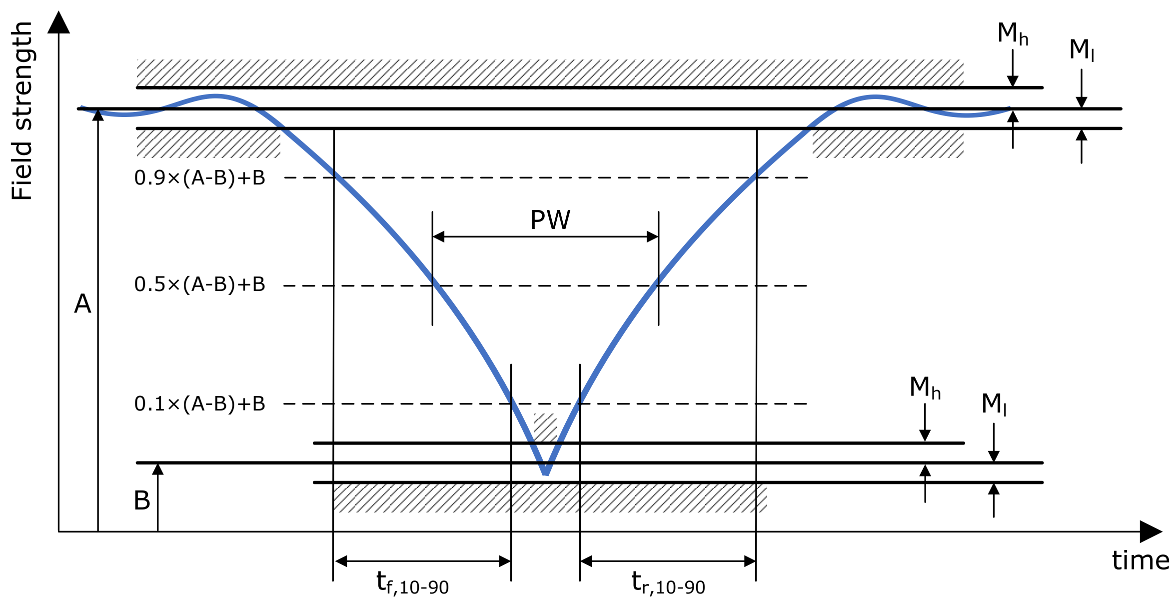


Table 6-5: RF envelope parameters

Tari	Parameter	Symbol	Minimum	Nominal	Maximum	Units
6.25 µs to 25 µs	Modulation Depth	(A–B)/A	80	90	100	%
	Modulation Depth, Adjusted	$(A_{adj}- B_{adj})/A_{adj}$				
	RF Envelope Ripple	$M_h = M_l$	0		0.05(A–B)	V/m or A/m
	RF Envelope Ripple, Adjusted	$M_{adjh} = M_{adjl}$	0		$0.05(A_{adj} - B_{adj})$	
	RF Field Adjust Down	M_{adjd}	0		0.20A	V/m or A/m
	RF Field Adjust Up	M_{adju}	0		$0.25A_{adj}$	
	RF Adjust Lead Time	t_{lead}	500			µs
	RF Adjust Lag Time	t_{lag}	0		0.67RTCal	µs
	RF Envelope Rise Time	$t_{r,10-90}$	0		0.33Tari	µs
	RF Envelope Fall Time	$t_{f,10-90}$	0		0.33Tari	µs
	RF Pulsewidth	PW	MAX(0.265Tari, 2)		0.525Tari	µs

6.3.1.2.6 Interrogator power-up waveform

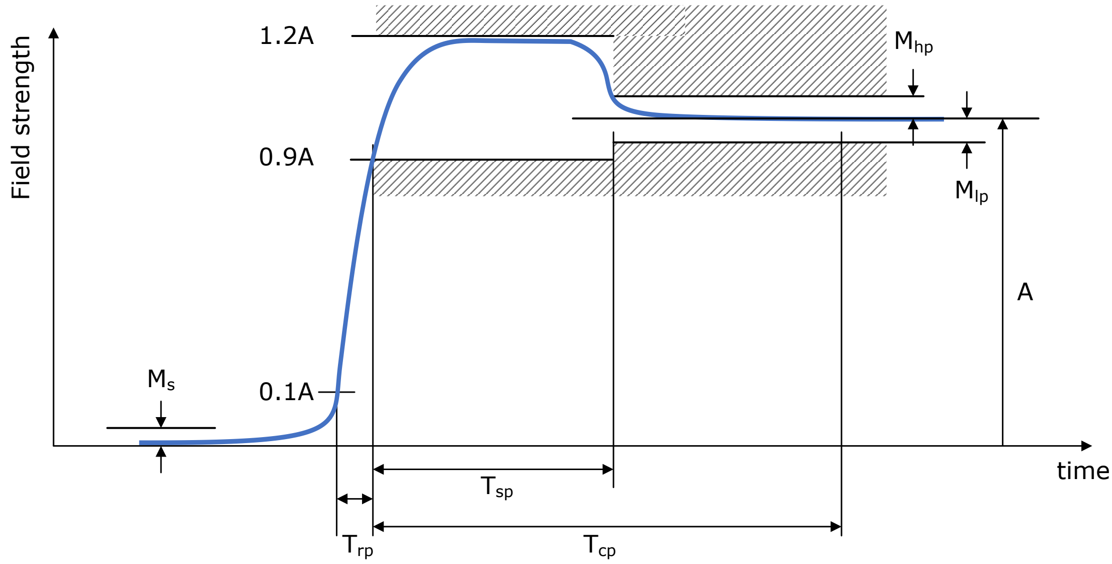
The Interrogator power-up RF envelope shall comply with *Figure 6-3a* and *Table 6-6*:

- Once the carrier level has risen above M_s, the power-up envelope shall monotonically increase to 0.1A.
- During the T_{rp} interval the power-up envelope shall increase from 0.1A to 0.9A, and the increasing envelope may momentarily decrease at a rate ≤ 1%/10 µsec with a maximum decrease of 2.5%.
- During the T_{sp} interval the power-up envelope shall not fall below 0.9A as in *Figure 6-3a*. Within the constraints of *Figure 6-3a* and *Table 6-6* this protocol recommends the Interrogator maximizes the time duration and maximizes the field strength to a constant value during the T_{sp} interval. An Interrogator shall meet the frequency-accuracy requirement specified in *6.3.1.2.1* by the end of the T_{sp} interval.
- An Interrogator shall not issue commands before the minimum time of the T_{cp} interval in *Table 6-6*. An interrogator compliant to prior versions of this protocol may issue a command prior to the minimum T_{cp} interval; if so, a Tag compliant to this protocol may not participate in an inventory round.

Interrogators shall not begin field strength adjust (see *6.3.1.2.9*) before the minimum time of the T_{cp} interval in *Table 6-6*. Interrogators may begin field strength adjust after the minimum time of the T_{cp} interval by decreasing the field strength (adjust down), dwelling on the adjusted field strength, and issuing a *Select*, *Challenge*, *Query*, or *QueryX* command.

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

a) power-up RF envelope



b) power-down RF envelope

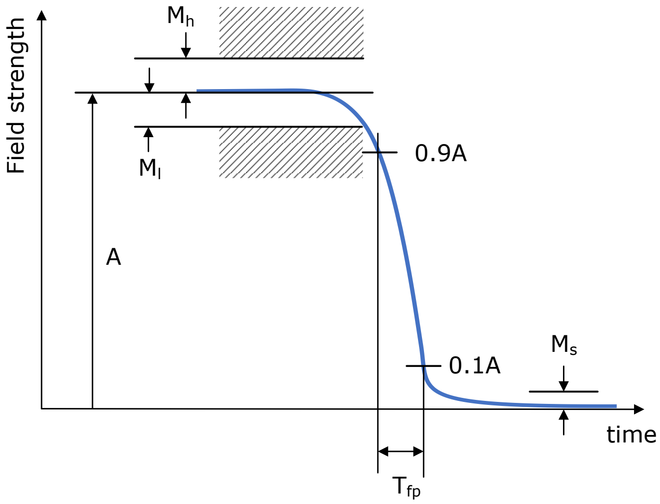


Table 6-6: Interrogator power-up waveform parameters

Parameter	Definition	Minimum	Nominal	Maximum	Units
M_s	RF signal level when OFF			0.01A	V/m or A/m
T_{rp}	Rise time, power-up	1		250	µs
T_{sp}	Settling time, power-up			1000	µs
T_{cp}	Time before command, power-up	2500			µs
M_{lp}	RF undershoot, power-up			0.05A	V/m or A/m
M_{hp}	RF overshoot, power-up			0.05A	V/m or A/m

6.3.1.2.7 Interrogator power-down waveform

The Interrogator power-down RF envelope shall comply with *Figure 6-3b* and *Table 6-7*. Except as permitted by modulation and by field-strength adjust transmitted by an Interrogator, once the carrier level has fallen below the A-M_l level, the Interrogator shall decrease the RF envelope monotonically from 0.9A until the power-off limit M_s. Once powered off, an Interrogator shall remain powered off for a least 1ms before powering up again.

Table 6-7: Interrogator power-down waveform parameters

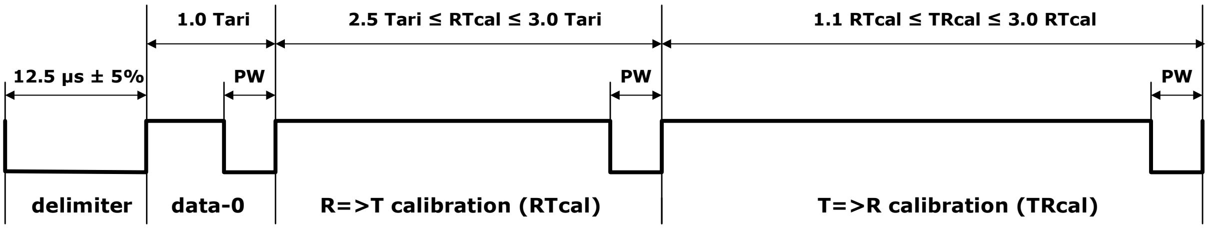
Parameter	Definition	Minimum	Nominal	Maximum	Units
T_{fp}	Fall time, power-down			500	µs
M_s	see *Table 6-6*				
M_l	see *Table 6-5*				
M_h	see *Table 6-5*				

6.3.1.2.8 R=>T preamble and frame-sync

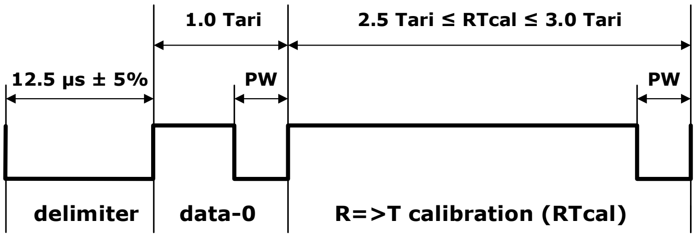
An Interrogator shall begin all R=>T signaling with either a preamble or a frame-sync, both of which are shown in *Figure 6-4*. An interrogator shall precede a *Query* command (see *6.3.2.12.2.1*) and a *QueryX* command (see *6.3.2.12.2.2*) with a preamble. A preamble-*Query* or preamble-*QueryX* begins an inventory round. All other signaling shall begin with a frame-sync. The tolerance on all parameters specified in units of Tari shall be ± 1%. PW shall be as specified in *Table 6-5*. The RF envelope shall be as specified in *Figure 6-2a* and *Figure 6-2b*.

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

a) **R=>T Preamble**



b) **R=>T Frame-Sync**



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. The rise time, fall time, and 12.5 µs pulse width delimiter shall comply with the R=>T RF envelope using the Tari defined after the delimiter.

- **RTcal:** An Interrogator shall set RTcal equal to the length of a data-0 symbol plus the length of a data-1 symbol. A Tag shall measure the length of RTcal and compute *pivot* = RTcal / 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 FM0 data rate or the frequency of its Miller subcarrier) using the TRcal and divide ratio (DR) in the preamble and payload, respectively, of a *Query* or *QueryX* command that begins 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):

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

$$1.1 \times \text{RTcal} \le \text{TRcal} \le 3 \times \text{RTcal} \quad (2)$$

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 R=>T Field strength adjust

Interrogators may reduce (adjust down) the nominal field strength by M_{adjd} before the start delimiter of a *Select*, *Challenge*, *Query*, or *QueryX* command as in *Figure 6-5a* and *Table 6-5*. If an Interrogator adjusts the nominal field strength down, the Interrogator shall monotonically decrease the field strength from $A\text{-}M_l$ to $A_{adj}+M_{adjh}$, and then the field strength shall remain within A_{adj} - M_{adjl} to A_{adj} + M_{adjh} for the minimum t_{lead} time before modulating the delimiter. If an Interrogator adjusts the nominal field strength down, the Interrogator shall use the adjusted nominal field strength (A_{adj}) and other adjusted envelope parameters (B_{adj}, M_{adjh}, M_{adjl}) in *Table 6-5* to modulate the command and modulate the frame-sync or preamble that precedes the command.

Interrogators may increase (adjust up) the field strength by M_{adju} after the last transmitted bit of a

Query, *QueryX* with Init=1_2, or *QueryY* with Init=1_2 command as in Figure 6-5b and Table 6-5. If an Interrogator adjusts the field strength up after the last transmitted bit, the Interrogator shall monotonically increase the field strength from the end of the pulsewidth to A-M_l within maximum t_{lag} time. An Interrogator may adjust up, adjust down, or use different values for M_{adjd} and M_{adju}.

Tags shall communicate with Interrogators that adjust the field strength of the R=>T envelope. Using the parameters in Table 6-8 and an RF envelope sequence of power-up, field strength adjust down, a *QueryX* command, field strength adjust up, and then *ACK* command, then the minimum nominal field strength for a Tag to complete a reply to an *ACK* shall be less than 1.25 times the maximum nominal field strength with no reply to a *QueryX.*

The minimum nominal field strength for a complete reply from a command includes all backscatter data to complete a valid reply. The maximum nominal field strength from a command with no reply is the absence of any (partial or complete) backscatter data after the command.

An Interrogator may be unable to inventory all Tags when using field strength adjust for a *Query* command. Tags that are compliant to prior versions of this protocol may not tolerate the field strength adjustment and consequently will not be inventoried.

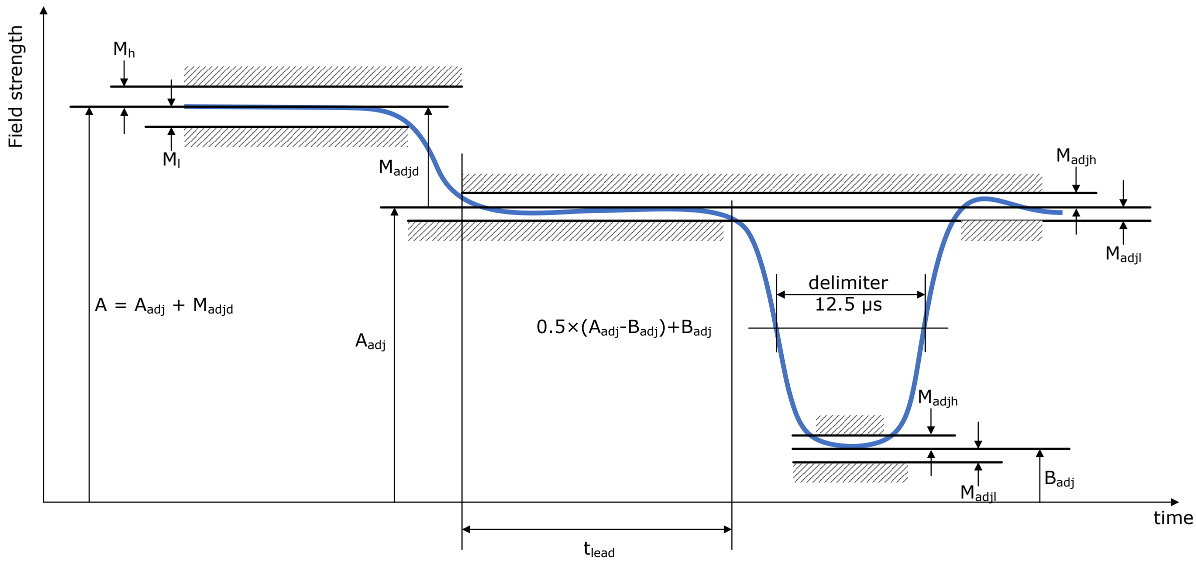
Table 6-8: Conformance test parameters for Tag's field strength adjust for *QueryX* command

Parameter or memory	Value	Additional details
Modulation	DSB-ASK	
Modulation Depth	90%	
Tari	6.25 µs	
PIE ratio	1.5	
TRcal	33 µs	TRcal=33 µs for BLF=640kHz
Session	00_2	Session S0
Action	000_2	S0 **inventoried** → A
SelType	011_2	always matching
Flx	0_2	no flexible filter
ReplyCRC	0_2	reply with RN16
AckData	01_2	reply with EPC to *ACK*
DBLF	001_2	BLF=640kHz
DR	1_2	DR=64/3 for BLF=640kHz
M	01_2	Miller = 2 encoding
TRext	0_2	No extended preamble
Sel	00_2	All tags
Target	0_2	Target A
Q	000_2	Q=0
EPC value	$00...00_h$	All zeros, 96-bit length (6 words)

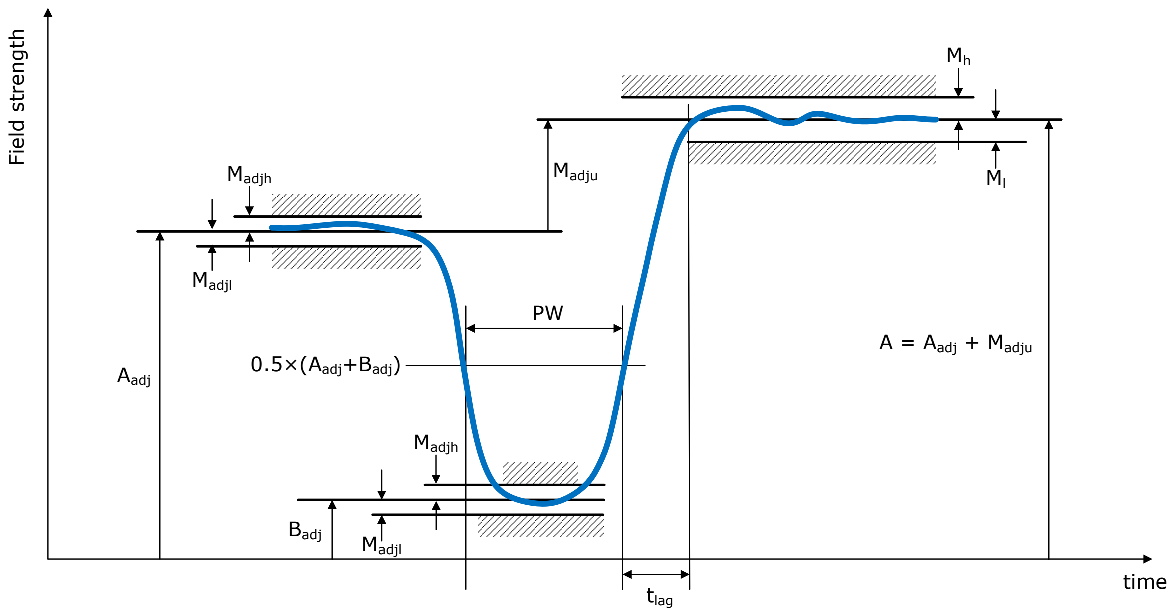
Note 1: These are conformance test parameters. Interrogator may use other parameters with field strength adjust.

Figure 6-5: RF envelope for field strength adjust

a) Field Adjust before *Query*/*QueryX*



b) Field Ajust after *Query*/*QueryX*/*QueryY*



6.3.1.2.10 Frequency-hopping spread-spectrum waveform and channelization

When an Interrogator uses frequency-hopping spread spectrum (FHSS) signaling, the Interrogator's RF envelope shall comply with the power-up and power-down waveforms in *Figure 6-3a*, *Figure 6-3b*, *Table 6-6* and *Table 6-7*. The maximum time between frequency hops and the minimum RF-off time during a hop shall meet local regulatory requirements.

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 (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*.

- **Multiple-Interrogator Transmit Mask:** For an Interrogator modulating an R=>T inventory command sequence in channel *R*, and any other channel *S*≠*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. 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 modulating an R=>T inventory command sequence shall not exceed the specified values:
  - $|n| = 1$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -30$ 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/Tari 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.

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

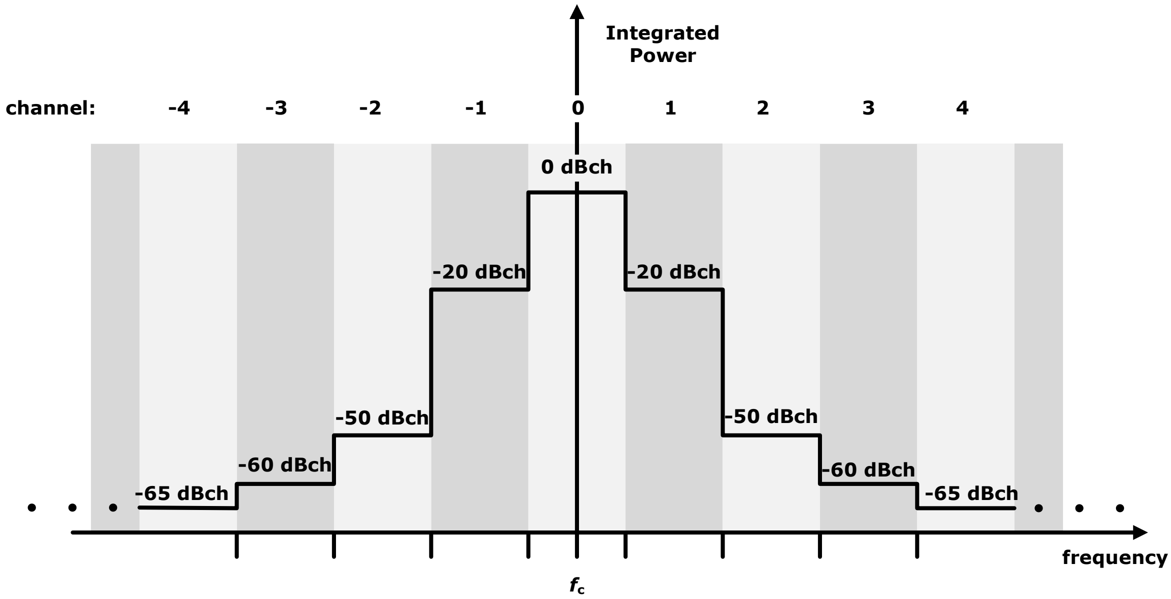
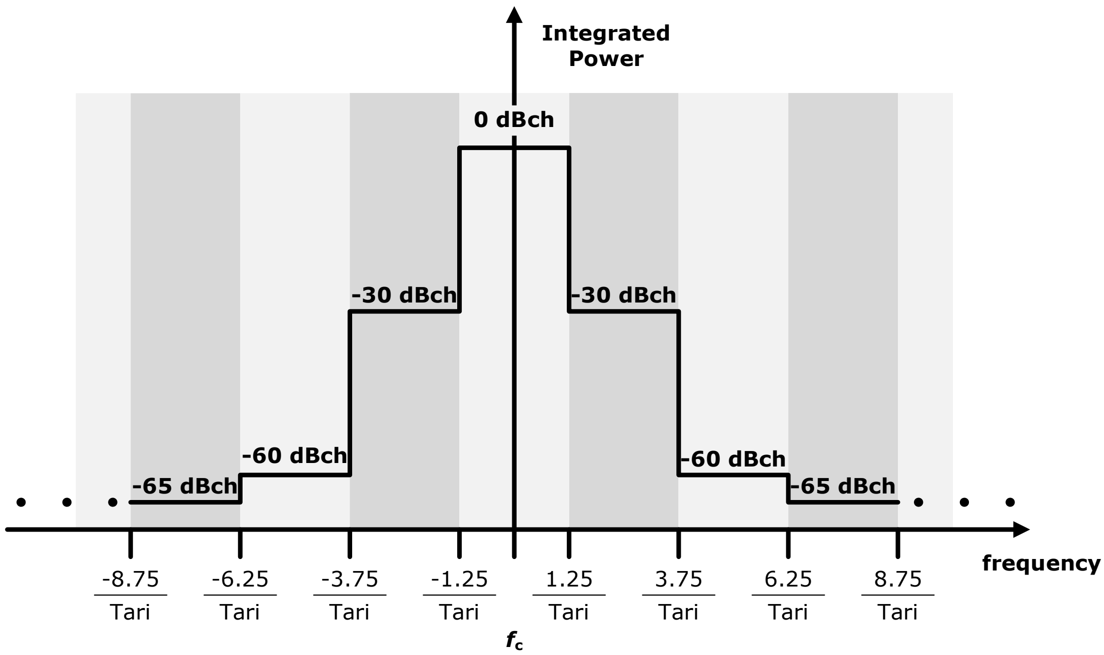


Figure 6-7: Transmit mask for dense-Interrogator environments



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* or *QueryX* command that begins 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 FM0 baseband or Miller modulation of a subcarrier at the data rate. The Interrogator specifies the encoding type.

6.3.1.3.2.1 FM0 baseband

Figure 6-8 shows basis functions and a state diagram for generating FM0 (bi-phase space) encoding. FM0 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 FM0 basis functions that are transmitted. The state labels, S_1–S_4, indicate four possible FM0-encoded symbols, represented by the two phases of each of the FM0 basis functions. The state labels also represent the FM0 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 FM0 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 an extra data-1 bit, known as a "dummy" 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* or *QueryX* that begins 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_2 regardless of the TRext value specified in the *Query* (see *6.3.2.12.2.1*) or in the *QueryX* (see *6.3.2.12.2.2*)). An FM0 violation (i.e. a phase inversion should have occurred but did not) is indicated by "v" in *Figure 6-11*.

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

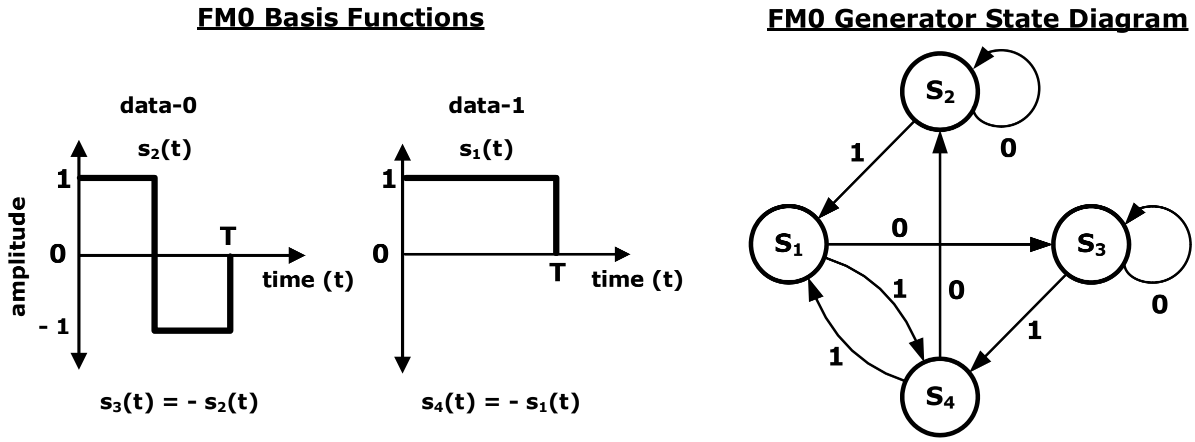


Figure 6-9: FM0 symbols and sequences

FM0 Symbols **FM0 Sequences**

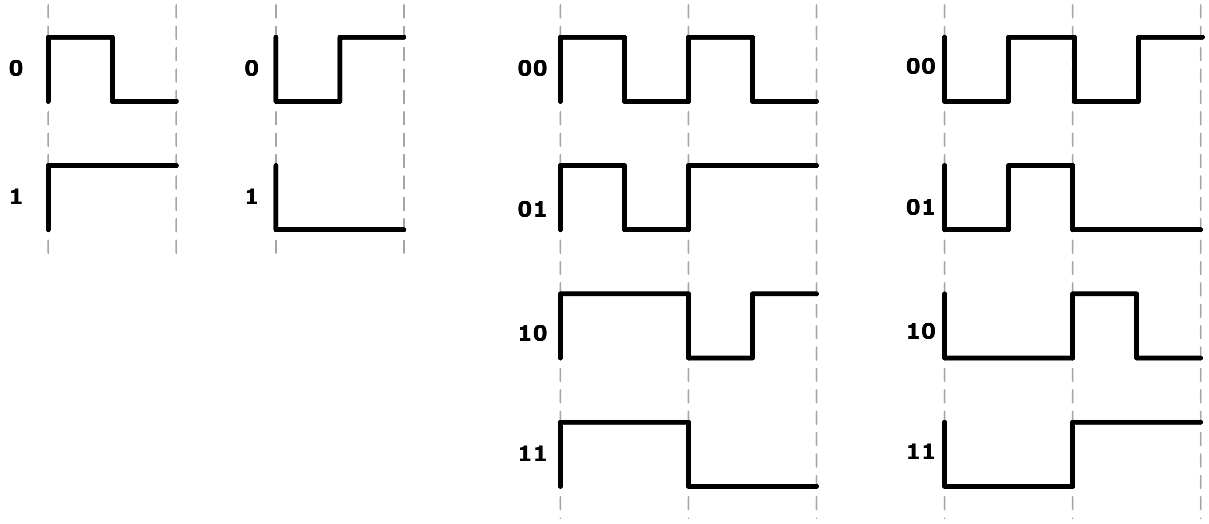


Figure 6-10: Terminating FM0 transmissions

FM0 End-of-Signaling

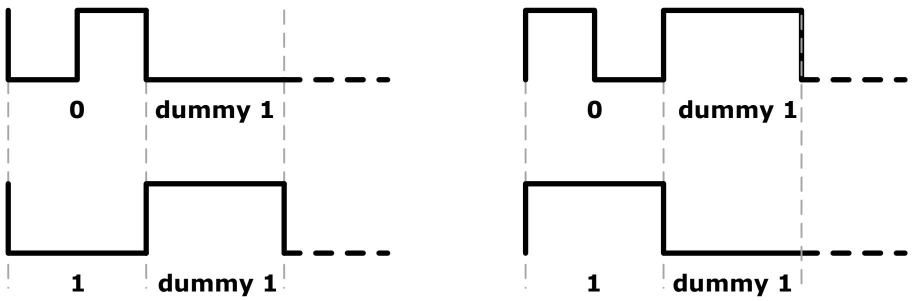
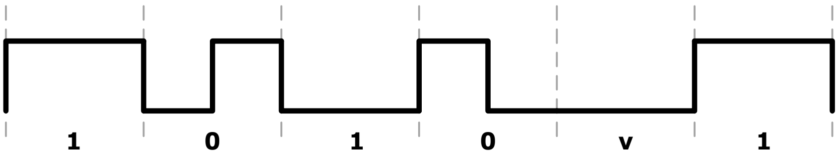
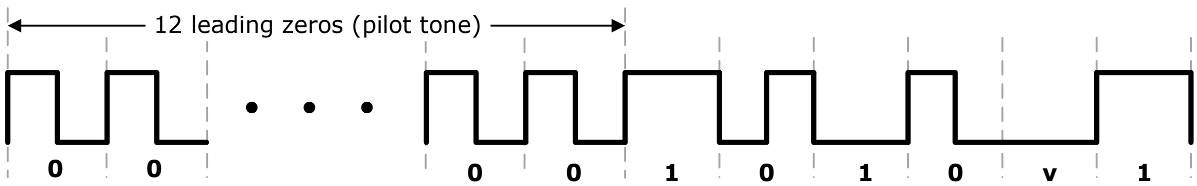


Figure 6-11: FM0 T=>R preamble

FM0 Preamble (TRext=0)



FM0 Extended Preamble (TRext=1) with Pilot Tone



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

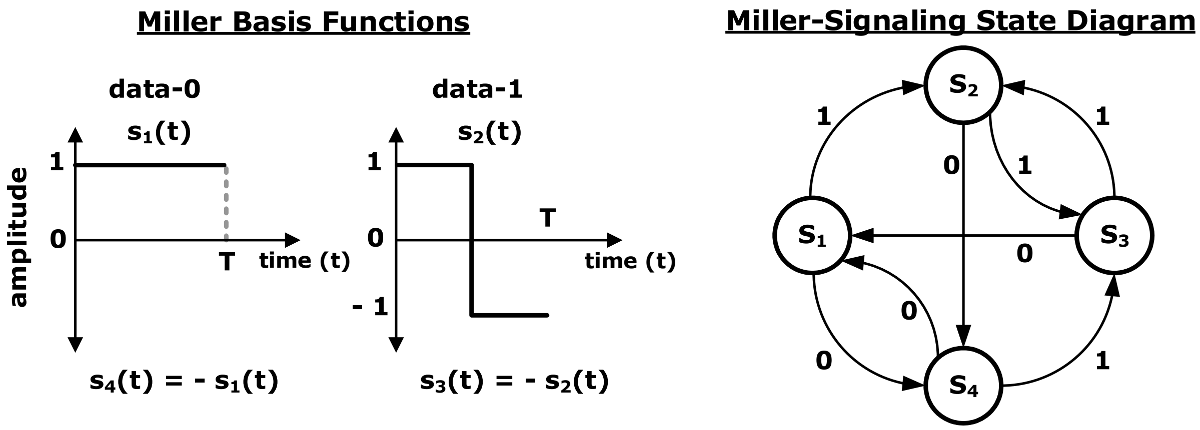


Figure 6-13 shows Miller-modulated subcarrier sequences; the Miller sequence shall contain exactly two, four, or eight subcarrier cycles per bit, depending on the M value specified in the *Query* or *QueryX* command that began the inventory round (see *Table 6-10*). The duty cycle of a data-0 or data-1 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 an extra data-1 bit, known as a "dummy" 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* or *QueryX* that began 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_2 regardless of the TRext value specified in the *Query* (see *6.3.2.12.2.1*) or in the *QueryX* (see *6.3.2.12.2.2*).

Figure 6-13: Subcarrier sequences

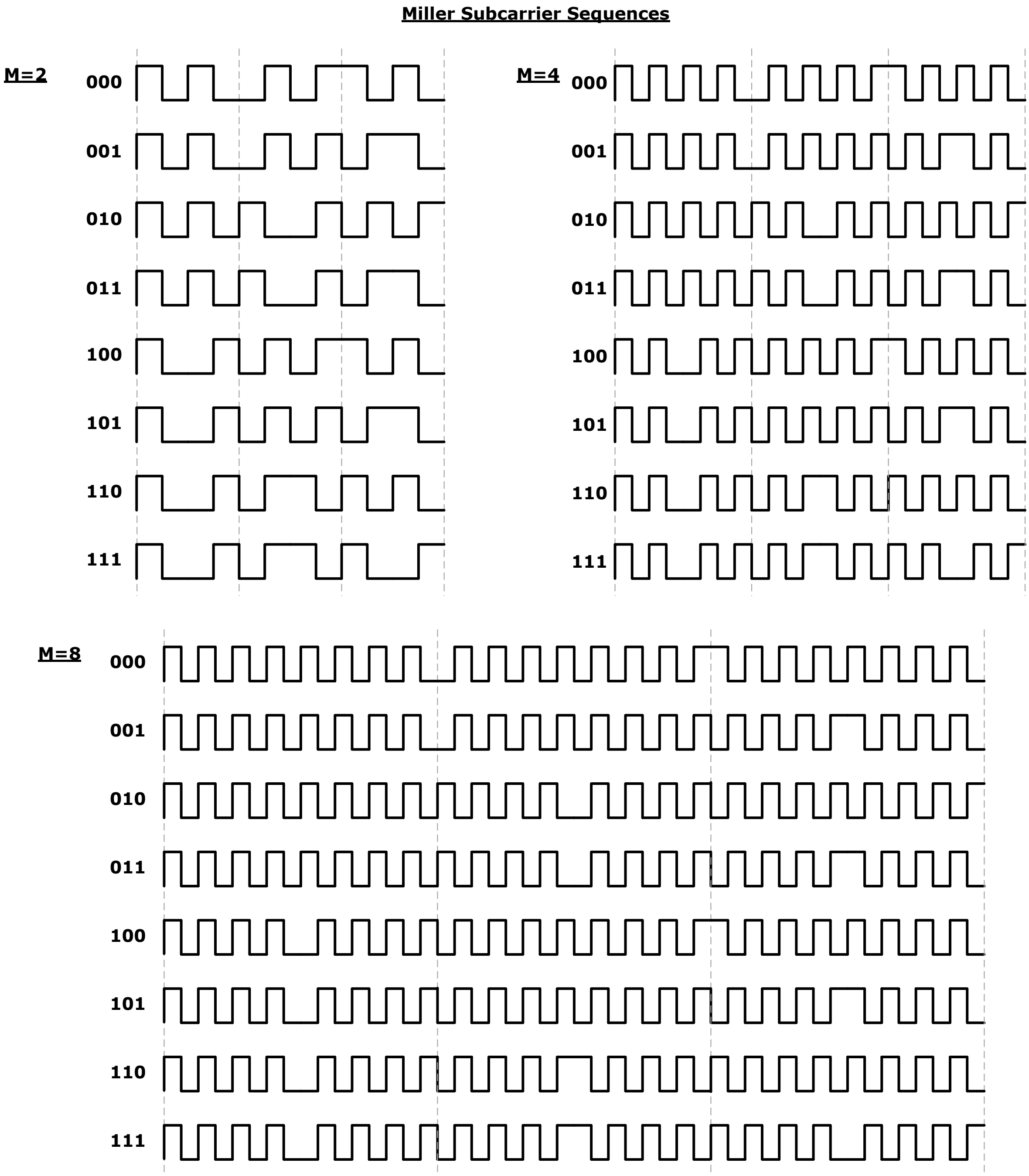


Figure 6-14: Terminating subcarrier transmissions

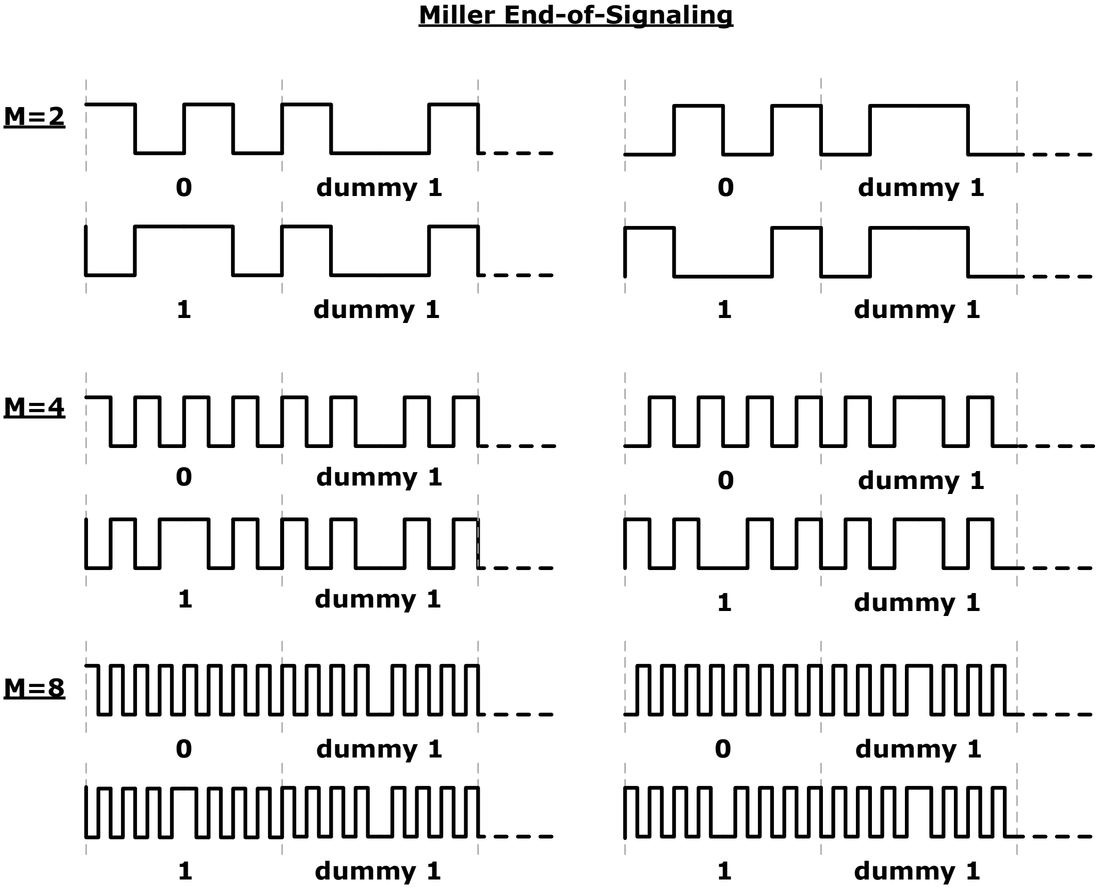
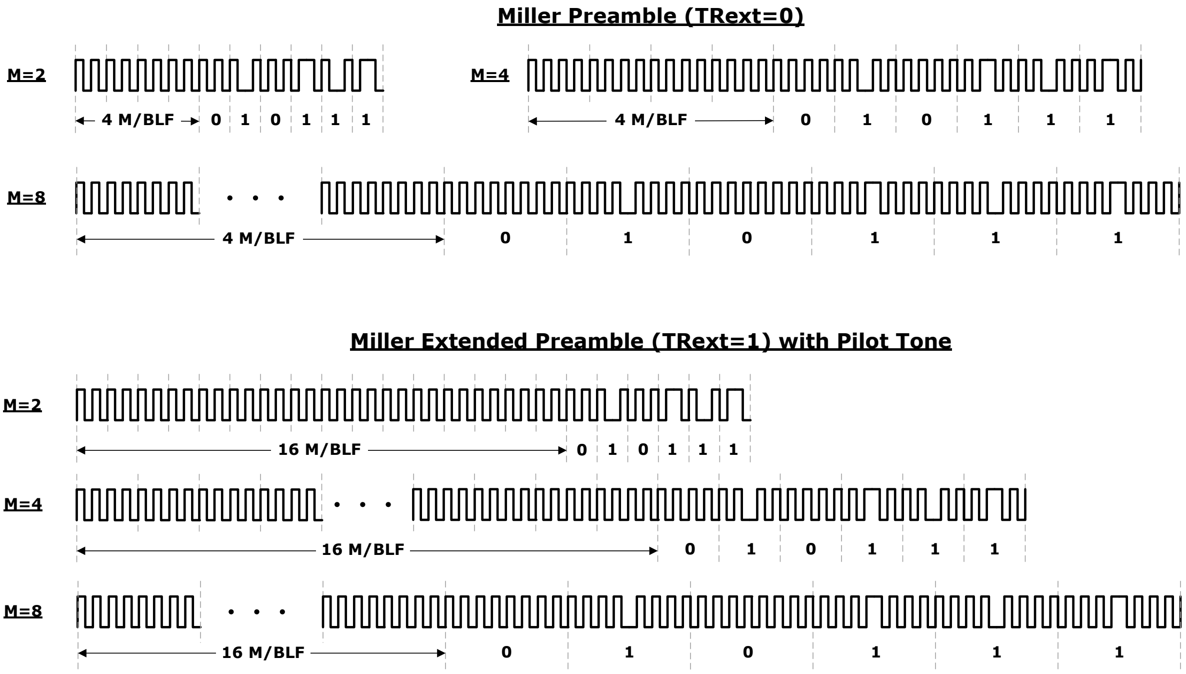


Figure 6-15: Subcarrier T=>R preamble



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 6.25 µs to 25 µs, over all parameters allowed by 6.3.1.2.3.

For inventory rounds beginning with a *Query*, Tags shall support the required T=>R backscatter link frequencies and tolerances (for *Query*) specified in Table 6-9 and the T=>R data rates specified in Table 6-10. For inventory rounds beginning with a *QueryX*, Tags shall support required T=>R backscatter link frequencies and tolerances (for *QueryX*) 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* or *QueryX* command that begins an inventory round specifies DR in Table 6-9 and M in Table 6-10, and the *QueryX* command additionally specifies DBLF. The preamble that precedes the *Query* or *QueryX* specifies TRcal. BLF is computed using Eq. (1). These parameters define the backscatter frequency, modulation type (FM0 or Miller), and T=>R data rate for the round for *Query* and *QueryX* (see also 6.3.1.2.8).

A Tag shall support the required T=>R backscatter link frequencies for a *QueryX* by using the DBLF parameter ("digital BLF") and/or by using the TRcal value and the DR parameter ("analog BLF"):

- Tags may support T=>R backscatter link frequencies specified in Table 6-9 that are optional for inventory rounds beginning with a *QueryX*.
- If a Tag does not support an optional *QueryX* T=>R backscatter link frequency, then the Tag shall not participate in the inventory round (see 6.3.2.12.2.2).
- A Tag that supports a T=>R backscatter link frequency using analog BLF may ignore DBLF values.

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

	For *QueryX* and *Query*		For *QueryX*			For *Query*			Frequency variation during backscatter for *QueryX* and *Query*
DR: Divide Ratio	TRcal[1] (µs ± 1%)	BLF (kHz)	Required by Tag for *QueryX*	DBLF[4]: Digital BLF	FrT[2] ext. temp	Required by Tag for *Query*	FrT[3] nom. temp	FrT[2] ext. temp	
64/3	33.3	640	yes	001_2	± 7%	yes	± 15%	± 15%	± 2.5%
	33.3 < TRcal < 50.0	427 < BLF < 640	optional	101_2	± 7%	yes	± 22%	± 22%	± 2.5%
	50.0	427	yes	010_2	± 7%	yes	± 22%	± 22%	± 2.5%
	50.0 < TRcal < 66.7	320 < BLF < 427	optional	101_2	± 7%	yes	± 22%	± 22%	± 2.5%
	66.7	320	yes	011_2	± 7%	yes	± 10%	± 15%	± 2.5%
	66.7 < TRcal < 83.3	256 < BLF < 320	optional	101_2	± 7%	yes	± 12%	± 15%	± 2.5%
	83.3	256	yes	100_2	± 7%	yes	± 10%	± 10%	± 2.5%
	83.3 < TRcal < 133.3	160 < BLF < 256	optional	101_2	± 7%	yes	± 10%	± 12%	± 2.5%
	133.3	160	yes	111_2	± 7%	yes	± 7%	± 7%	± 2.5%
	133.3 < TRcal ≤ 200	107 ≤ BLF < 160	optional	110_2	± 7%	yes	± 7%	± 7%	± 2.5%
	200 < TRcal ≤ 225	95 ≤ BLF < 107	optional	110_2	± 5%	yes	± 5%	± 5%	± 2.5%
8	17.2 ≤ TRcal < 18.74	427 < BLF ≤ 465	optional	101_2	± 7%	yes	± 19%	± 19%	± 2.5%
	18.74	427	yes	010_2	± 7%	yes	± 19%	± 19%	± 2.5%
	18.74 < TRcal < 25	320 < BLF < 427	optional	101_2	± 7%	yes	± 19%	± 19%	± 2.5%
	25	320	yes	011_2	± 7%	yes	± 10%	± 15%	± 2.5%
	25 < TRcal < 31.25	256 < BLF < 320	optional	101_2	± 7%	yes	± 12%	± 15%	± 2.5%
	31.25	256	yes	100_2	± 7%	yes	± 10%	± 10%	± 2.5%
	31.25 < TRcal < 50	160 < BLF < 256	optional	101_2	± 7%	yes	± 10%	± 10%	± 2.5%
	50	160	yes	111_2	± 7%	yes	± 7%	± 7%	± 2.5%
	50 < TRcal ≤ 75	107 ≤ BLF < 160	optional	110_2	± 7%	yes	± 7%	± 7%	± 2.5%
	75 < TRcal ≤ 200	40 ≤ BLF < 107	optional	110_2	± 4%	yes	± 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.

Note 2: Frequency Tolerance over extended temperature range

Note 3: Frequency Tolerance over nominal temperature range

Note 4: see DBLF parameter in *6.3.2.12.2.2*

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

M: Number of subcarrier cycles per symbol	Modulation type	Data rate (kbps)
1	FM0 baseband	BLF
2	Miller subcarrier	BLF/2
4	Miller subcarrier	BLF/4
8	Miller subcarrier	BLF/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 to exceed the minimum time before command, power-up (i.e. by minimum T_{cp}), specified in *Table 6-6*.

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
3. 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 a checksum residue using the CRC-16 parameters in *Table 6-11*. The Tag or Interrogator shall then take the ones-complement of the computed residue 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 or a CRC-5 to those replies that use a CRC-5. 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 6-29*.

Table 6-11: CRC-16 parameters (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 parameters (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 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 6-29 specifies the reply type that a Tag uses for each Interrogator command. Figure 6-18 also illustrates timing for *QueryX* and for *QueryX* followed by *QueryY* that may start a T_8 timeout as defined in 6.3.1.6.5.

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:

a. **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.

b. **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 reply shall meet the T_5 limits in Table 6-16. An error reply uses header=1_2 (see Annex I) to differentiate it from a successful reply.

c. **The Tag fails to execute the command:** If the Interrogator does not observe a Tag reply within $T_{5(max)}$ then the Tag may have 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.

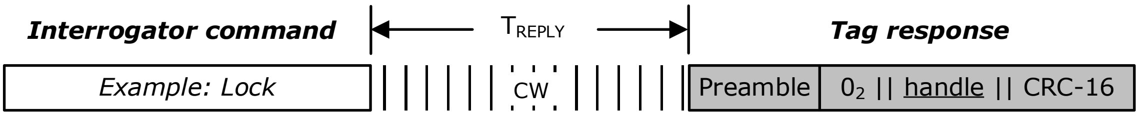
A Tag shall ignore Interrogator commands while processing a prior command that specified a *delayed* reply. If an Interrogator transmits a command while the Tag is processing then the Tag may continue with its processing or, in environments with limited power availability, may undergo a power-on reset.

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 TRext=1_2 regardless of the TRext value in the *Query* or *QueryX* that began 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	handle	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 *6.3.2.12.3.12*), 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 first reply includes length then all subsequent replies shall include length.

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_2, header=0_2, 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_2 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_2, header=1_2, 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_2 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 may have 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_2 means the Tag is still processing; done=1_2 means the Tag has finished processing. Header indicates whether computation was successful or not: header=0_2 means the response includes a result; header=1_2 means response includes 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_2 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_2 regardless of the TRext value in the *Query* or *QueryX* that began the inventory round).

6.3.1.6.4 ResponseBuffer

A Tag that implements a *Challenge* command, or supports SenRep=0_2 in an access command 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 6-19*), indicates whether the Tag has a stored response (result or error code) in its ResponseBuffer. **C**=1_2 indicates that the ResponseBuffer contains data; **C**=0_2 indicates that the ResponseBuffer is empty.
- A Tag shall set **C**=1_2 after storing a response (result or error code) in its ResponseBuffer.
- A Tag shall set **C**=0_2 upon either (1) receiving an access command containing SenRep=0_2 (c.f. *6.3.2.12.3*), (2) receiving a *Challenge* command, or (3) when specified by a cryptographic suite. A Tag may set **C**=0_2 upon receiving an access command containing SenRep=1_2, or when the Tag transitions from the **acknowledged**, **open**, or **secured** states.
- The **C** flag shall be selectable using a *Select*, *QueryX* or *QueryY* 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	result or error code	handle	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 length of result or error code followed by even parity bit	result or error code	handle	CRC-16

- If an access command with SenRep=0_2 or a *Challenge* command specifies IncRepLen=0_2 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_2 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 *6.3.2.12.3.16*.
- 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 6-21*). When **C**=1_2 then a Tag shall maintain the data in its ResponseBuffer. When **C** is or becomes 0_2 then a Tag shall deallocate its ResponseBuffer.

The Tag memory for the ResponseBuffer is Tag-manufacturer defined. Also, because a ResponseBuffer is not writable by an Interrogator, this protocol does not specify a mechanism for an Interrogator to write to it.

Figure 6-17: ResponseBuffer data storage

length included

15 length bits	1 parity bit	maximum 2^{15} response bits

00_h $0E_h$ $0F_h$ 10_h

length omitted

maximum 2^{15} response bits

00_h

Table 6-15: Possible *in-process* Tag replies

<table>
<tr><th rowspan="2">Sent or stored reply</th><th rowspan="2">Time for Tag to execute command?</th><th rowspan="2">Omit or include length?</th><th colspan="2">Reply</th></tr>
<tr><th>Tag executed command</th><th>Tag failed to execute command</th></tr>
<tr><td rowspan="4">Sent</td><td rowspan="2">$\leq T_{6(max)}$</td><td>Omit</td><td>done: 1
header: 0
response: result</td><td>done: 1
header: 1
response: error code</td></tr>
<tr><td>Include</td><td>done: 1
header: 0
length: length of sent result
response: result</td><td>done: 1
header: 1
length: length of sent error code
response: error code</td></tr>
<tr><td rowspan="2">Any</td><td>Omit</td><td>Intermediate reply or replies
done: 0
header: 0
response: null
Final reply
done: 1
header: 0
response: result</td><td>Intermediate reply or replies
done: 0
header: 0
response: null
Final reply
done: 1
header: 1
response: error code</td></tr>
<tr><td>Include</td><td>Intermediate reply or replies
done: 0
header: 0
length: 0000_h
response: null
Final reply
done: 1
header: 0
length: length of sent result
response: result</td><td>Intermediate reply or replies
done: 0
header: 0
length: 0000_h
response: null
Final reply
done: 1
header: 1
length: length of sent error code
response: error code</td></tr>
<tr><td rowspan="3">Stored</td><td rowspan="2">$\leq T_{6(max)}$</td><td>Omit</td><td>done: 1
header: 0
response: null</td><td>done: 1
header: 1
response: null</td></tr>
<tr><td>Include</td><td>done: 1
header: 0
length: length of stored result
response: null</td><td>done: 1
header: 1
length: length of stored error code
response: null</td></tr>
<tr><td>Any</td><td>Omit</td><td>Intermediate reply or replies
done: 0
header: 0
response: null
Final reply
done: 1
header: 0
response: null</td><td>Intermediate reply or replies
done: 0
header: 0
response: null
Final reply
done: 1
header: 1
response: null</td></tr>
</table>

Sent or stored reply	Time for Tag to execute command?	Omit or include length?	Reply	
			Tag executed command	Tag failed to execute command
		Include	Intermediate reply or replies done: 0 header: 0 length: 0000_h Response: null Final reply done: 1 header: 0 length: length of stored result response: null	Intermediate reply or replies done: 0 header: 0 length: 0000_h response: null Final reply done: 1 header: 1 length: length of stored error code response: null

6.3.1.6.5 T_8 timeout

A Tag shall process a T_8 timeout as shown in *Figure 6-24* and *Figure 6-25*.

Interrogators and Tags shall implement the *QueryX* (*6.3.2.12.2.2*) and *QueryY* (*6.3.2.12.2.3*) commands. As illustrated in *Figure 6-18*, a *QueryX* with Init=1_2 has similar link timing as a *Query*. An illustration in *Figure 6-18* shows a *QueryX* with Init=0_2, a first *QueryY* with Init=0_2, and a second *QueryY* with Init=1_2. Both the first and second *QueryY* commands are "following" *QueryY* commands since they both follow a *QueryX* or *QueryY* with Init=0_2. Setting the parameter Init=0_2 in *QueryX* or *QueryY* indicates to the Tag to expect a following *QueryY* and instructs the Tag to start a T_8 timeout. If within time $T_{8(max)}$:

- the Tag fails to receive a command, then the Tag shall transition to **ready**;
- the Tag receives a command that is not-exempt (see *6.3.2.12*), then the Tag shall ignore the command and transition to **ready**;
- the Tag receives an exempt command (see *6.3.2.12*), then the Tag shall execute the command.

Figure 6-18: Link timing

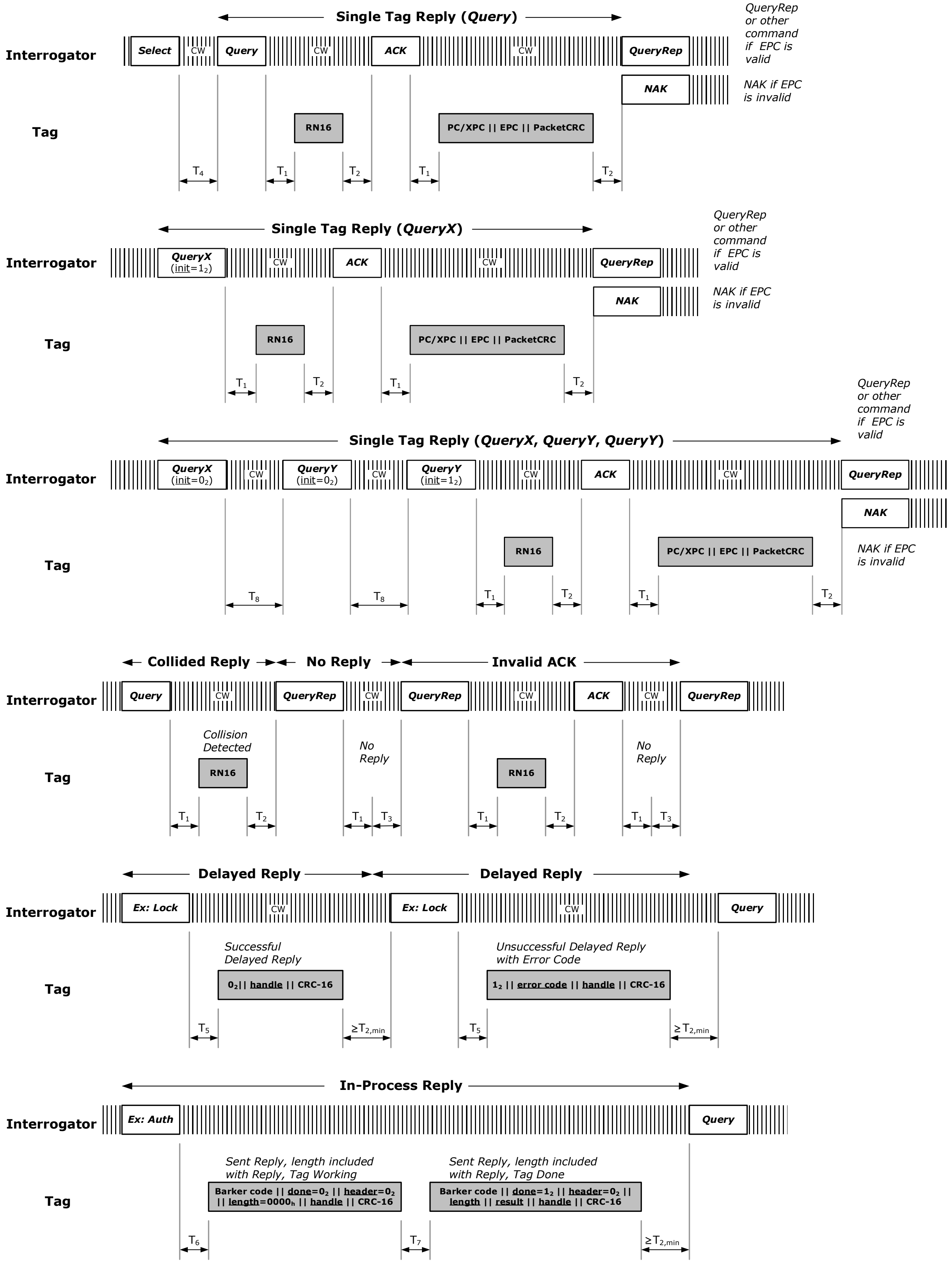


Table 6-16: Link timing parameters

Parameter	Minimum	Nominal	Maximum	Description
T_1	MAX(RTcal,10T_{pri}) × (1 − |FrT|) − 2µs	MAX(RTcal,10T_{pri})	MAX(RTcal,10T_{pri}) × (1 + |FrT|) + 2µs	*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	2.0RTcal			Minimum time between Interrogator commands
T_5	MAX(RTcal,10T_{pri}) × (1 − |FrT|) − 2µs		20ms	*Delayed* 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 *delayed* Tag reply, measured at the Tag's antenna terminals.
T_6	MAX(RTcal,10T_{pri}) × (1 − |FrT|) − 2µs		20ms	*In-process* 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	MAX(250µs, $T_{2(max)}$)		20ms	*In-process* 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
T_8	2.0RTcal	3.0RTcal	4.0RTcal	Interrogator time between the last rising edge of a previous *QueryX*/*QueryY* (Init=0_2) and the falling edge of the delimiter from a following *QueryY*

The following items apply to the requirements specified in *Table 6-16*:

1. T_{pri} denotes either the commanded period of an FM0 symbol or the commanded period of a single subcarrier cycle, as appropriate.
2. The maximum value for T_2 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 T_2 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 T_2 shall be unrestricted. A Tag shall be allowed a tolerance of $20.0T_{pri} \leq T_{2(max)} \leq 32.0T_{pri}$ in determining whether T_2 has expired. "Invalid command" is defined in 6.3.2.12.

3. An Interrogator may transmit a new command prior to interval T_2 (i.e. during a Tag response). In this case the responding Tag may ignore the new command and, in environments with limited power availability, may undergo a power-on reset.
4. FrT is the frequency tolerance specified in *Table 6-9*.
5. T_1+T_3 shall not be less than T_4.
6. T_8 starts after QueryX or QueryY if Init=0_2, otherwise T_8 stops.
7. A Tag shall be allowed a tolerance of 4.0RTcal < $T_{8(max)}$ < 6.0RTcal in determining whether T_8 has expired.

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, 6.3.2.12.2.2, and 6.3.2.12.2.3).

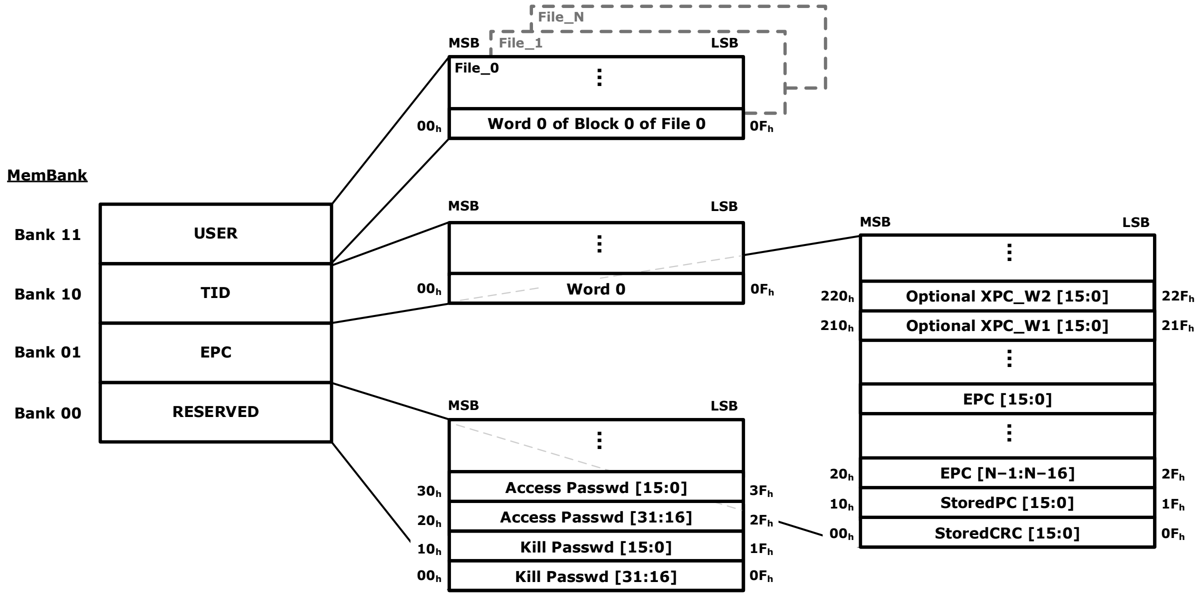
MemBank shall be defined as follows:

00_2	Reserved
01_2	EPC
10_2	TID
11_2	User

Operations that modify memory contents in one logical memory bank shall not modify 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 *6.3.2.12.3.6*) 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 unwritable 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 unwritable 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 *6.3.2.12.3.17*).

If a Tag implements User memory then it partitions each File_N, N≥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 *6.3.2.12.3.10*). 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 unwritable but readable. If blocks within File_N, N>0 are permalocked then these blocks are permanently unwritable but readable by an Interrogator with appropriate read privileges (see *Table 6-25* and *Table 6-26*).

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 *6.3.2.12.3.6*), 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 00_h to $1F_h$, 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 *6.3.2.12.3.5*). 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 20_h to $3F_h$, 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 *6.3.2.12.3.17*, *6.3.2.12.1.1*, *6.3.2.12.2.2*, and *6.3.2.12.2.3*). 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 or TID, 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_{cp} in *Figure 6-3*. 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, and before performing the calculation, the Tag shall:

- assert or deassert **RUM** in the StoredPC (see *Table 6-17*);
- deassert **XI**;
- omit XPC_W1 and XPC_W2 from the calculation.

If the Tag subsequently changes **RUM** then the Tag's StoredCRC may be incorrect until the Interrogator power-cycles the Tag.

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-34*).

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 untraceably hidden and/or truncated) or TID (which may be untraceably hidden), and a PacketCRC to protect the backscattered data stream (see *Table 6-18*). The TID is the traceable part of TID memory starting at address 00_h and ending at the last word of serialization in the TID memory, and if the entire TID memory is untraceably hidden then the Tag shall not backscatter any TID contents in the reply. A Tag shall compute the PacketCRC as specified in *6.3.1.5* over the PC word, optional XPC word(s), and backscattered EPC or backscattered TID, and the Tag 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 or TID 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 6-19* and defined in *Table 6-20*. Note that some bit assignments are different for GS1 EPCglobal (**T**=0_2) versus non-GS1 EPCglobal (**T**=1_2) 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_2 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_2 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_2 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-34*).

- **RUM (Read user memory indicator, bit 15_h):** Bit 15_h indicates that a Tag has memory allocated to File_0 and, if the Interrogator initiated the inventory round using a *QueryX*, that the Tag has encoded data in File_0. A Tag shall compute **RUM** according to *Table 6-17* regardless of the lock or permalock status of EPC memory or the untraceability status of File_0.

  If an Interrogator changes a Tag's User Word Count (UWC) value (see *6.3.2.12.3.3*) or changes the number of words allocated to File_0 memory, then a Tag's **RUM** may be incorrect until the Interrogator power-cycles the Tag. Additionally, **RUM** may change without power cycling; for example, a Tag with memory allocated to File_0 and with UWC=0 will have **RUM**=0_2 after *QueryX* begins initializing an inventory round, but after a *Write* to the StoredPC, then **RUM** may change since the Tag may recompute its StoredCRC.

- **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_2 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_2 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_2 then the Tag shall set these bits to 00_h. If **T**=1_2 then the Tag (if the bits are not writable) or an Interrogator (if the bits are writable) shall set these bits as specified in ISO/IEC 15961.

Table 6-17: Setting the value of **RUM**

Event	Condition	RUM
Executing a *QueryX* or *QueryY*	File_0 does not exist or has no allocated memory words	0
	UWC=0 and Memory allocated to File_0	0
	UWC>0 (Number of words allocated to File_0≥UWC, see *6.3.2.12.3.3*)	1
Executing a *Select*, *Challenge*, or *Query*	File_0 does not exist or has no allocated memory words	0
	Memory allocated to File_0	1
Power-up or computing StoredCRC	File_0 does not exist or has no allocated memory words	0
	Memory allocated to File_0	1

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-34*).

A Tag 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 6-18*. 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. A *QueryX* with AckData=10_2 instructs a Tag to reply to an *ACK* with TID until the end of serialization, and the Tag adjusts the **L** bits in the PacketPC to match the length of the backscattered TID. If **XI**=1_2 but **XEB**=0_2 then a Tag backscatters an XPC_W1 before the EPC or TID, so the Tag shall add one to (i.e. increment) its **L** bits for EPC or add one to the length of the backscattered TID. If both **XI**=1_2 and **XEB**=1_2 then the Tag backscatters both an XPC_W1 and an XPC_W2 before the EPC or TID, so the Tag shall add two to (i.e. double increments) its **L** bits for EPC or add two to the length of the backscattered TID. 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.

The fields that a Tag includes in its reply to an *ACK* (*Table 6-18*) depend on the values of **T**, **C**, **XI**, and **XEB** (see *Table 6-20*); whether the Tag implements an XPC_W1; whether the Tag is truncating its reply (see *6.3.2.12.1.1*, *6.3.2.12.2.2*, and *6.3.2.12.2.3*); the value of Immed (see *6.3.2.12.1.2*); and the value of AckData if *QueryX* began the inventory round. If a Tag has **T**=0_2, **XI**=0_2, 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_2, **XI** =1_2, or both **XI**=1_2 and **XEB**=1_2.

When sending a truncated EPC a Tag substitutes 00000_2 for its PC field — see *Table 6-18*, *6.3.2.12.1.1*, *6.3.2.12.2.2*, and *6.3.2.12.2.3*);.

If a Tag has a response (result or error code) in its ResponseBuffer (i.e. **C**=1_2) and the Interrogator set Immed=1_2 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 6-18*).

Table 6-18: Tag reply to an *ACK* command

T	XI	XEB	Trunc-ation	**C** AND Immed[6]	Tag Backscatter					
					PC	XPC	EPC[1] or TID[5]	CRC	Response	CRC
0	0	0	0	0	If TID, then PacketPC; If EPC and Tag does not implement XPC_W1, then StoredPC(10_h–$1F_h$) If EPC and Tag implements XPC_W1 (see note 2), then StoredPC(10_h–17_h)|| XPC_W1(218_h–$21F_h$)	None	TID or Full EPC	PacketCRC	–	–
0	0	0	0	1	**C**=1_2 so Tag implements XPC_W1 (note 2): StoredPC(10_h–17_h)|| XPC_W1(218_h–$21F_h$)	None	TID or Full EPC	PacketCRC	response	CRC-16
0	0	0	1	0	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
0	0	0	1	1	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	response	CRC-16
0	0	1	All		Disallowed[3]					
0	1	0	0	0	PacketPC	XPC_W1	TID or Full EPC	PacketCRC	–	–
0	1	0	0	1	PacketPC	XPC_W1	TID or Full EPC	PacketCRC	response	CRC-16
0	1	0	1	0	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
0	1	0	1	1	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	response	CRC-16
0	1	1	0	0	PacketPC	XPC_W1, XPC_W2	TID or Full EPC	PacketCRC	–	–
0	1	1	0	1	PacketPC	XPC_W1, XPC_W2	TID or Full EPC	PacketCRC	response	CRC-16
0	1	1	1	0	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
0	1	1	1	1	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	response	CRC-16
1	0	0	0	0 (**C**=0_2)	PacketPC if TID; StoredPC(10_h–$1F_h$) if Full EPC	None	TID or Full EPC	PacketCRC	–	–
1	0	0	0	0 (**C**=1_2)	Disallowed[4]					
1	0	0	0	1	Disallowed[4]					
1	0	0	1	0 (**C**=0_2)	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
1	0	0	1	0 (**C**=1_2)	Disallowed[4]					
1	0	0	1	1	Disallowed[4]					
1	0	1	All		Disallowed[3]					
1	1	0	0	0	PacketPC	XPC_W1	TID or Full EPC	PacketCRC	–	–
1	1	0	0	1	PacketPC	XPC_W1	TID or Full EPC	PacketCRC	response	CRC-16
1	1	0	1	0	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
1	1	0	1	1	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	response	CRC-16
1	1	1	0	0	PacketPC	XPC_W1, XPC_W2	TID or Full EPC	PacketCRC	–	–

T	XI	XEB	Trunc -ation	**C** AND Immed[6]	Tag Backscatter					
					PC	XPC	EPC[1] or TID[5]	CRC	Response	CRC
1	1	1	0	1	PacketPC	XPC_W1, XPC_W2	TID or Full EPC	PacketCRC	response	CRC-16
1	1	1	1	0	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	–	–
1	1	1	1	1	PacketPC if TID; 00000_2 if truncated EPC	None	TID or Truncated EPC	PacketCRC	response	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 command specifying truncation (see *6.3.2.12.1.1*, *6.3.2.12.2.2*, and *6.3.2.12.2.3*).

Note 2: If a Tag has **T**=0_2, **XI**=0_2, implements an XPC_W1, and is not truncating then the Tag substitutes EPC memory bits 218_h–$21F_h$ for EPC memory bits 18_h–$1F_h$ in its reply to an *ACK*.

Note 3: If **T**=0_2 then **XI** 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 chooses whether a Tag implements (i) or (ii). If **T**=1_2 then **XI** is the logical OR of the entirety of XPC_W1 (210_h–$21F_h$). Because **XEB** is the MSB (210_h) of XPC_W1, if **XEB**=1_2 then **XI**=1_2 regardless of the **T** value.

Note 4: If **T**=1_2 then **XI** is the logical OR of the entirety of XPC_W1 (210_h–$21F_h$), so if **C**=1_2 then **XI**=1_2.

Note 5: If a *Query* or *QueryX* with AckData=01_2 begins an inventory round then full or truncated EPC; if a *QueryX* with AckData=10_2 begins the inventory round then TID is the traceable part of TID memory starting at address 00_h and ending after serialization.

Note 6: see *6.3.2.12.1.2*.

Table 6-19: 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	$1A_h$	$1B_h$	$1C_h$	$1D_h$	$1E_h$	$1F_h$
GS1 EPCglobal	**L4**	**L3**	**L2**	**L1**	**L0**	**RUM**	**XI**	**T**=0_2	RFU=00_h							
Non-GS1 EPCglobal	**L4**	**L3**	**L2**	**L1**	**L0**	**RUM**	**XI**	**T**=1_2	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	RFU	RFU	**SA**	**SS**	**FS**	**SN**	**B**	**C**	**SLI**	**TN**	**U**	**K**	**NR**	**H**
Non-GS1 EPCglobal	**XEB**	RFU	**MIIM**	RFU	**SA**	**SS**	**FS**	**SN**	**B**	**C**	**SLI**	**TN**	**U**	**K**	**NR**	**H**

Table 6-20: 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	**RUM**	Fixed or computed	Read User Memory indicator	see *Table 6-17*

Hex	Name	How Set?[1]	Descriptor	Settings
16	**XI**	Computed	XPC_W1 indicator	0: either (i) Tag has no XPC_W1, or (ii) **T**=0_2 and either bits 210_h–217_h or bits 210_h–218_h (at Tag-manufacturer's option) of EPC memory are all zero, or (iii) **T**=1_2 and bits 210_h–$21F_h$ of EPC memory are all zero. 1: Tag has an XPC_W1 and either (i) **T**=0_2 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_2 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=00_h 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	RFU	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal Application: RFU and fixed[1] at zero Non-GS1 EPCglobal Application: see ISO/IEC 18000-63
212	RFU	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal Application: RFU and fixed[1] at zero Non-GS1 EPCglobal Application: see ISO/IEC 18000-63
213	RFU	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal Application: RFU and fixed[1] at zero Non-GS1 EPCglobal Application: see ISO/IEC 18000-63
214	**SA**	Computed	Sensor Alarm indicator	0: Tag is not reporting an alarm condition or does not support the SA flag 1: Tag is reporting an alarm condition
215	**SS**	Tag mfr defined	Simple Sensor indicator	0: Tag does not have a Simple Sensor 1: Tag has a Simple Sensor
216	**FS**	Tag mfr defined	Full Function Sensor indicator	0: Tag does not have a Full Function Sensor 1: Tag has a Full Function Sensor
217	**SN**	Tag mfr defined	Snapshot Sensor indicator	0: Tag does not have a Snapshot Sensor 1: Tag has a Snapshot Sensor
218	**B**	Tag mfr 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 response
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 mfr 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 *Kill* command or does not support the **K** bit 1: Tag can be killed by *Kill* 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 mfr 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 writable and may change despite the lock/permalock status of the EPC memory bank. Fixed bits are not writable 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_W1 a Tag shall not write and shall instead ignore the data values the Interrogator provides for **XEB**, **SA**, **C**, **SLI**, and **K**.

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_2 then **XI**=1_2.

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*, *QueryX*, or *QueryY* command (see 6.3.2.12.1, 6.3.2.12.2.2, and 6.3.2.12.2.3) 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 6.3.2.12.3.2).

The XPC_W1 bits and values shall be as follows (see also Table 6-19 and Table 6-20):

- **XEB** (bit 210_h): If bit 210_h is 0_2 then either a Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued. If bit 210_h is 1_2 then a Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero.
- RFU (bit 211_h): The Tag manufacturer (if the bit is not writable) or an Interrogator (if the bit is writable) shall set bit 211_h to zero.
- RFU or as defined in ISO/IEC 18000-63 (bit 212_h): If **T**=0_2 then the Tag manufacturer (if the bit is not writable) or an Interrogator (if the bit is writable) shall set bit 212_h to zero. If **T**=1_2 then bit 212_h is defined by ISO/IEC 18000-63.
- RFU (bit 213_h): The Tag manufacturer (if the bit is not writable) or an Interrogator (if the bit is writable) shall set bit 213_h to zero.
- **SA** (bit 214_h): If bit 214_h is 0_2 then a Tag does not have an alarm condition or does not support the **SA** flag. If bit 214_h is 1_2 then a Tag has an alarm condition. At the option of the Tag manufacturer, the Sensor Alarm (**SA**) bit may indicate one or more sensor alarms, tamper detection, low battery, or any other exception conditions on the tag.
- **SS** (bit 215_h): If bit 215_h is 0_2 then a Tag does not have a Simple Sensor. If bit 215_h is 1_2 then a Tag has a Simple Sensor as defined in 8.5
- **FS** (bit 216_h): If bit 216_h is 0_2 then a Tag does not have a Full Function Sensor. If bit 216_h is 1_2 then a Tag has one ore more Full Function Sensor as defined in 8.6

- **SN** (bit 217_h): If bit 217_h is 0_2 then a Tag does not have a Snapshot Sensor. If bit 217_h is 1_2 then a Tag has one or more Snapshot Sensors as defined in *8.7*. If a Tag has at least one Snapshot Sensor then the Tag shall implement XPC_W2 and XPC_W2 shall be used as the interface to all Snapshot Sensors on the Tag (see *8.7*).
- **B** (Battery-assisted passive indicator, bit 218_h): If bit 218_h is 0_2 then a Tag is either passive or does not support the **B** flag. If bit 218_h is 1_2 then a Tag is battery assisted.
- **C** (Computed response indicator, bit 219_h): If bit 219_h is 0_2 then the Tag's ResponseBuffer is empty or the Tag does not support a ResponseBuffer. If bit 219_h is 1_2 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 0_2 then a Tag has a deasserted **SL** flag or does not support an **SL** indicator. If bit $21A_h$ is 1_2 then a Tag has an asserted **SL** flag. Upon receiving a *Query* or *QueryX*, 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 0_2 then a Tag does not have a Tag notification or does not support the **TN** flag. If bit $21B_h$ is 1_2 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 writable, 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 0_2 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 1_2 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 *6.3.2.12.3.17*.
- **K** (Killable indicator, bit $21D_h$): If bit $21D_h$ is 0_2 then a Tag is not killable or does not support the **K** bit. If bit $21D_h$ is 1_2 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_2) OR (kill-pwd-permalock=0_2)]. See also *Table 6-22*. In words:
    - If the Tag supports authenticated kill and any key has a AuthKill=1_2 then the Tag is killable
    - If any bits of the kill password are 1_2 then the Tag is killable
    - If kill-pwd-read/write (see *6.3.2.12.3.6*) is 0_2 then the Tag is killable
    - If kill-pwd-permalock (see *6.3.2.12.3.6*) is 0_2 then the Tag is killable
- **NR** (Nonremovable indicator, bit $21E_h$): If bit $21E_h$ is 0_2 then a Tag is either removable or does not support the **NR** flag. If bit $21E_h$ is 1_2 then a Tag is nonremovable. See *4.1*.
- **H** (Hazmat indicator, bit $21F_h$): If bit $21F_h$ is 0_2 then a Tag is either not affixed to hazardous material or does not support the **H** flag. If bit $21F_h$ is 1_2 then a Tag is affixed to hazardous material.

If **SN**=1_2 then XPC_W2 shall be set as defined in *8.7.2* and *8.7.3*. If **SN**=0_2 then all XPC_W2 bits are RFU, in which case a Tag manufacturer (if XPC_W2 exists but is not writable) or an Interrogator (if XPC_W2 exists and is writable) shall set all XPC_W2 bits to 0_2.

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).

As per ISO/IEC 15963, 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.

As per ISO/IEC 15963, 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 5.1)
- 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 (MDID – see 5.2)
- 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.

Tags shall support a serialized TID by using either:

- class identifier $E0_h$, or
- class identifier $E2_h$ with **X**=1_2 and a unique serial number as defined in the GS1 EPC Tag Data Standard.

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 then a Tag shall implement UWC (see 6.3.2.12.3.3).

If File_0 of User memory exists and has not yet been written then the Tag shall set the first byte (i.e. File_0 memory addresses 00_h to 07_h) to the default value 00_h as specified in ISO/IEC 15961.

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 begins an inventory round during which it singulates *A* Tags in session S2 to *B,*
  - It powers off.
- Interrogator #2 powers-on, then
  - It begins 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 6-21*. 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 6-21*.

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, or 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 (i.e. upon receiving a *Select*, *Query*, or *QueryX*). 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*, *Query*, or *QueryX*.

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* and *QueryX* commands 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 6-21*. 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 6-21*.

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 6-21*. 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 6-21* (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 I*). 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 6-21*.

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*) or *QueryX* command (see *6.3.2.12.2.2*) whose inventoried parameter (for the Session specified in the *Query* or *QueryX*) and Sel parameter match its current flag values. Matching Tags shall load a 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

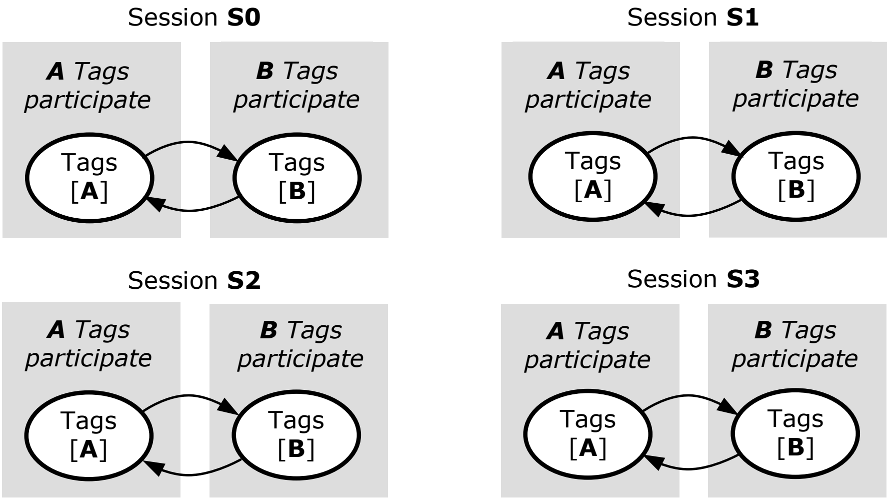


Table 6-21: Tag flags and persistence values

Flag	Time to Set[5]	Required persistence
S0 **inventoried** flag	≤ 2ms regardless of initial or final value[3]	Tag energized: indefinite Tag not energized: none
S1 **inventoried** flag[1]	≤ 2ms regardless of initial or final value[3]	Tag energized: nominal temperature range: 500ms < persistence < 5s extended temperature range: not specified Tag not energized: nominal temperature range: 500ms < persistence < 5s extended temperature range: not specified
S2 **inventoried** flag[1]	≤ 2ms regardless of initial or final value[3]	Tag energized: indefinite Tag not energized: nominal temperature range: 2s < persistence extended temperature range: not specified
S3 **inventoried** flag[1]	≤ 2ms regardless of initial or final value[3]	Tag energized: indefinite Tag not energized: nominal temperature range: 2s < persistence extended temperature range: not specified
Selected (**SL**) flag[1]	≤ 2ms regardless of initial or final value[3]	Tag energized: indefinite Tag not energized: nominal temperature range: 2s < persistence extended temperature range: not specified
C flag[1,2]	Deassert: ≤ 2ms[3] Assert: ≤ 0ms 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: 0s ≤ persistence < 5s extended temperature range: not specified
Optional security timeout	Prior to the first rising edge of the Tag reply to the Interrogator command that caused the security timeout.	Tag energized[4]: nominal temperature range: 20ms ≤ persistence < 200ms extended temperature range: not specified Tag not energized[4]: nominal temperature range: 20ms ≤ persistence < 200ms 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.

Note 5: A Tag may not properly set a persistence flag if the Interrogator turns off the energizing RF field prematurely or if the Tag does not remain in the Interrogator's energizing RF field.

6.3.2.6.2 Arbitrate state

Tags shall implement an **arbitrate** state. **Arbitrate** can be viewed as a "holding state" for Tags that began or completed initialization of an inventory round (see *6.3.2.10*) and the Tags' slot counters (see *6.3.2.6.8*) hold nonzero values. A Tag in **arbitrate**, for example, shall decrement its slot counter every time it receives a *QueryRep* command (see *6.3.2.12.2.5*) 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. If a Tag in **arbitrate** with an active T_8 timeout fails to receive a *QueryY* or other exempt command within $T_{8(max)}$, then the Tag shall return to **ready**. 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 $7FFF_h$ 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 a *Query* or *QueryX* with ReplyCRC=0_2 began the inventory round or shall backscatter RN16||CRC-5 if a *QueryX* with ReplyCRC=1_2 began the inventory round. If the Tag receives a valid acknowledgement (*ACK*) then it shall transition to the **acknowledged** state, backscattering the reply shown in *Table 6-18*. If the Tag fails to receive an *ACK* within time $T_{2(max)}$, 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 6-18*. 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 6-28*. 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 6-18*. 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 6-18*. 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

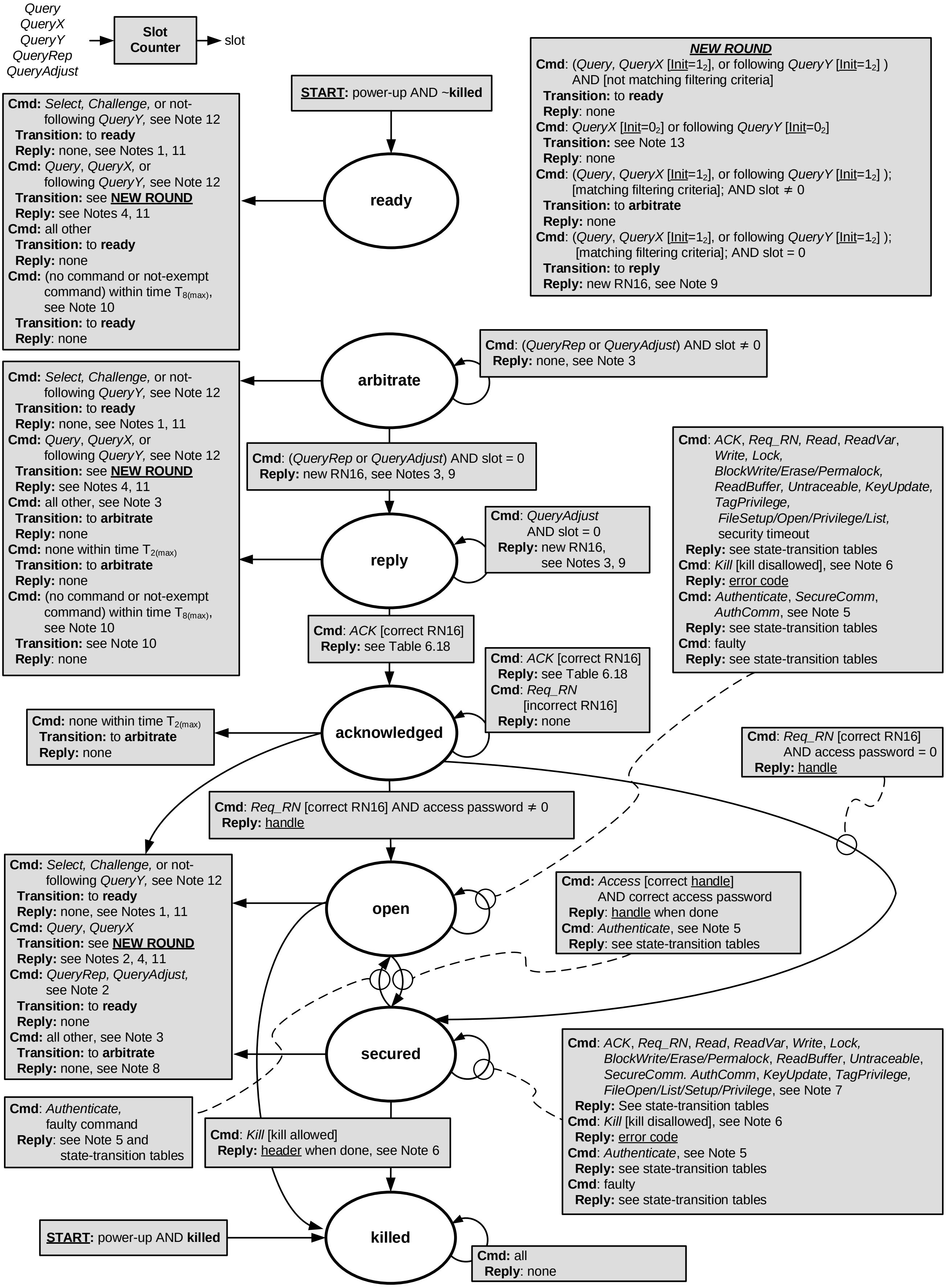
Tags shall implement a **killed** state. A Tag in either the **open** or **secured** state shall enter the **killed** state upon receiving a successful password-based *Kill*-command sequence with a correct nonzero kill password and handle. A Tag in the **secured** states shall enter the **killed** state upon a successful authenticated *Kill* (see *6.3.2.12.3.5*). *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*, *QueryX* with Init=1_2, *QueryY* with Init=1_2 or *QueryAdjust* command a Tag shall load 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* or *QueryX* specifies *Q*; a *QueryAdjust* may modify *Q* from the prior *Query* or *QueryX*.

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 (or RN16||CRC-5) 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*, *QueryX* or *QueryY* and that 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



Note 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.

Note 2: *Query*, *QueryX*: 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 the Session of the current inventory round.

Note 3: If the command is a *QueryRep* or *QueryAdj* and Session does not match the inventory round then the Tag ignores the command and remains in current state. If the Tag is in **reply**, **acknowledged**, **open**, or **secured** and the Tag receives a *QueryY* with a Session that does not match the inventory round then the Tag ignores the command and remains in current state. If the command is a *Select* and the parameters are incorrect, then the Tag ignores the command and remains in current state.

Note 4: *Query* begins and completes initialization of an inventory round. *QueryX* begins initializing an inventory round, and the (same) *QueryX* or a following *QueryY* completes initialization of an inventory round using Init=1_2. *Query* and *QueryX* may change the Session. Tags may go to **ready**, **arbitrate**, or **reply**.

Note 5: See the state-transition tables and the cryptographic suite for conditions, message formatting, tag responses, and state changes.

Note 6: Whether a kill is allowed or disallowed depends on the kill password, Tag privileges, and security timeout. See the *Kill* command-response table.

Note 7: If the Interrogator is authenticated then certain commands require encapsulation in an *AuthComm* or a *SecureComm*. See *Table 6-28*.

Note 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*.

Note 9: If the Interrogator initializes the inventory round with *QueryX* and ReplyCRC=1_2, then the Tag computes a CRC-5 over the new RN16 and appends the new RN16 with CRC-5.

Note 10: If T_8 timeout expires while the Tag is in **ready** or **arbitrate**, then the Tag remains in or returns to **ready**; it T_8 timeout expires when the Tag is in **reply**, then the Tag ignores the command and remains in current state. If the Tag receives a not-exempt command (see *6.3.2.12*) within $T_{8(max)}$, then the Tag ignores the command, stops the T_8 timeout, and remains in or returns to **ready**.

Note 11: If T_8 timeout is active, then the Tag stops T_8 timeout. If the command is a *QueryX* or is a following *QueryY* with Init=0_2, then the Tag starts a (new) T_8 timeout.

Note 12: A following *QueryY* is a *QueryY* received in **ready** or **arbitrate** if the *QueryY*: 1) is received within $T_{8(max)}$ timeout; 2) has MemBank≠00_2 or has Flx=0_2; and 3) has Session that matches the inventory round started by a *QueryX*. A not-following *QueryY* is a *QueryY* received in **ready** or **arbitrate** that 1) is received when T_8 timeout is disabled; 2) has MemBank=00_2; or 3) has a Session that does not match the inventory round started by a *QueryX*. In addition, a not-following *QueryY* is a *QueryY* with a Session that matches the inventory round if the Tag is in **reply**, **acknowledged**, **open**, or **secured**.

Note 13: If filtering criteria matches, set slot=$7FFF_h$ and go to **arbitrate**; or if filtering criteria does not match, go to **ready**.

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 6-26* or *Figure 6-28*).

- **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 10 ms after the end of T_{rp} 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 10 ms after T_{rp} in *Figure 6-3* 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

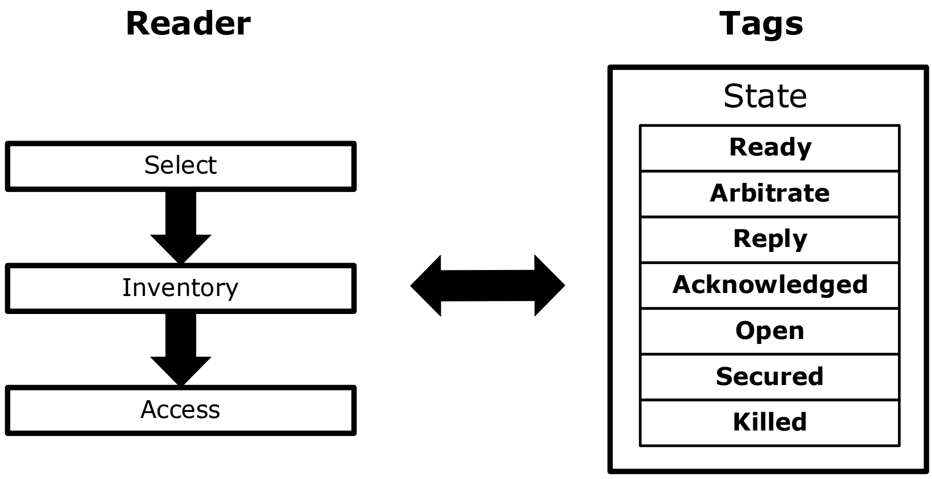
Interrogators manage Tag populations using the three basic operations shown in *Figure 6-22*. Each of these operations comprises multiple commands. The operations are defined as follows:

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. An Interrogator may also use *QueryX* command followed by zero or more *QueryY* commands to select a population of Tags based on a value or values in Tag memory.

b. **Inventory:** The process by which an Interrogator identifies Tags. An Interrogator begins an inventory round by transmitting a *Query* or *QueryX* command in one of four sessions. If the Interrogator sends a *QueryX*, the Interrogator can continue filtering Tags participating in the inventory round using one or more *QueryY* commands. After beginning an inventory round, one or more Tags may reply. The Interrogator detects a single Tag reply and requests the PC, optional XPC word(s), EPC (*Query* or *QueryX*) or TID (*QueryX*), 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.

An Interrogator may also use *QueryX* command followed by zero or more *QueryY* commands to select a population of Tags based on a value or values in Tag memory.

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 (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, 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*, *QueryX* or *QueryY* command uses these flags to choose which Tags participate in a subsequent inventory round. An Interrogator may inventory and access **SL** or ~**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 field 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 *6.3.2.12.3.11*) 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*, *QueryX*, *QueryY*, *QueryAdjust*, *QueryRep*, *ACK*, and *NAK. Query*, *QueryX*, and *QueryY* begin an inventory round and decide which Tags participate in the round ("inventory round" is defined in *4.1*).

An inventory round starts with initialization.

- *Query* begins and completes initialization of an inventory round.
- *QueryX* begins initializing an inventory round, and the same *QueryX* or a subsequent *QueryY* completes initialization of an inventory round using the Init=1_2 parameter (see *6.3.2.12.2.2* and *6.3.2.12.2.3*). While initializing an inventory round, an Interrogator may filter a population of Tags with user-defined criteria in *QueryX* and *QueryY*.

Tags participate in an inventory round if they match the criteria in the *Query*, *QueryX,* or *QueryX*/*QueryY*(s) that completed initialization of the inventory round. *Query* and *QueryX* contain a slot-count parameter *Q*, and a *QueryY* uses the *Q* value from the *QueryX* that preceded the *QueryY*. Upon receiving a *Query*, *QueryX* with Init=1_2, or *QueryY* with Init=1_2, participating Tags pick a random value in the range (0, 2^Q–1), inclusive, and load this value into their slot counter. Participating Tags that pick a slot counter value that is:

- zero, transition to the **reply** state and reply immediately; or
- nonzero, transition to the **arbitrate** state and await a *QueryAdjust* or *QueryRep* command.

Assuming a single Tag replies, the inventorying proceeds as follows:

a. The Tag backscatters an RN16 or an RN16||CRC-5 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 6-18*, and

d. The Interrogator issues a *QueryAdjust* or *QueryRep,* causing the identified Tag to invert its **inventoried** flag (i.e. *A*→*B* or *B*→*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 $T_{2(max)}$ (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 6-18*.

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 6-18*.

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*, *QueryX* with Init=1_2, or *QueryY* with Init=1_2 to initialize an inventory round, the Interrogator typically issues one or more *QueryAdjust* or *QueryRep* commands. Without introducing new Tags into the round, *QueryAdjust* instructs a Tag to load the slot counter with a new random value in the range (0, 2^Q-1) with *Q* incremented or decremented as specified by *QueryAdjust*. *QueryRep* decrements the slot counter 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* or *QueryX*, 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 when they receive a *QueryRep* and if a Tag's slot counter reaches 0000_h, then the Tag transitions to the **reply** state and backscatters an RN16 (or an RN16||CRC-5). Tags whose slot counter reached 0000_h, who replied, and who were not acknowledged (including Tags that responded to the original *Query*, *QueryX*, or *QueryY* and that 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*, *QueryX*, *QueryY*, *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*, *QueryX*: A *Query* or *QueryX* command begins an inventory round and chooses the session for the round. Tags in any state except **killed** execute a *Query* or *QueryX*. A *Query*, *QueryX* with Init=1_2, or *QueryY* with Init=1_2 completes initialization of an inventory round in the specified session and the Tag transitions to **ready**, **arbitrate**, or **reply**, as appropriate (see *Figure 6-21*).
  - If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* or *QueryX* whose Session parameter matches the prior session then it inverts its **inventoried** flag (i.e. *A*→*B* or *B*→*A*). After inverting its **inventoried** flag, the Tag transitions to **ready**, **arbitrate**, or **reply** states based on the session and other parameters of the *Query* or *QueryX*.

- If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* or *QueryX* whose Session parameter does not match the prior session then the Tag leaves its **inventoried** flag unchanged and the Tag transitions to **ready**, **arbitrate**, or **reply** based on the session and other parameters of the *Query* or *QueryX*.

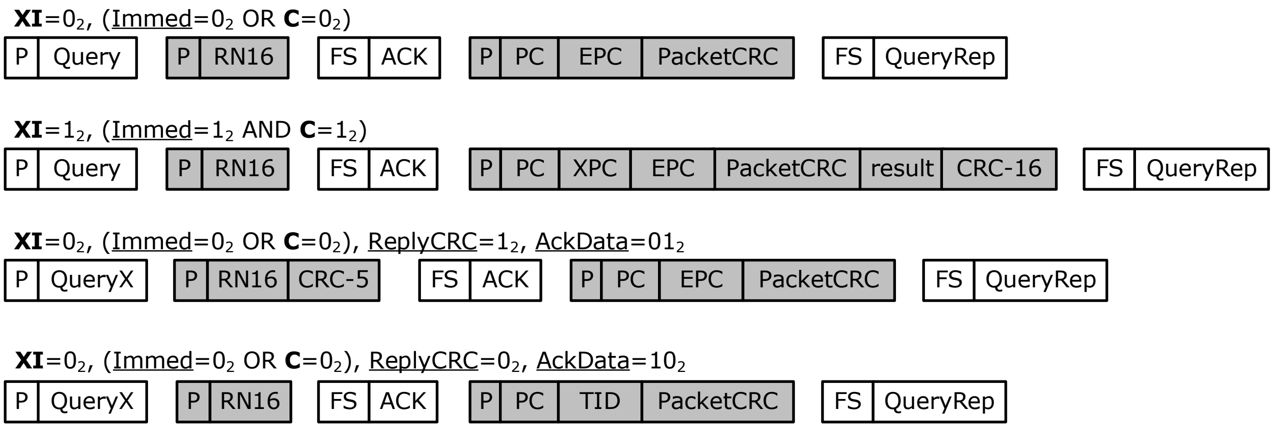
- *QueryAdjust, QueryRep*: Tags in any state except **ready** or **killed** execute a *QueryAdjust* or *QueryRep* command if, and only if, (i) the Session parameter in the command matches the Session parameter in the *Query* or *QueryX* that began the round, and (ii) the Tag is not in the middle of a *Kill*- or *Access*-command sequence (see *6.3.2.12.3.5* or *6.3.2.12.3.7*, 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* or *QueryX*, and the Tag is not in the middle of a *Kill*- or *Access*-command sequence (see *6.3.2.12.3.5* or *6.3.2.12.3.7*, respectively), then it inverts its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) for the current session and then transitions to **ready**.

Figure 6-23: One Tag reply

Condition	Description
Immed	Immed field in *Challenge*
C	**C** flag in XPC_W1
XI	**XI** flag in StoredPC
ReplyCRC	ReplyCRC field in *QueryX*
AckData	AckData field in *QueryX*

Symbol	Description
P	Preamble (R=>T or T=>R)
FS	Frame-Sync
RN16	16-bit Random Number
PC	Either StoredPC or PacketPC
XPC	Optional XPC word or words
EPC	EPC contents in MB01
result	ResponseBuffer contents
CRC-16	16-bit CRC computed over result
TID	Tag Identifier in MB10
CRC-5	5-bit CRC computed over RN16

R=>T Signaling (white box)
T=>R Signaling (grey box)



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:

a. **No Tags reply:** The Interrogator may issue a *QueryX* or another *Query*, or it may issue a *QueryAdjust* or *QueryRep*.

b. **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 6-18* 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* or *QueryX* with matching prior-round Session parameter will also invert the **inventoried** flag from $A \rightarrow B$).

c. **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 time $T_{2(max)}$ (see *Figure 6-18*), return to **arbitrate** and await the next *Query*, *QueryX* or *QueryAdjust* command.

As with other examples of one tag reply (see *Figure 6-23*), a *QueryX* begins an inventory round with a filtering criterion to inventory a subpopulation of tags. If a Tag receives a *QueryX* with ReplyCRC=1_2 that starts an inventory round, the Tag will backscatter RN16||CRC-5 if the Tag replies. If a Tag receives a *QueryX* with AckData=10_2 that starts an inventory round, the Tag will backscatter the traceable part of TID memory starting at address 00_h and ending at the last word of serialization.

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*, *ReadVar*, *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 6-28*. A Tag shall treat as invalid (see *Table C-34*) 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 6-18*.

As shown in *Table 6-28*, 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-21* and *Table C-34*.

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*, *QueryX*, *QueryAdjust*, *QueryRep*, or *NAK*. The Tag's response to a *Query*, *QueryX*, *QueryY*, *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 6-28*). 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*, *ReadVar*, *Write*, *Lock*, *Kill*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, and *Untraceable*. *Req_RN*, *Read*, *ReadVar*, *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. An Interrogator with a deasserted Untraceable privilege may read User-memory files for which it has read privileges if User memory is not untraceably hidden.

A *ReadVar* command allows an Interrogator to read one or more words of TID memory or of File_0 memory. An Interrogator with an asserted Untraceable privilege may read TID memory, and relevant portions of File_0 for which it has read privileges. An Interrogator with a deasserted Untraceable privilege may read the portions of TID memory that are not untraceably hidden. An Interrogator with a deasserted Untraceable privilege may read File_0 for which it has read privileges if File_0 is not untraceably hidden.

Table 6-22: 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 *Kill* command. If kill pwd is locked and access pwd is nonzero then requires access before writing new kill pwd
	Nonzero	All	Yes	Use *Kill* command with kill pwd.
Yes (and at least one key has AuthKill=1_2)	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 *Kill* 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 *Kill* 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 6-25*, *Table 6-26*, and *Table 6-28*.

The *Lock* and *BlockPermalock* commands allow an Interrogator to configure portions of Tag memory to be changeably or permanently writable or unwritable. 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 *6.3.2.12.3.17*).

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 *6.3.2.12.3.7* and outlined in *Figure 6-28*, 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.

An Untraceable command allows an Interrogator with an asserted Untraceable privilege (*Table 6-23* and *Table 6-24*) 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 reenable 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 6-26*. If a Tag supports authenticated killing then an Interrogator that authenticated itself using a key with an AuthKill privilege (see *Table 6-24*) may kill the Tag regardless of its kill-password value (zero or nonzero) using the abbreviated, authenticated kill process shown in *Figure 6-26*. 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 6-22*, 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 RN16 words as cover coding to obscure the data 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 bitwise XOR 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 bitwise XOR 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 *6.3.2.12.3.8*) 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 6-29* 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 6-29* 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 6-24*) 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 6-23* or *Table 6-24* 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 6-23*. A Tag that supports one or more cryptographic suites shall provide the Tag privileges shown in *Table 6-24*.

The privileges field in a *TagPrivilege* command is 16 bits in length, with a bit for each corresponding Tag privilege. Privilege bit assignments 12–15 in *Table 6-23* and *Table 6-24* are assigned by this protocol. Privilege bit assignments 8–11 are RFU for the access password (*Table 6-23*); they are assigned by the cryptographic suite for all keys (*Table 6-24*). Privilege bit assignments 4–7 are RFU for all keys. Privilege bit assignments 0–3 are defined by the Tag manufacturer and not specified by this protocol.

The labels in *Table 6-23* and *Table 6-24* 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 1_2 means an asserted or granted privilege; a 0_2 means a deasserted or denied privilege.
- CryptoSuperuser: whether a Tag grants the crypto superuser privilege to a key. If CryptoSuperuser=1_2 then a Tag grants the crypto superuser privilege to the key; if CryptoSuperuser=0_2 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_2 then a Tag grants the authenticated-kill privilege to the key; if AuthKill=0_2 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_2 then a Tag grants the privilege; if Untraceable=0_2 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 6-23* 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_2 in *Table 6-24* 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_2 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_N: 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 6-24*) 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 6-24*) for any key other than the one it used to authenticate itself.
- assert a deasserted privilege (value in *Table 6-24*) 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-34*).

Table 6-23: 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
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 6-24: 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 6-29* 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-34*).

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 *AuthComm* or a *SecureComm* command and instead treat the command's parameters as unsupported (see *Table C-34*).

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 (0000000000_2 – 1111111110_2). The files may have different size (including zero size). FileNum=0000000000_2 shall be reserved for the base file (File_0) of User memory. FileNum=1111111111_2 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 (0000000000_2 – 1111111110_2, respectively). FileSize 1111111111_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 (0000000000_2) to 1024 (1111111111_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 6-25*. If a Tag supports File_N, N>0 then it shall also provide the file privileges shown in *Table 6-26*. 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)×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 4×3=12 4-bit privileges.

A *FilePrivilege* may assign privileges 0000_2–0011_2 and 1100_2–1111_2 in *Table 6-25* and *Table 6-26*. 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-34*).

The access password or a key with a 0011_2 **secured**-state file privilege in *Table 6-25* or *Table 6-26* 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 permaunlocked 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 6-25* or *Table 6-26*) 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 DecFilePriv=1_2 (see *Table 6-23* and *Table 6-24*) then a Tag shall permit an Interrogator to self-reduce its privileges (move up one or more rows in *Table 6-25* or *Table 6-26*) to the currently open file for this access password or key. To be clear, if DecFilePriv=1_2 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 6-25: File_0 privileges

Privilege value	**Open** State (Privilege by File)				
	Read *ReadVar*	*Write* *BlockWrite* *BlockErase*	*Lock* *BlockPermalock*	*File-Privilege*	*File-Setup*
0000	✓	×	D	D	D
0001	✓	×	D	D	D
0010	✓	L	D	D	D
0011	✓	L	D	D	D
0100	RFU				
0101	RFU				
0110	RFU				
0111	RFU				
1000	RFU				
1001	RFU				
1010	RFU				
1011	RFU				
1100	Manufacturer defined				
1101	Manufacturer defined				
1110	Manufacturer defined				
1111	Manufacturer defined				
Privilege value	**Secured** State (Privilege by Access Password or Key)				
	Read *ReadVar*	*Write* *BlockWrite* *BlockErase*	*Lock* *BlockPermalock*	*File-Privilege*	*File-Setup*
0000	✓	×	×	×	×
0001	✓	L	×	P	×
0010	✓	L	✓	P	×
0011	✓	L	✓	✓	✓
0100	RFU				
0101	RFU				
0110	RFU				
0111	RFU				
1000	RFU				
1001	RFU				
1010	RFU				
1011	RFU				
1100	Manufacturer defined				
1101	Manufacturer defined				
1110	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_2 **secured**-state privilege is a superuser for File_0.

Table 6-26: File_N (N>0) privileges

Privilege value	**Open** State (Privilege by File)				
	Read	*Write BlockWrite BlockErase*	*Block-Permalock*	*File-Privilege*	*File-Setup*
0000	×	×	D	D	D
0001	✓	×	D	D	D
0010	✓	L	D	D	D
0011	✓	L	D	D	D
0100	RFU				
0101	RFU				
0110	RFU				
0111	RFU				
1000	RFU				
1001	RFU				
1010	RFU				
1011	RFU				
1100	Manufacturer defined				
1101	Manufacturer defined				
1110	Manufacturer defined				
1111	Manufacturer defined				
Privilege value	**Secured** State (Privilege by Access Password or Key)				
	Read	*Write BlockWrite BlockErase*	*BlockPermalock*	*FilePrivilege*	*FileSetup*
0000	×	×	×	×	×
0001	✓	×	×	P	×
0010	✓	L	×	P	×
0011	✓	L	✓	✓	✓
0100	RFU				
0101	RFU				
0110	RFU				
0111	RFU				
1000	RFU				
1001	RFU				
1010	RFU				
1011	RFU				
1100	Manufacturer defined				
1101	Manufacturer defined				
1110	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_2 **secured**-state privilege is a superuser for File_0.

Table 6-27: Allowed file resizing

File number	File is permalocked or permaunlocked	One or more file blocks are permalocked or permaunlocked	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 6-28: Access commands and Tag states in which they are permitted

Command	State			Subclass	Remark
	Acknowledged	**Open**	**Secured**		
Req_RN	allowed	allowed	allowed	Core	mandatory command
Read	disallowed	allowed	allowed	Core	mandatory command
ReadVar	disallowed	allowed	allowed	Core	mandatory command
Write	disallowed	allowed	allowed	Core	mandatory command; requires prior *Req_RN*
Kill (password-based)	disallowed	allowed	allowed	Core	mandatory command; requires prior *Req_RN*
Kill (authenticated)	disallowed	disallowed	allowed	Core	optional usage of mandatory *Kill* command
Lock	disallowed	disallowed	allowed	Core	mandatory command
Access	disallowed	allowed	allowed	Core	optional command; requires prior *Req_RN*
BlockWrite	disallowed	allowed	allowed	Core	optional command
BlockErase	disallowed	allowed	allowed	Core	optional command
BlockPermalock	disallowed	disallowed	allowed	Core	optional command
ReadBuffer	disallowed	allowed	allowed	Core	optional command
Untraceable	disallowed	disallowed	allowed	Core	optional command
Authenticate	disallowed	allowed	allowed	Security	optional command
AuthComm	disallowed	allowed	allowed	Security	optional command; requires prior authentication
SecureComm	disallowed	allowed	allowed	Security	optional command; requires prior authentication
KeyUpdate	disallowed	disallowed	allowed	Security	optional command; requires prior authentication
TagPrivilege	disallowed	disallowed	allowed	Security	optional command
FileOpen	disallowed	allowed	allowed	File	optional command
FileList	disallowed	allowed	allowed	File	optional command
FilePrivilege	disallowed	disallowed	allowed	File	optional command
FileSetup	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 0000_2 then the Tag opens the file but does not grant the Interrogator any file privileges.

Some privilege fields in *Table 6-25* and *Table 6-26* show "×" (disallowed by privilege). If *Read* is "×" for a privilege value then a Tag shall behave as if the memory location does not exist. Otherwise, if *Write*, *BlockWrite*, or *BlockErase* are "×" then the Tag shall behave as if the memory location is permalocked; and if *Lock* or *BlockPermalock* are "×" then the Tag shall behave as if the memory location is neither lockable nor unlockable. If *FilePrivilege* or *FileSetup* are "×" 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 6-23* and *Table 6-24*); if the Interrogator has a deasserted Untraceable privilege then the Tag shall treat these commands' parameters as unsupported (see *Table C-34*).

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, and XPC_W2 when **SN**=1_2.

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 6-27*.

6.3.2.12 Interrogator commands and Tag replies

Interrogator-to-Tag commands shall use the command codes, protection, and parameters shown in *Table 6-29*.

- *QueryRep* and *ACK* have 2-bit command codes beginning with 0_2.
- *Query*, *QueryX*, *QueryY*, *QueryAdjust*, and *Select* have 4-bit command codes beginning with 10_2.
- Other commands that are sensitive to link throughput use 8-bit command codes beginning with 110_2.
- Other commands that are insensitive to link throughput use 16-bit command codes beginning with 1110_2.
- *QueryRep*, *ACK*, *Query*, *QueryAdjust,* and *NAK* have the unique command lengths shown in *Table 6-29*. 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-34*).
- *Query* is protected by a CRC-5, shown in *Table 6-12* and detailed in *Annex F*.
- *Select*, *QueryX*, *QueryY*, *Req_RN*, *Read*, *ReadVar*, *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 6-29* 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.

- *Select, Challenge, Query, QueryX,* and *QueryY* are exempt commands; that is, these commands are exempt from being ignored if received within $T_{8(max)}$. If a Tag receives a not-exempt command within $T_{8(max)}$, then the Tag ignores the command and transitions to the **ready** state.

Table 6-29: 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)?	
QueryRep	00	4	Yes	Immediate	No	No	No	Unique length
ACK	01	18	Yes	Immediate	Yes	Yes	No	Unique length
Query	1000	22	Yes	Immediate	No	No	No	Unique length and a CRC-5
QueryAdjust	1001	9	Yes	Immediate	No	No	No	Unique length
Select	1010	> 44	Yes	None	No	No	No	CRC-16
QueryX with preamble	1011	> 46	Yes	Immediate	No	No	No	CRC-16
QueryY with frame-sync	1011	> 29	Yes	Immediate	No	No	No	CRC-16
NAK	11000000	8	Yes	Immediate	No	No	No	Unique length
Req_RN	11000001	40	Yes	Immediate	Yes	Yes	No	CRC-16
Read	11000010	> 57	Yes	Immediate	Yes	Yes	Yes	CRC-16
Write	11000011	> 58	Yes	Delayed	No	No	No	CRC-16
Kill	11000100	59	Yes	Delayed	Yes[1]	Yes[1]	Yes[1]	CRC-16
Lock	11000101	60	Yes	Delayed	Yes	Yes	Yes	CRC-16
Access	11000110	56	No	Immediate	No	No	No	CRC-16
BlockWrite	11000111	> 57	No	Delayed	Yes	Yes	Yes	CRC-16
BlockErase	11001000	> 57	No	Delayed	Yes	Yes	Yes	CRC-16
BlockPermalock	11001001	> 66	No	Immediate and Delayed	Yes	Yes	Yes	CRC-16
Used by ISO 18000-63	11001010 … 11010001	–	–	–	–	–	–	–
ReadBuffer	11010010	67	No	Immediate	No	Yes	No	CRC-16
FileOpen	11010011	52	No[4]	Immediate	Yes	Yes	Yes	CRC-16
Challenge	11010100	> 48	No	None	No	No	No	CRC-16
Authenticate	11010101	> 64	No	In-process	No	No	No	CRC-16
SecureComm	11010110	> 56	No	In-process	No	No	No	CRC-16
AuthComm	11010111	> 42	No	In-process	No	No	No	CRC-16
ReadVar	11011000	> 57	Yes	Immediate	Yes	Yes	Yes	CRC-16
Used by ISO 18000-63	11011001	–	–	–	–	–	–	–

Command	Code	Length (bits)	Mandatory Command (Y/N)?	Reply Type	Encapsulation			Protection
					SecureComm[2] (Y/N)?	AuthComm[2] (Y/N)?	Mandatory[3] (Y/N)?	
Reserved for future use	11011010 … 11011111	–	–	–	–	–	–	–
Reserved for custom commands	11100000 00000000 … 11100000 11111111	–	–	–	–	–	–	Mfr defined
Reserved for proprietary commands	11100001 00000000 … 11100001 11111111	–	–	–	–	–	–	Mfr defined
Untraceable	11100010 00000000	62	No	Delayed	Yes	Yes	Yes	CRC-16
FileList	11100010 00000001	71	No	In-process	Yes	Yes	Yes	CRC-16
KeyUpdate	11100010 00000010	> 72	No	In-process	Yes	Yes	No	CRC-16
TagPrivilege	11100010 00000011	78	No	In-process	Yes	Yes	Yes	CRC-16
FilePrivilege	11100010 00000100	68	No	In-process	Yes	Yes	Yes	CRC-16
FileSetup	11100010 00000101	71	No	In-process	Yes	Yes	Yes	CRC-16
Reserved for future use	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 6-30. 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. An Interrogator may also select a Tag subpopulation using QueryX and QueryY 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 101_2, 110_2, or 111_2.
- Action elicits the Tag behavior in *Table 6-31*, 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 (memory bank) specifies how a Tag applies Mask. If MemBank=00_2 and the Tag supports File_N, N>0, then the Tag searches for at least one file whose FileType matches Mask; if MemBank=00_2 and the Tag does not support File_N, N>0, then the Tag shall ignore the *Select*. If MemBank=01_2, 10_2, 11_2 then a Tag applies Mask to the EPC memory bank, TID memory bank, or File_0, 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=00_2 then an Interrogator shall set Pointer to 00_h; if a Tag supports File_N, N>0, receives a *Select* with MemBank=00_2, and receives a nonzero Pointer value then the Tag shall ignore the *Select*.
- Length specifies the length of Mask. Length is 8 bits, allowing Masks from 0 to 255 bits in length. If Length=00_h and MemBank≠00_2 then the Tag shall treat the condition as matching, unless Pointer references a memory location that does not exist, or Pointer references a memory location that is untraceably hidden, or Truncate=1_2 and Mask is outside the EPC specified in the length field in the StoredPC, in which case the Tag is not-matching. If MemBank=00_2 then an Interrogator shall set Length=00001000_2; if a Tag supports File_N, N>0 and receives a *Select* with MemBank=00_2 and Length≠00001000_2 then the Tag shall ignore the *Select*.
- Mask is either a FileType (if MemBank=00_2) or a bit string that a Tag compares to a memory location that begins at Pointer and ends Length bits later (if MemBank≠00_2). An untraceable Tag that supports File_N, N>0, shall process a *Select* with MemBank=00_2 whose User memory is traceable; an untraceable Tag shall process a *Select* with MemBank≠00_2 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=00_2: If a Tag supports File_N, N>0, and has a file with the specified FileType then the Tag is matching; if a Tag supports File_N, N>0, and does not have a file with the specified FileType then the Tag is not-matching.
  - MemBank≠00_2: 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_2 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_2 or Sel=11_2, 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 the Tag shall assert it:
  - in the last *Select* that the Interrogator issues prior to sending a *Query*,
  - only if the *Select* has Target=100_2, 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_2 and:

- Target≠100_2 or MemBank≠01_2 then the Tag shall ignore the *Select*.
- MemBank=01_2 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 (00000_2) inserted between the preamble and the truncated reply. Specifically, when truncating its replies a Tag backscatters 00000_2, then the portion of its EPC following Mask, and then a PacketCRC. See *Table 6-18*.

A Tag shall power-up with truncate disabled.

Mask may end at the last bit of the EPC, in which case a truncating Tag shall backscatter 00000_2 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 00100_2 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 00101_2 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 6-30: *Select* command

	# of bits	description
Command	4	1010
Target	3	000: **Inventoried** (S0) 001: **Inventoried** (S1) 010: **Inventoried** (S2) 011: **Inventoried** (S3) 100: **SL** 101: RFU 110: RFU 111: RFU
Action	3	See *Table 6-31*
MemBank	2	00: FileType 01: EPC 10: TID 11: File_0
Pointer	EBV	Starting Mask address
Length	8	Mask length (bits)
Mask	variable	Mask value
Truncate	1	0: disable truncation 1: enable truncation
CRC	16	CRC-16

Table 6-31: 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 6-32*. 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 (include reply length) specifies whether the Tag omits or includes Length in its stored reply. If IncRepLen=0_2 then the Tag omits Length from its stored reply; if IncRepLen=1_2 then the Tag includes Length in its stored reply.
- Immed (immediate reply) specifies whether a Tag concatenates response to its EPC or TID when replying to an *ACK.* If Immed=0_2, or if Immed=1_2 and **C**=0_2, then the Tag does not concatenate response to its EPC or TID when replying to an *ACK*; if Immed=1_2 and **C**=1_2 then the Tag backscatters (EPC or TID)|| response when replying to an *ACK*. Unless otherwise specified by the cryptographic suite, a Tag shall support Immed=1_2 and may support Immed=0_2.
- 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 00_2. 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_2 then the Tag also stores the Length of the response (in bits) as shown in *Figure 6-17*. An Interrogator may subsequently read response using a *ReadBuffer* command. See *6.3.2.12.3.16*.

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 6-21*; otherwise a Tag may deassert its **C** flag upon transitioning from **acknowledged**, **open,** or **secured**. As described above the ResponseBuffer contents may include a Length field and may be a cryptographic response or an error code. A Tag shall not permit an Interrogator to read its ResponseBuffer when **C**=0_2.

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 6-18*. See also *Figure 6-23*.

A *Challenge* may precede a *Query* or *QueryX*. 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_2) 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 6-32: *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 Length from reply 1: include Length in reply	0: do not transmit result in response to *ACK* 1: transmit result in response to *ACK*	CSI	length of Message	Message (depends on CSI)	CRC-16

6.3.2.12.2 Inventory commands

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

6.3.2.12.2.1 *Query* (mandatory)

Interrogators and Tags shall implement the *Query* command shown in *Table 6-33*. *Query* initiates and specifies an inventory round. *Query* includes the following fields:

- DR (TRcal divide ratio) sets the T=>R backscatter 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 (select **SL** flag) 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 6-29*).

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-34*).

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 6-34* 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 6-33: *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 6-34: Tag reply to a *Query* command

	RN
# of bits	16
description	RN16

6.3.2.12.2.2 *QueryX* (mandatory)

Interrogators and Tags shall implement the *QueryX* command shown in *Table 6-35*.

Table 6-35: *QueryX* command[1]

<table>
<tr><th colspan="2"></th><th># of bits</th><th colspan="4">description</th></tr>
<tr><td colspan="2">Cmd</td><td>4</td><td colspan="4">1011</td></tr>
<tr><td colspan="2">Init</td><td>1</td><td colspan="4">0: continue initializing inventory round with QueryY
1: complete initializing inventory round</td></tr>
<tr><td colspan="2">Session</td><td>2</td><td>00: S0</td><td>01: S1</td><td>10: S2</td><td>11: S3</td></tr>
<tr><td colspan="2">Action[2]</td><td>3</td><td colspan="4">See Table 6-31</td></tr>
<tr><td colspan="2">SelType</td><td>3</td><td colspan="4" rowspan="2">See Table 6-36</td></tr>
<tr><td colspan="2">FastMask</td><td>0 or 9</td></tr>
<tr><td colspan="2">Flx</td><td>1</td><td colspan="4">0: no flexible filter
1:flexible filter included</td></tr>
<tr><td rowspan="6">Flexible filter (included if $Flx=1_2$, excluded if $Flx=0_2$)</td><td>Comp</td><td>0 or 2</td><td>00: ≥</td><td>01: ≤</td><td>10: ≠</td><td>11: =</td></tr>
<tr><td>MemBank</td><td>0 or 2</td><td colspan="4">00: terminate initializing inventory round
01: EPC
10: TID
11: File_0</td></tr>
<tr><td>Pointer</td><td>0 or EBV</td><td colspan="4">Start Mask address (bit address not word address)</td></tr>
<tr><td>Length</td><td>0 or 8</td><td colspan="4">Mask length (in bits)</td></tr>
<tr><td>Mask</td><td>variable</td><td colspan="4">Mask value</td></tr>
<tr><td>Truncate</td><td>0 or 1</td><td colspan="4">0: disable truncation
1: enable truncation</td></tr>
<tr><td colspan="2">ReplyCRC</td><td>1</td><td colspan="4">0: reply with RN16
1: reply with RN16||CRC-5</td></tr>
<tr><td colspan="2">AckData</td><td>2</td><td colspan="4">00: terminate initializing inventory round
01: EPC
10: TID
11: terminate initializing inventory round</td></tr>
<tr><td colspan="2">DBLF</td><td>3</td><td colspan="4">See Table 6-9</td></tr>
<tr><td colspan="2">DR</td><td>1</td><td colspan="4">0: DR=8
1: DR=64/3</td></tr>
<tr><td colspan="2">M</td><td>2</td><td>00: M=1</td><td>01: M=2</td><td>10: M=4</td><td>11: M=8</td></tr>
<tr><td colspan="2">TRext</td><td>1</td><td colspan="4">0: no pilot tone
1: use pilot tone</td></tr>
<tr><td colspan="2">Sel</td><td>2</td><td>00: All</td><td>01: All</td><td>10: ~SL</td><td>11: SL</td></tr>
<tr><td colspan="2">Target</td><td>1</td><td colspan="4">0: A
1: B</td></tr>
<tr><td colspan="2">Q</td><td>4</td><td colspan="4">0–15</td></tr>
<tr><td colspan="2">CRC</td><td>16</td><td colspan="4">CRC-16</td></tr>
</table>

Note 1: Prepend preamble to *QueryX* command.

Note 2: Action applies only to Session **inventoried** flags

Upon receiving a *QueryX*, a Tag shall follow the procedure outlined in *Figure 6-24* and *Figure 6-25*. *QueryX* begins an inventory round. *QueryX* allows an Interrogator to filter a Tag subpopulation based on user-defined criteria. *QueryX* begins an inventory round and includes the following parameters:

- Init (initialize an inventory round) indicates whether the *QueryX* completes initialization of an inventory round or a following *QueryY* continues initialization. A *QueryX* with Init=1_2 completes initialization of the inventory round. A *QueryX* with Init=0_2 indicates that a following *QueryY* will continue initializing the inventory round.
- Session indicates the **inventoried** flag modified by Action and chooses the session for the inventory round (see *6.3.2.10*).
- Action elicits the Tag behavior in *Table 6-31*, in which matching and not-matching Tags assert their **inventoried** flag to *A* or *B* as specified by Session. A Tag shall compute matching or not-matching for Action as specified in *Table 6-37*. Computing matching/not-matching for Action depends on the SelType filtering condition and Flexible condition. Based on *Table 6-37*, a Tag shall compute the Action on the **inventoried** flag as specified by Session before comparing to Target.
- SelType (selection type) and FastMask specify the SelType condition as specified in *Table 6-36*. SelType is a 3-bit length parameter, and FastMask is zero or 9-bit length parameter.
- Flx (flexible filter) indicates if the Flexible condition parameters are included in the command. If Flx=1_2 then an Interrogator shall include the Comp, MemBank, Pointer, Length, Mask, and Truncate fields, and if Flx=0_2 then these fields are excluded. *Table 6-38* specifies the Flexible condition if the Flexible filter fields are included (Flx=1_2):
    - Comp (comparison) specifies the operator to be used by a Tag in a comparison between Mask and memory in MemBank that begins at Pointer and ends Length bits later. Both Mask and memory contents in MemBank are interpreted by a Tag as unsigned integer values. When a Tag evaluates the Flexible condition, the memory contents are to the left of the operator, and Mask is to the right of the operator. For example, if the MSB in the memory location (specified by Pointer) is 1_2 and the most significant bit in Mask (transmitted first) is 0_2, then the condition would be matching if Comp=00_2, not-matching if Comp=01_2, matching if Comp=10_2, and not-matching if Comp=11_2.
    - Pointer specifies a starting bit address for the Mask comparison. Pointer uses EBV formatting (see *Annex A*) and bit (not word) addressing.
    - MemBank specifies the memory bank for the comparison. If a Tag receives MemBank=00_2, then the Tag shall transition to **ready**.
    - Length specifies the length of Mask. Length is 8 bits, allowing Masks from 0 to 255 bits in length. If Length=00_h and MemBank≠00_2 then the Tag shall treat the Flexible condition as matching, unless Pointer references a memory location that does not exist, or Pointer references a memory location that is untraceably hidden, or Truncate=1_2 and Mask is outside the EPC specified in the length field in the StoredPC, in which case the Tag is not-matching.
    - Mask is a bit string that a Tag compares to a memory location that begins at Pointer and ends Length bits later.
    - Truncate indicates whether a Tag's backscattered reply shall be truncated to those EPC bits that follow Mask. If an Interrogator sets Init=1_2, Flx=1_2, Truncate=1_2, MemBank=01_2, and AckData=01_2, 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 sets Init=0_2, or Flx=0_2, or Truncate=0_2, or MemBank≠01_2, or AckData=10_2, then the Tag shall disable truncation and the Tag is not-matching.
- ReplyCRC (reply with CRC-5) indicates whether a Tag shall calculate a CRC-5 over the RN16. If ReplyCRC=1_2 then the Tag shall backscatter the RN16 followed by the CRC-5, and if ReplyCRC=0_2 then the Tag shall backscatter the RN16 without a CRC-5.
- AckData (data for *ACK*) indicates whether a Tag shall backscatter an EPC or TID (see *Table 6-18*) in reply to an *ACK*. If AckData=01_2 then a Tag shall backscatter an EPC; if AckData=10_2 then a Tag shall backscatter the traceable part of TID memory starting at address 00_h and ending at the last word of serialization; and if AckData=00_2 or if AckData=11_2 then a Tag shall stop any T_8 timeout and transition to **ready**.

- DBLF (digital backscatter link frequency) sets the T=>R backscatter link frequency as described in 6.3.1.3.3 and Table 6-9. If a Tag does not support a T=>R backscatter link frequency that is not required by *QueryX* (DBLF=000_2, =101_2, or =110_2) then the Tag shall stop (if active) T_8 timeout and transition to **ready**.
- DR (divide ratio) sets the T=>R backscatter link frequency as described in 6.3.1.2.8 and Table 6-9.
- M (number of subcarrier 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 filters which Tags participate in an inventory round (see 6.3.2.12.1.1 and 6.3.2.10).
- Target filters Tags whose **inventoried** flag is *A* or *B* as specified by Session. 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 *QueryX* with a preamble (see 6.3.1.2.8).

An Interrogator shall not encapsulate a *QueryX* in a *SecureComm* or *AuthComm* (see Table 6-29).

The CRC-16 that protects a *QueryX* is calculated over the first command-code bit to the last Q bit. If a Tag receives a *QueryX* with a CRC-16 error then it shall treat the command as invalid (see Table C-34).

A Tag shall stop any T_8 timeout (if active) upon receiving a *QueryX* with Init=1_2 or Init=0_2.

If a Tag receives *QueryX* with Init=1_2:

- 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, and then:
  - if the Tag loads its slot counter with zero, then the Tag shall transition to **reply** and shall backscatter an RN16 followed by CRC-5 if ReplyCRC=1_2 or shall (only) backscatter an RN16 if ReplyCRC=0_2 as shown in Table 6-39 using the *immediate* reply type specified in 6.3.1.6.1; or
  - if the Tag loads a non-zero value into its slot counter, the Tag shall remain silent and transition to **arbitrate**.
- Tags with not-matching Sel or not-matching Target shall remain silent and transition to **ready**.

If a Tag receives *QueryX* with Init=0_2:

- Tags with matching Sel and Target shall start a T_8 timeout (see Table 6-16), load their slot counter with $7FFF_h$, remain silent, and transition to **arbitrate**.
- Tags with matching Sel and not-matching Target shall start a T_8 timeout, remain silent and transition to **ready**.
- Tags with not-matching Sel shall remain silent and transition to **ready**.
- If a Tag starts a T_8 timeout (e.g. Init=0_2 and matching Sel) then the Tag shall process the T_8 timeout and subsequent commands as described in 6.3.1.6.5.

Table 6-36: SelType filtering condition

<table>
<tr><th rowspan="2">SelType value</th><th rowspan="2">FastMask length (bits)</th><th colspan="2">Resulting SelType condition</th></tr>
<tr><th>Matching[4]</th><th>Not-matching[4]</th></tr>
<tr><td>000: None</td><td rowspan="4">0</td><td>never</td><td>always</td></tr>
<tr><td>001: None</td><td>never</td><td>always</td></tr>
<tr><td>010: T</td><td>T=0_2</td><td>T=1_2</td></tr>
<tr><td>011: All</td><td>always</td><td>never</td></tr>
<tr><td>100: T+</td><td rowspan="4">9</td><td>((T=0_2) AND (FastMask$_{1h:8h}$ = EPC$_{20h\text{-}27h}$)) if FastMask$_{0h}$ = 0_2;
((T=1_2) AND (FastMask$_{1h:8h}$ = EPC$_{18h\text{-}1Fh}$)) if FastMask$_{0h}$ = 1_2</td><td>((T=1_2) OR (FastMask$_{1h:8h}$ ≠ EPC$_{20h\text{-}27h}$)) if FastMask$_{0h}$ = 0_2;
((T=0_2) OR (FastMask$_{1h:8h}$ ≠ EPC$_{18h\text{-}1Fh}$)) if FastMask$_{0h}$ = 1_2</td></tr>
<tr><td>101: MDID[6]</td><td>FastMask$_{0h:8h}$ = TID$_{0Bh:13h}$</td><td>FastMask$_{0h:8h}$ ≠ TID$_{0Bh:13h}$</td></tr>
<tr><td>110: CCSI[1]</td><td>FastMask$_{0h:8h}$ = (FastMask$_{0h}$ & C) || CSI</td><td>FastMask$_{0h:8h}$ ≠ (FastMask$_{0h}$ & C) || CSI</td></tr>
<tr><td>111: XPC[2]</td><td>FastMask$_{0h:8h}$ = (FastMask$_{0h}$ & SuppSens[3]) || (FastMask$_{1h:8h}$ & EPC$_{218h\text{-}21Fh}$)[5]</td><td>FastMask$_{0h:8h}$ ≠ (FastMask$_{0h}$ & SuppSens[3]) || (FastMask$_{1h:8h}$ & EPC$_{218h\text{-}21Fh}$)[5]</td></tr>
</table>

Note 1: A Tag shall be not-matching if the Tag does not support a cryptographic suite. The Tag may or may not check if the CSI transmitted in a previous *Challenge* command is the same as that transmitted in the *QueryX* command.

Note 2: A Tag shall be not-matching if the Tag does not support at least one bit in EPC$_{214h\text{-}21Fh}$; if a Tag supports one or more bits in EPC$_{214h\text{-}21Fh}$ then the Tag shall treat unsupported bits in EPC$_{218h\text{-}21Fh}$ as zero in the '&' Boolean bitwise 'and' operation. As an example, if a Tag supports one or more bits in in EPC$_{214h\text{-}21Fh}$ and if FastMask$_{0h:8h}$ = 000000000_2 then the Tag shall be matching.

Note 3: If a Tag supports one or more sensor bits in EPC$_{214h\text{-}217h}$ then SuppSens=1_2; if a Tag does not support one or more sensor bits in EPC$_{214h\text{-}217h}$ then SuppSens=0_2.

Note 4: 'AND'/'OR' are logical operations; & is a Boolean bitwise 'and' operation; || is a concatenation operation.

Note 5: As an example, a Tag shall be not-matching when one or more bits are asserted in FastMask$_{1h:8h}$ and one or more of the corresponding bits in EPC$_{218h\text{-}21Fh}$ are deasserted.

Note 6: A tag shall be not-matching if the allocation class identifier (see *6.3.2.1*) is not E2$_h$ or if the MDID is untraceably hidden.

Table 6-37: Matching or not-matching for *QueryX* Action parameter

SelType condition	Flx value	Flexible condition	Result for Action
matching	0	-	matching
matching	1	matching	
not-matching	don't care	don't care	not-matching
matching	1	not-matching	

Table 6-38: Flexible condition

Comp		Resulting Flexible condition	
Value	Name	Matching	Not-matching
00	≥	unsigned integer value[1] specified by memory location MemBank, Pointer, and Length is greater than or equal to unsigned integer value[2] of Mask	MemBank = 00_2; or memory location specified by MemBank, Pointer, Length is untraceably hidden or does not exist; or unsigned integer value[1] specified by memory location MemBank, Pointer, and Length, is less than unsigned integer value[2] of Mask
01	≤	unsigned integer value[1] specified by memory location MemBank, Pointer, and Length is less than or equal to unsigned integer value[2] of Mask	MemBank = 00_2; or memory location specified by MemBank, Pointer, Length is untraceably hidden or does not exist; or unsigned integer value[1] specified by memory location MemBank, Pointer, and Length, is greater than unsigned integer value[2] of Mask
10	≠	memory location specified by MemBank, Pointer, and Length is not equal to Mask	MemBank = 00_2; or memory location specified by MemBank, Pointer, Length is untraceably hidden or does not exist; or memory location specified by MemBank, Pointer, and Length equals Mask
11	=	memory location specified by MemBank, Pointer, and Length equals Mask	MemBank = 00_2; or memory location specified by MemBank, Pointer, Length is untraceably hidden or does not exist; or memory location specified by MemBank, Pointer, and Length is not equal to Mask

Note 1: Pointer specifies the most significant bit in memory location for the unsigned integer value.

Note 2: The first transmitted bit of Mask specifies the most significant bit of the unsigned integer value.

Table 6-39: Tag reply for *QueryX* command

	RN	CRC
# of bits	16	0 or 5
description	RN16	CRC-5

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

A Tag shall support DBLF, DR, and M values specified in *6.3.1.3.3*, *Table 6-9* and *Table 6-10*, respectively.

A Tag in any state other than **killed** shall execute a *QueryX* command, begin a new round in the specified session and transition to **ready**, **arbitrate**, or **reply**, as appropriate (see *Figure 6-21*). A Tag in the **killed** state shall ignore a *QueryX*.

6.3.2.12.2.3 *QueryY* (mandatory)

Interrogators and Tags shall implement the *QueryY* command shown in *Table 6-40*.

Table 6-40: *QueryY* command[1]

<table>
<tr><th colspan="2"></th><th># of bits</th><th colspan="4">description</th></tr>
<tr><td colspan="2">Cmd</td><td>4</td><td colspan="4">1011</td></tr>
<tr><td colspan="2">Init</td><td>1</td><td colspan="4">0: continue initializing inventory round with (new) QueryY
1: complete initializing inventory round</td></tr>
<tr><td colspan="2">Session</td><td>2</td><td>00: S0</td><td>01: S1</td><td>10: S2</td><td>11: S3</td></tr>
<tr><td colspan="2">Action[2]</td><td colspan="5" rowspan="10">See Table 6-35</td></tr>
<tr><td colspan="2">SelType</td></tr>
<tr><td colspan="2">FastMask</td></tr>
<tr><td colspan="2">Flx</td></tr>
<tr><td rowspan="6">Flexible filter (included if Flx=1_2, excluded if Flx=0_2)</td><td>Comp</td></tr>
<tr><td>MemBank</td></tr>
<tr><td>Pointer</td></tr>
<tr><td>Length</td></tr>
<tr><td>Mask</td></tr>
<tr><td>Truncate</td></tr>
<tr><td colspan="2">CRC</td><td>16</td><td colspan="4">CRC-16</td></tr>
</table>

Note 1: Prepend frame-sync to *QueryY* command.

Note 2: Action applies only to Session **inventoried** flags.

Table 6-41: Tag reply for *QueryY* command

	RN	CRC
# of bits	16	0 or 5
description	RN16	CRC-5

A Tag shall follow the procedure outline in *Figure 6-24* and *Figure 6-25*. An Interrogator shall transmit a *QueryY* within T_8 after a previous *QueryX* with Init=0_2 or a previous *QueryY* with Init=0_2. *QueryY* allows an Interrogator to filter a Tag subpopulation based on additional user-defined criteria. *QueryY* includes the following parameters:

- Init (initialize an inventory round) indicates whether the *QueryY* completes initialization of an inventory round or a following *QueryY* continues initialization. A *QueryY* with Init=1_2 completes initialization of the inventory round. A *QueryY* with Init=0_2 indicates that a following *QueryY* will continue initializing the inventory round, and a Tag shall start a (new) T_8 timeout when Init=0_2.
- Session corroborates the session number of the *QueryX* that began initializing the inventory round. An Interrogator shall transmit Session in *QueryY* equal to the Session of the *QueryX* that began initializing the inventory round.
- Action – see *6.3.2.12.2.2*.
- SelType – see *6.3.2.12.2.2*.
- Flx – see *6.3.2.12.2.2*.
  - Comp – see *6.3.2.12.2.2*.
  - MemBank – see *6.3.2.12.2.2*.
  - Pointer – see *6.3.2.12.2.2*.
  - Length – see *6.3.2.12.2.2*.

- Mask – see 6.3.2.12.2.2.
- Truncate indicates whether a Tag's backscattered reply shall be truncated to those EPC bits that follow Mask. If an Interrogator sets Init=1_2, Flx=1_2, Truncate=1_2, and MemBank=01_2 in a *QueryY* and if AckData=01_2 in the *QueryX* that began the inventory round, 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 sets Init=0_2, Flx=0_2, Truncate=0_2, or MemBank≠01_2 in a *QueryY* or if AckData=10_2, in the *QueryX* that began the inventory round, then the Tag shall disable truncation.

An Interrogator shall prepend a *QueryY* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate a *QueryY* in a *SecureComm* or *AuthComm* (see Table 6-29).

The CRC-16 that protects a *QueryY* is calculated over the first command-code bit to the Truncate bit if Flx=1_2 or to the Flx bit if Flx=0_2. If a Tag receives a *QueryY* with a CRC-16 error then it shall treat the command as invalid (see Table C-34).

If Tags in the **ready** or **arbitrate** states receive a *QueryY* within $T_{8(max)}$ and if the *QueryY* has a Session that matches the inventory round and has Init=1_2, then:

- Tags with matching Target from the prior *QueryX* shall pick a random value in the range (0, 2^Q–1), inclusive, and shall load this value into their slot counter, and then:
  - if the Tag loads its slot counter with zero, then the Tag shall transition to **reply** and shall backscatter as shown in Table 6-41 using the *immediate* reply type specified in 6.3.1.6.1 and using the ReplyCRC specified in the prior *QueryX*; or
  - if the Tag loads a non-zero value into it slot counter, the Tag shall remain silent and transition to **arbitrate**.
- Tags with not-matching Target from the prior *QueryX* shall remain silent and transition to **ready**.

If Tags in the **ready** or **arbitrate** states receive a *QueryY* within $T_{8(max)}$ and if the *QueryY* has a Session that matches the inventory round and has Init=0_2, then:

- Tags with matching Target from the prior *QueryX* shall start a (new) T_8 timeout (see Table 6-16), load their slot counter with $7FFF_h$, remain silent, and transition to **arbitrate**.
- Tags with not-matching Target from the prior *QueryX* shall start a T_8 timeout, remain silent and transition to **ready**.
- If a Tag starts a T_8 timeout (e.g., Init=0_2) then the Tag shall process the T_8 timeout and subsequent commands as described in 6.3.1.6.5.

A Tag that receives a *QueryY* in the **ready** or **arbitrate** states shall stop (if active) the T_8 timeout, remain silent, and transition to **ready** if the Tag receives the *QueryY* after a T_8 timeout, if the Session does not match the inventory round, or if MemBank=00_2.

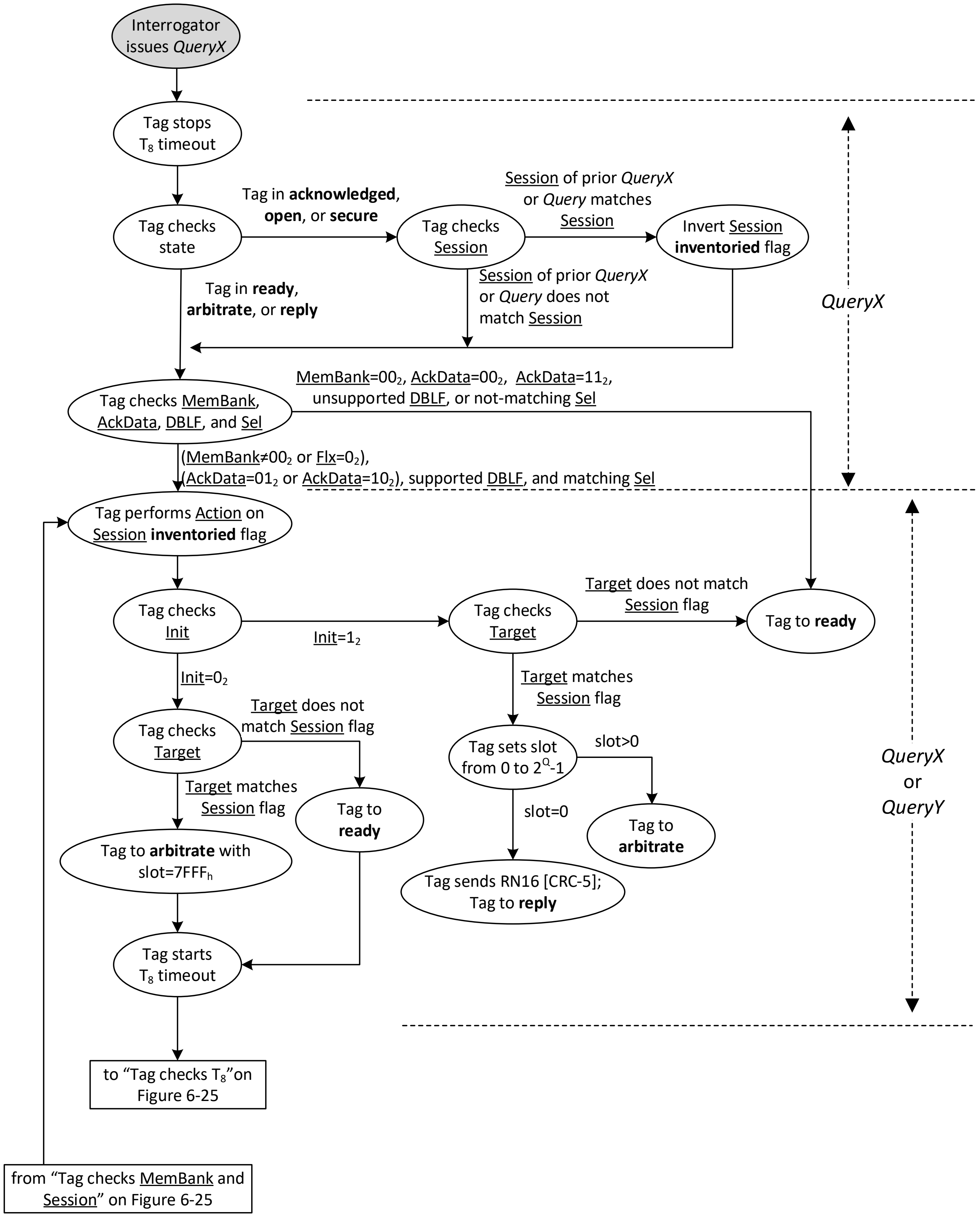
If a Tag in the **reply, acknowledged, open, or secured** states receives a *QueryY* shall stop (if active) the T_8 timeout, and if:

- the Session in the *QueryY* does not match the Session of the inventory round, then the Tag shall ignore the *QueryY* and remain in the same state;
- the Session in the *QueryY* matches the Session of the inventory round, then the Tag shall ignore the *QueryY* and transition to **ready**.

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

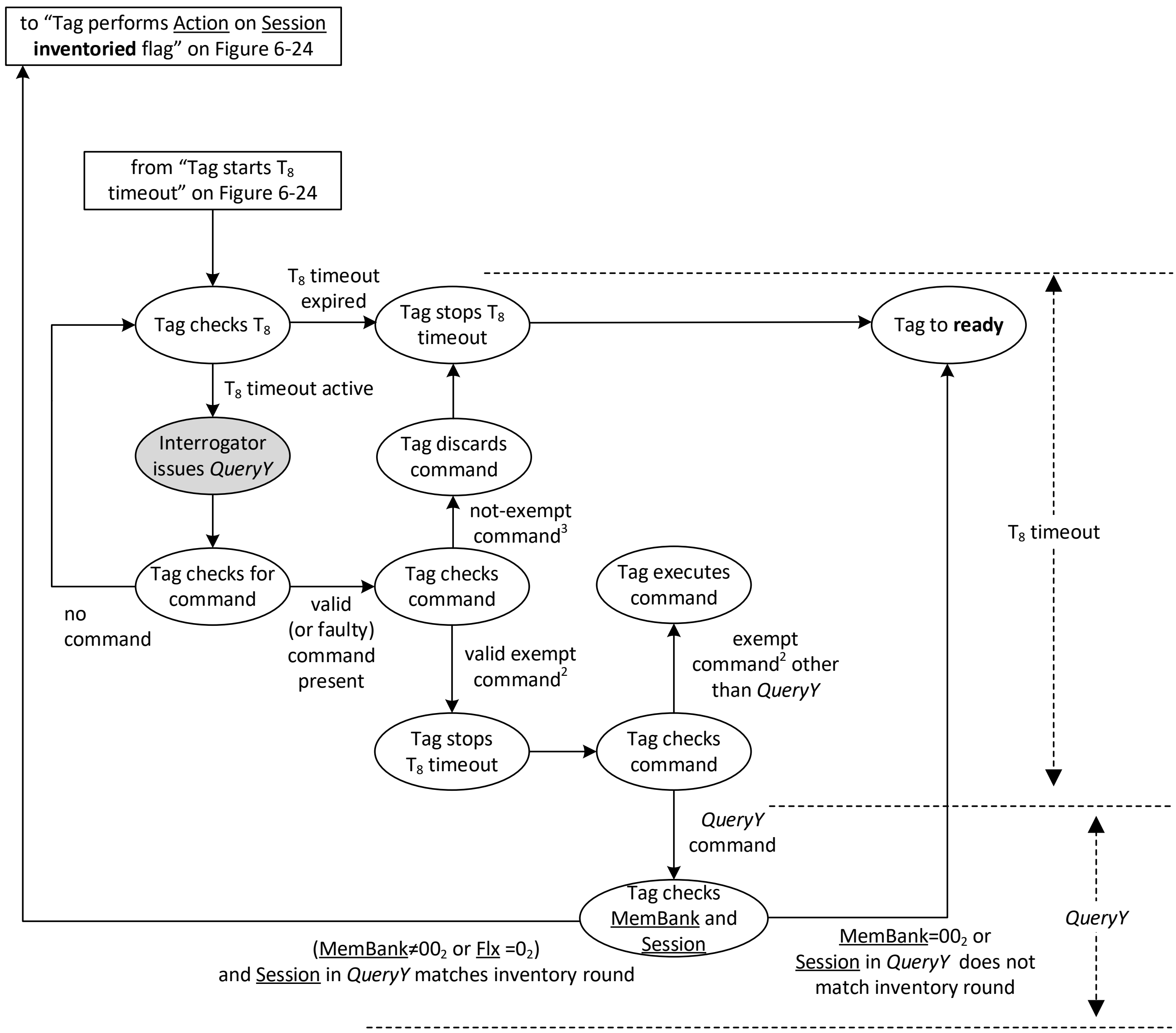
Page intentionally blank

Figure 6-24: *QueryX* / *QueryY* procedure[1] part 1



Note 1: Procedure does not account for a Tag in the **killed** state

Figure 6-25: *QueryX* / *QueryY* procedure[1] part 2



Note 1: Procedure does not account for a Tag in the **killed** state

Note 2: Exempt commands are *Select*, *Challenge*, *Query*, *QueryX*, and *QueryY*, and they are processed, as shown, if their delimiter is received before T_8 timeout expires.

Note 3: Commands that are not-exempt are ignored if their delimiter is received before T_8 timeout expires.

6.3.2.12.2.4 *QueryAdjust* (mandatory)

Interrogators and Tags shall implement the *QueryAdjust* command shown in *Table 6-42*. *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*, *6.3.2.12.2.1*, *6.3.2.12.2.2*, and *6.3.2.12.2.3*). If a Tag receives a *QueryAdjust* whose session number is different from the session number in the *Query* or *QueryX* that began the inventory round, the Tag shall ignore the command.
- UpDn (up or down) determines whether and how the Tag adjusts *Q*, as follows:
  - 110_2: Increment *Q* (i.e. *Q* = *Q* + 1).

- 000_2: No change to *Q*.
- 011_2: 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-34). If a Tag whose *Q* value is 15 receives a *QueryAdjust* with UpDn=110_2 then it shall change UpDn to 000_2 prior to executing the command; likewise, if a Tag whose *Q* value is 0 receives a *QueryAdjust* with UpDn=011_2 then it shall change UpDn to 000_2 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* or *QueryX* 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 6-29).

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 6-43 using the *immediate* reply type specified in 6.3.1.6.1; otherwise, the Tag shall remain silent. When a Tag replies to a *QueryAdjust*, the Tag shall backscatter a CRC-5 after the RN16 if a *QueryX* with ReplyCRC=1_2 began the inventory round; otherwise the Tag shall backscatter only the RN16. A Tag shall respond to a *QueryAdjust* only if it received a prior *Query*, *QueryX* (Init=1_2), or *QueryY* (Init=1_2).

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* or the *QueryX* that began the inventory round, and (ii) the Tag is not in the middle of a *Kill*- or *Access*-command sequence (see 6.3.2.12.3.5 or 6.3.2.12.3.7, 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* or *QueryX*, and that is not in the middle of a *Kill*- or *Access*-command sequence (see 6.3.2.12.3.5 or 6.3.2.12.3.7, 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 6-42: *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 6-43: Tag reply to a *QueryAdjust* command

	RN	CRC
# of bits	16	0 or 5
description	RN16	CRC-5

6.3.2.12.2.5 *QueryRep* (mandatory)

Interrogators and Tags shall implement the *QueryRep* command shown in Table 6-44. *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, 6.3.2.12.2.1, 6.3.2.12.2.2, and 6.3.2.12.2.3). If a Tag receives a *QueryRep* whose session number is different from the session number in the *Query* or *QueryX* that began the inventory 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 6-29).

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 6-45 using the *immediate* reply type specified in 6.3.1.6.1; otherwise the Tag shall remain silent. When a Tag replies to *QueryRep*, the Tag shall backscatter a CRC-5 after the RN16 if a *QueryX* with ReplyCRC=1_2 began the inventory round; otherwise the Tag shall backscatter only the RN16. A Tag shall respond to a *QueryRep* only if it received a prior *Query* or prior *QueryX*.

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* or *QueryX* that started the round, and (ii) the Tag is not in the middle of a *Kill*- or *Access*-command sequence (see 6.3.2.12.3.5 or 6.3.2.12.3.7, 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* or *QueryX*, and that is not in the middle of a *Kill*- or *Access*-command sequence (see 6.3.2.12.3.5 or 6.3.2.12.3.7, respectively), shall invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*, as appropriate) for the current session and transition to **ready**.

Table 6-44: *QueryRep* command

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

Table 6-45: Tag reply to a *QueryRep* command

	RN	CRC
# of bits	16	0 or 5
description	RN16	CRC-5

6.3.2.12.2.6 *ACK* (mandatory)

Interrogators and Tags shall implement the *ACK* command shown in Table 6-46. 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 6-29).

The Tag reply to a successful *ACK* shall be as shown in Table 6-47, using the immediate reply type specified in 6.3.1.6.1. As described in 6.3.2.1.2 and shown in Table 6-18, the reply may be truncated or include a concatenated response. A Tag in **reply** or **acknowledged** that receives an *ACK* with an incorrect RN16 shall return to **arbitrate** without responding. A Tag in **open** or **secured** that receives an *ACK* with an incorrect handle 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 6-18).

Table 6-46: *ACK* command

	Command	RN
# of bits	2	16
description	01	Echoed RN16 or handle

Table 6-47: Tag reply to a successful *ACK* command

	Reply
# of bits	21 to 33,328
description	See Table 6-18

6.3.2.12.2.7 *NAK* (mandatory)

Interrogators and Tags shall implement the *NAK* command shown in Table 6-48. 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 6-29).

A Tag shall not reply to a *NAK*.

Table 6-48: *NAK* command

	Command
# of bits	8
description	11000000

6.3.2.12.3 Access commands

The access command set comprises *Req_RN*, *Read*, *ReadVar*, *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-34.

6.3.2.12.3.1 *Req_RN* (mandatory)

Interrogators and Tags shall implement the *Req_RN* command shown in Table 6-49. *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).

In situations in which Tags reply with the same RN16 in the same slot, an Interrogator may misassociate one Tag's EPC from an *ACK* command obtained during inventory operations (see 6.3.2.10) with other data from another Tag obtained during access operations (see 6.3.2.11). To avoid this misassociation, this protocol recommends that the Interrogator:

- issue a new *ACK* with handle in the command from **open** or **secured** before issuing other access commands; then
- use the Tag reply from this new *ACK* instead of the Tag reply from a previous *ACK* with RN16 in the command from **reply**.

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 6-29).

A Tag's reply to a *Req_RN* shall be as shown in Table 6-50, using the *immediate* reply type specified in 6.3.1.6.1. The RN16 or handle are protected by a CRC-16.

Table 6-49: *Req_RN* command

	Command	RN	CRC
# of bits	8	16	16
description	11000001	Prior RN16 or handle	CRC-16

Table 6-50: Tag reply to a *Req_RN* command

	RN	CRC
# of bits	16	16
description	handle or new RN16	CRC-16

6.3.2.12.3.2 *Read* (mandatory)

Interrogators and Tags shall implement the *Read* command shown in Table 6-52. 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 (word pointer) 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=01_2 then a Tag shall backscatter the memory contents specified in Table 6-51.
  - if MemBank=10_2, part of TID memory is untraceably hidden (see 6.3.2.12.3.17), 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-34), depending on the Tag manufacturer's implementation.

Table 6-51: Tag *Read* reply when WordCount=00_h and MemBank=01_2

WordPtr 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 WordPtr 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 WordPtr and ending at the end of physical EPC memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2).
	Yes	No	EPC memory starting at WordPtr and ending at the end of physical EPC memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 if WordPtr 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 WordPtr and ending at the end of physical EPC memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 and XPC_W2 if WordPtr 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 WordPtr 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 *6.3.2.12.3.17*). XPC_W1 and XPC_W2 are never untraceably hidden.

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 6-29*).

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 6-53* 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-34*).

Table 6-52: *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	handle	CRC-16

Table 6-53: Tag reply to a successful *Read* command

	Header	Memory Words	RN	CRC
# of bits	1	Variable	16	16
description	0	Data	handle	CRC-16

6.3.2.12.3.3 *ReadVar* (mandatory)

Interrogators and Tags shall implement the *ReadVar* command shown in Table 6-55. A *ReadVar* allows an Interrogator to read part or all of a Tag's TID memory or File_0. *ReadVar* has the following fields:

- MemBank specifies whether the *ReadVar* accesses TID or File_0 in User memory. *ReadVar* commands shall apply to a single memory bank. Successive *ReadVars* may apply to different memory banks. If a Tag receives a *ReadVar* with MemBank=00_2 or 01_2 then the Tag shall treat the command's parameters as unsupported (see Table C-34).
- 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). If WordCount=00_h then the Interrogator shall set WordPtr=00_h. If WordCount=00_h and WordPtr≠00_h, then the Tag shall treat the command's parameters as unsupported (see Table C-34).
- WordCount specifies the maximum number of 16-bit words to read. A Tag shall reply with the number of words in memory up to the limit specified by WordCount. If WordCount=00_h then the Tag shall execute the *ReadVar* command as specified in Table 6-54.

An Interrogator shall prepend a *ReadVar* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *ReadVar* command in a *SecureComm* or *AuthComm* (see Table 6-29).

A Tag shall reply to a *ReadVar* using the *immediate* reply type specified in 6.3.1.6.1. If all memory words specified in a *ReadVar* exist, none are read-locked, all are traceable or the Interrogator has an asserted Untraceable privilege, and the Interrogator has read privileges for the memory bank (see 6.3.2.11.3), then a Tag's reply to a *ReadVar* shall be as shown in Table 6-56 comprising of the following fields:

- NumWords specifies the number of memory words that the Tag has available to backscatter in the reply. When Wordcount>0, a Tag shall backscatter NumWords where NumWords≤WordCount.
- MoreWords indicates whether additional words are available for reading, starting at memory location WordPtr+NumWords. If MoreWords=0000000_2 then the Tag reply includes the last word available. If MoreWords=1111111_2 then the Tag has additional words available. A Tag may indicate the contiguous quantity of additional words available at the memory location WordPtr+NumWords using an integer value between MoreWords=0000001_2 for one additional word available and MoreWords=1111110_2 for 126 additional words available.
- Parity is an even parity bit representing the number of 1's computed over NumWords||MoreWords. The number of 1's in the 16-bit length field NumWords||MoreWords||Parity shall be an even number, with NumWords||MoreWords||Parity=0000_h an allowed value.

- MemoryWords specifies the memory words backscattered by the Tag.

If a Tag has File_0, then the Tag shall support a UWC. UWC is a fixed or computed integer value between 0 and 255 that is Tag-manufacturer defined. A Tag shall have a UWC that is less than or equal to the number of memory words allocated to File_0. UWC specifies the value for NumWords when WordCount=00_h and MemBank=11_2.

Table 6-54: Tag reply to a *ReadVar* command when WordCount=00_h

MemBank	Interrogator Untraceable privilege	Untraceably hidden memory (see 6.3.2.12.3.17)	Tag Reply
10_2	Deasserted	Y	Either (i) backscatter MoreWords=0 and backscatter the traceable part of TID memory starting at WordPtr=00_h, or (ii) treat the command's parameters as unsupported (see Table C-34), depending on the Tag manufacturer's implementation
10_2	Asserted	Y	Backscatter TID memory starting at WordPtr=00_h and ending at the last word of the TID serialization (see 6.3.2.1.3)
10_2	Don't care	N	
11_2	Deasserted	Y	Treat the command's parameters as unsupported (see Table C-34)
11_2	Asserted	Y	Reply with NumWords as specified by UWC
11_2	Don't care	N	

The reply includes a CRC-16 calculated over the 0-bit, number of memory words backscattered, the number of additional words available for reading, the parity bit, memory words, and the Tag's handle. Otherwise, the Tag shall not execute the *ReadVar* and instead treat the command's parameters as unsupported (see Table C-34).

Table 6-55: *ReadVar* command

	Command	MemBank	WordPtr	WordCount	RN	CRC
# of bits	8	2	EBV	8	16	16
description	11011000	00: RFU 01: RFU 10: TID 11: File_0	Starting address pointer	Maximum number of words to read	handle	CRC-16

Table 6-56: Tag reply to a successful *ReadVar* command

	Header	NumWords	MoreWords	Parity	MemoryWords	RN	CRC
# of bits	1	8	7	1	16×NumWords	16	16
descrip-tion	0	Number of MemoryWords	Availability of additional words	Even parity bit	Data	handle	CRC-16

6.3.2.12.3.4 *Write* (mandatory)

Interrogators and Tags shall implement the *Write* command shown in Table 6-57. *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 XORing 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-34*).

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 unwritable 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-34*).

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 6-29*).

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 6-57: *Write* command

	Command	MemBank	WordPtr	Data	RN	CRC
# of bits	8	2	EBV	16	16	16
description	11000011	00: Reserved 01: EPC 10: TID 11: User	Address pointer	RN16 ⊗ word to be written	handle	CRC-16

6.3.2.12.3.5 *Kill* (mandatory)

Interrogators and Tags shall implement the *Kill* command shown in *Table 6-58*. *Kill* allows an Interrogator to permanently disable a Tag. To kill a Tag, an Interrogator shall follow the kill procedure shown in *Figure 6-26*. A Tag shall implement the password-based kill sequence shown in the left-side branch of the kill procedure in *Figure 6-26*. 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 6-26*. *Kill* has the following fields:

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

A *Kill* contains 3 RFU bits. An Interrogator shall set these bits to 000_2. 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 6-29*).

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 (password bits 31-16) XORed with an RN16, and the second containing the 16 LSBs of the Tag's kill password (password bits 15-0) XORed with a subsequent RN16. Each XOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be XORed 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 is as improper (see

Table C-34), unless the intervening command is a *Query* or *QueryX*, in which case the Tag shall execute the *Query* or *QueryX*; if the intervening command is a *Query* or *QueryX*, then the Tag shall invert its **inventoried** flag if the Session parameter in the *Query* or *QueryX* 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, backscattering an error code, and remaining in its current state. See *Figure 6-26*.

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 6-59. The reply shall use the TRext value specified in the *Query* or *QueryX* 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 6-26*.

If a Tag observes a properly formatted password-based *Kill*-command sequence but the kill fails (e.g., 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 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 6-24*) 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_2 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 6-14*, 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 6-24*), 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.

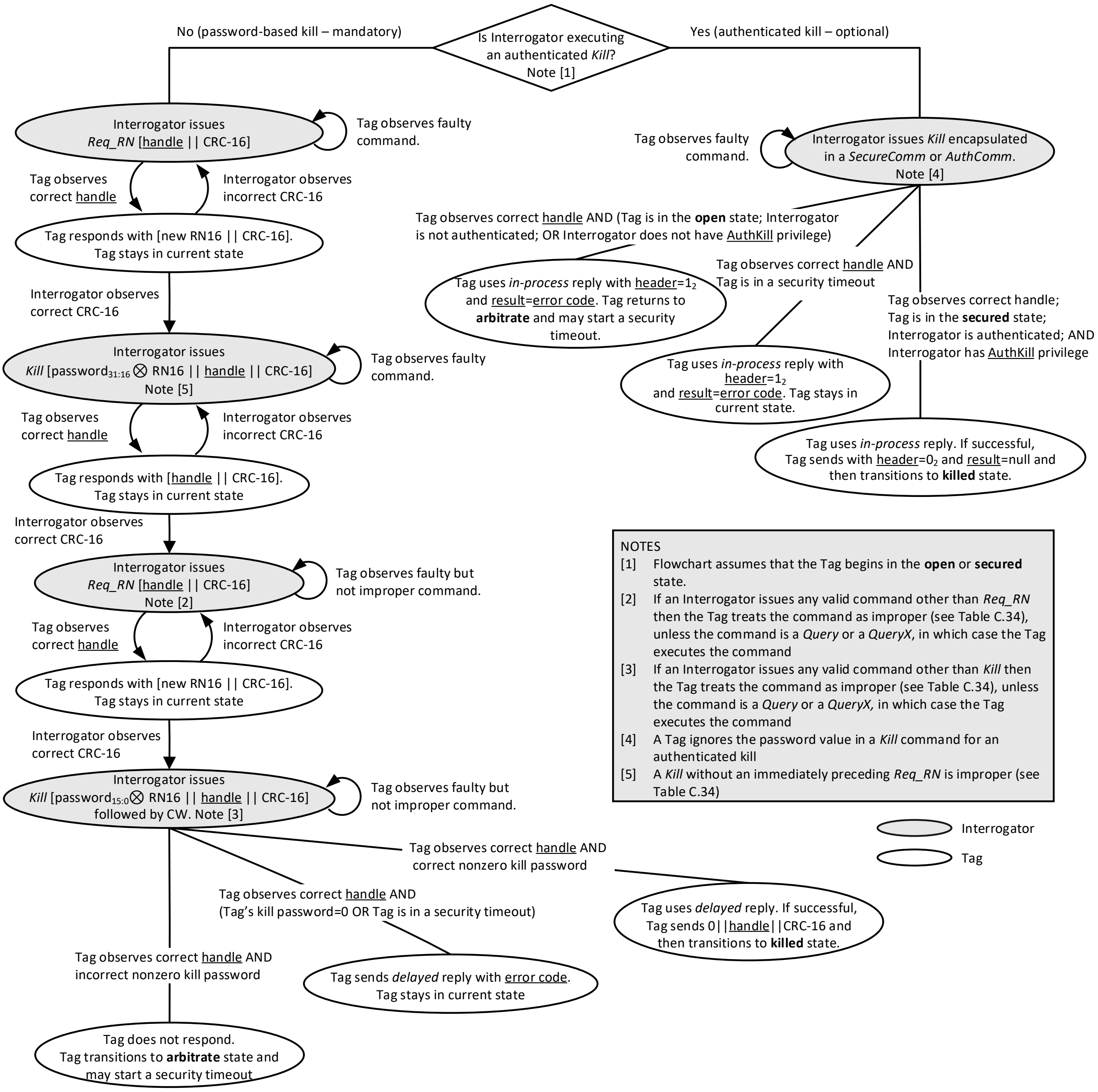
Table 6-58: *Kill* command

	Command	Password	RFU	RN	CRC
# of bits	8	16	3	16	16
description	11000100	Password-based kill: (16 MSBs or 16 LSBs of kill password) ⊗ RN16 Authenticated kill: any 16-bit value	000	handle	CRC-16

Table 6-59: Tag reply to the first *Kill* command

	RN	CRC
# of bits	16	16
description	handle	CRC-16

Figure 6-26: *Kill* procedure



6.3.2.12.3.6 *Lock* (mandatory)

Interrogators and Tags shall implement the *Lock* command shown in *Table 6-60* and *Figure 6-27*. *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_2: Ignore the associated Action field and retain the current lock setting.
  - Mask=1_2: 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_2: Deassert lock for the associated memory location.
  - Action=1_2: Assert lock or permalock for the associated memory location.

The functionality of the various Action fields is described in *Table 6-61*.

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 non-existent memory bank, nonexistent File_0, or non-existent password then a Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see *Table C-34*).

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≥0 of User memory in an unwritable state. *Table 6-66* specifies how a Tag reacts to a *Lock* targeting File_0 that follows a prior *BlockPermalock* (with Read/Lock=1_2), 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-34*). 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 *6.3.2.12.3.17*.

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 6-27*, 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-34*). 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 $FFFFF_h$, and shall backscatter an error code for any payload other than $FFFFF_h$.

A Tag in the **secured** state shall permit an Interrogator to write or erase memory locations with (pwd-write=1_2 AND permalock=0_2) or (pwd-read/write=1_2 AND permalock=0_2) 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 6-29).

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 6-60: *Lock* command

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

Figure 6-27: *Lock* payload and usage

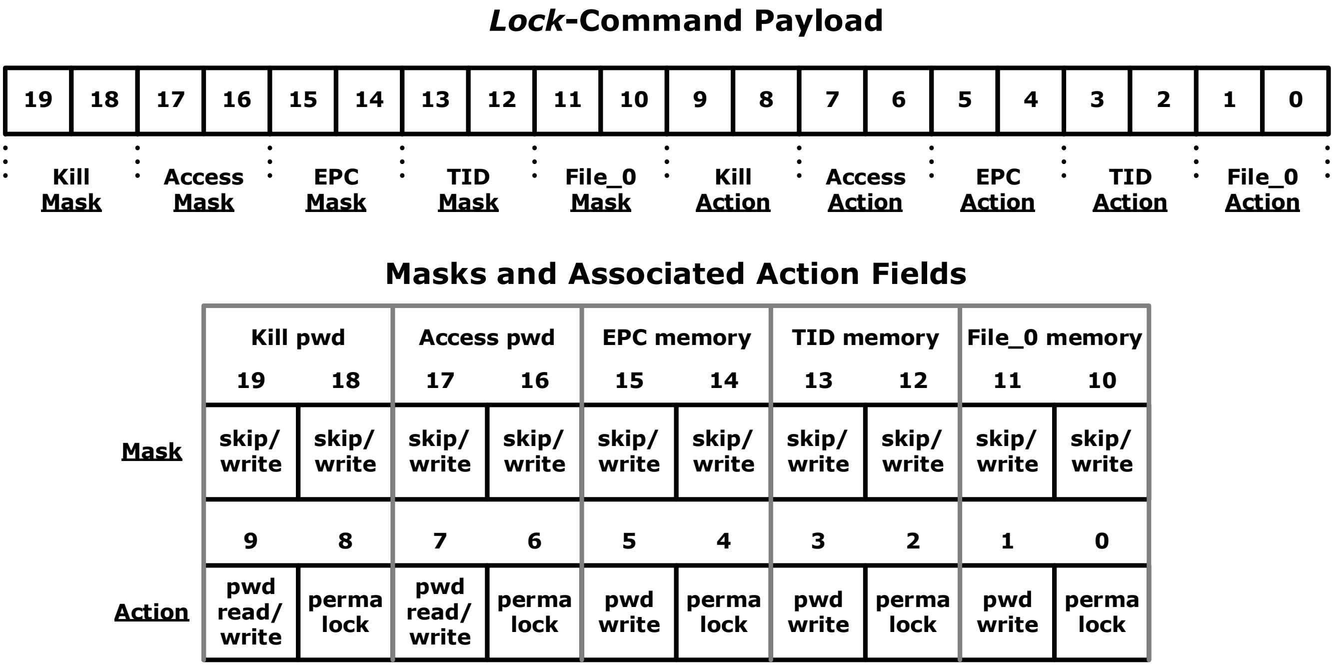


Table 6-61: *Lock* Action-field functionality

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

6.3.2.12.3.7 *Access* (optional)

Interrogators and Tags may implement an *Access* command; if they do, they shall implement it as shown in *Table 6-62*. *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 XORed with an RN16.

To access a Tag, an Interrogator shall follow the multi-step procedure outlined in *Figure 6-28*. Briefly, an Interrogator issues two *Access* commands, the first containing the 16 MSBs of the Tag's access password (password bits 31-16) XORed with an RN16, and the second containing the 16 LSBs of the Tag's access password (password bits 15-0) XORed with a different RN16. Each XOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be XORed 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-34*), unless the intervening command is a *Query* or *QueryX*, in which case the Tag shall execute the *Query* or *QueryX*; if the intervening command is a *Query* or *QueryX*, then the Tag shall invert its **inventoried** flag if the Session parameter in the *Query* or *QueryX* 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 6-29*).

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 6-63*. 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 shall backscatter its handle to acknowledge that it has executed the command successfully and shall transition to the **secured** state; otherwise the Tag shall 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.

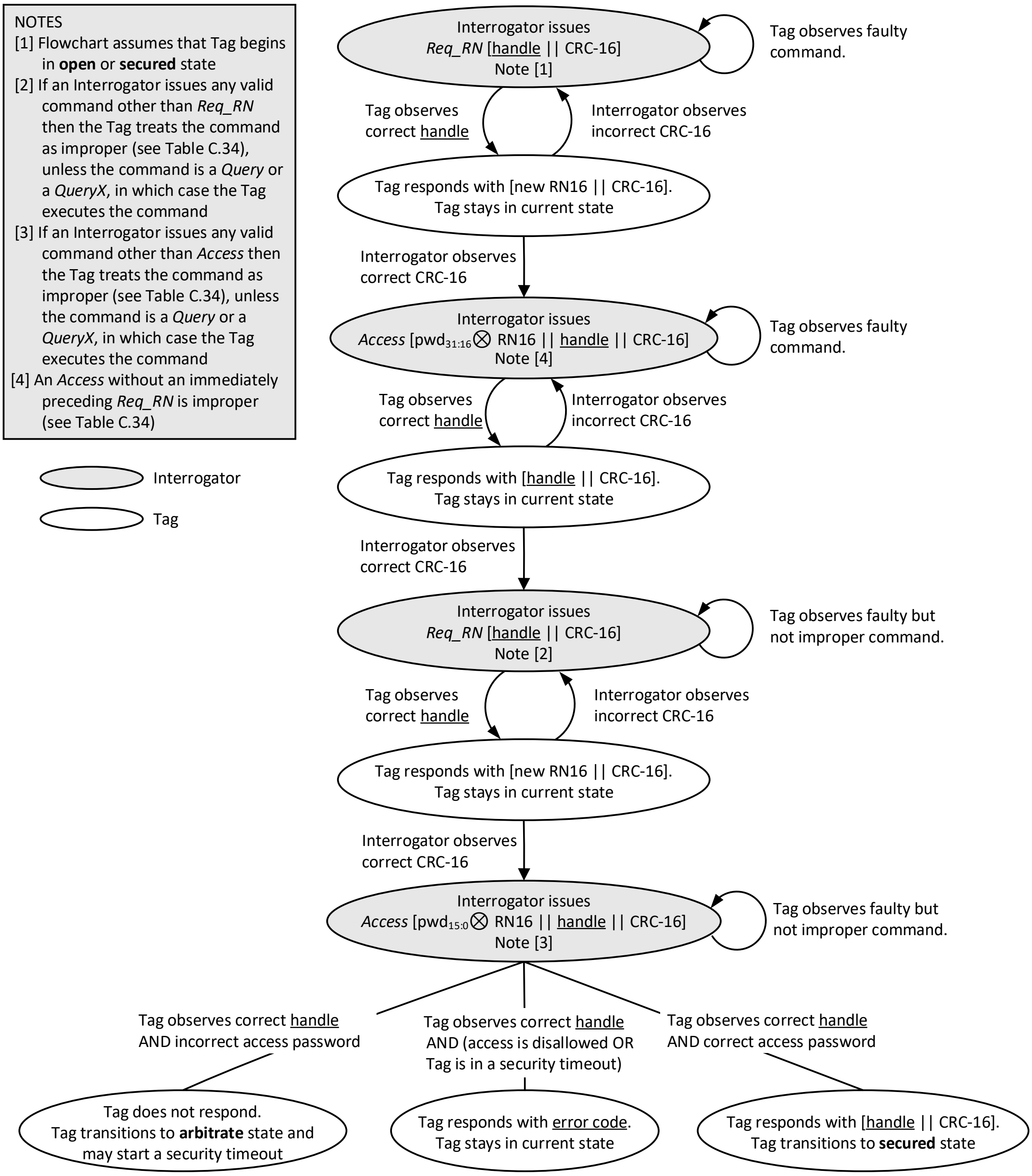
Table 6-62: *Access* command

	Command	Password	RN	CRC
# of bits	8	16	16	16
description	11000110	(16 MSBs or 16 LSBs of access password) ⊗ RN16	handle	CRC-16

Table 6-63: Tag reply to an *Access* command

	RN	CRC
# of bits	16	16
description	handle	CRC-16

Figure 6-28: *Access* procedure



6.3.2.12.3.8 *BlockWrite* (optional)

Interrogators and Tags may implement a *BlockWrite* command; if they do, they shall implement it as shown in *Table 6-64*. *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 *BlockWrite*s 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 unwritable 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-34*).

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 6-29*).

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 6-64: *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	handle	CRC-16

6.3.2.12.3.9 *BlockErase* (optional)

Interrogators and Tags may implement a *BlockErase* command; if they do, they shall implement it as shown in *Table 6-65*. *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 *BlockErase*s 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 unwritable 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-34*).

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 6-29*).

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 6-65: *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	handle	CRC-16

6.3.2.12.3.10 *BlockPermalock* (optional)

Interrogators and Tags may implement a *BlockPermalock* command; if they do, they shall implement it as shown in *Table 6-67*. *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 unwritable 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 writable or unwritable state. *Table 6-66* 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_2).

Table 6-66: Precedence for *Lock* and *BlockPermalock* targeting File_0

First Command			Second Command	Tag Action and Response to 2nd Command
	pwd-write	permalock		
Lock	0	0	*BlockPermalock* (Read/Lock=1_2)	Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.10
	0	1		Do not execute the *BlockPermalock*; respond with an error code (Table C-34, unsupported parameters)
	1	0		Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.10
	1	1		Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.10
First Command	**Second Command**			**Tag Action and Response to 2nd Command**
		pwd-write	**permalock**	
BlockPermalock (Read/Lock=1_2)	*Lock*	0	0	Implement the *Lock*, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.6
		0	1	Implement the *Lock*, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.6
		1	0	Implement the *Lock*, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.6
		1	1	Implement the *Lock*; respond as described in 6.3.2.12.3.6

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_2 (User memory); if a Tag receives a *BlockPermalock* with MemBank≠11_2 then it shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C-34). 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_2: 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_2.
  - Read/Lock=1_2: 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-34).
- Mask specifies which memory blocks a Tag permalocks. Mask depends on the Read/Lock bit as follows:
  - Read/Lock=0_2: The Interrogator shall omit Mask from the *BlockPermalock*.

- Read/Lock=1_2: 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=0_2: Retain the current permalock setting for the corresponding memory block.
  - Mask=1_2: 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_2, 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-34*). 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-34*). A Tag shall treat as invalid a *BlockPermalock* in which Read/Lock=0_2 but Mask is not omitted, or a *BlockPermalock* in which Read/Lock=1_2 but Mask has a length not equal to 16×BlockRange bits (see *Table C-34*).

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-34*). 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 non-existent 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 6-29*).

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-34*).

If Read/Lock=0_2 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 6-68* 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-34*, unsupported parameters) rather than the reply shown in *Table 6-68*. The Tag's reply when Read/Lock=0_2 shall use the preamble specified by the TRext value in the *Query* or *QueryX* that initiated the inventory round.

If Read/Lock=1_2 then a Tag shall reply to a *BlockPermalock* using the *delayed* reply specified in *6.3.1.6.2*.

Table 6-67: *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	Mask starting address, specified in units of 16 blocks	Mask range, specified in units of 16 blocks	0: retain current permalock setting 1: assert perma-lock	handle	CRC-16

Table 6-68: Tag reply to a successful *BlockPermalock* command with Read/Lock=0_2

	Header	Data	RN	CRC
# of bits	1	Variable	16	16
description	0	Permalock bits	handle	CRC-16

6.3.2.12.3.11 *Authenticate* (optional)

Interrogators and Tags may implement the *Authenticate* command; if they do, they shall implement it as shown in *Table 6-69*. *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 (send reply) specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer. If SenRep=1_2 a Tag shall backscatter its response; if SenRep=0_2 a Tag shall store the response in the ResponseBuffer. Unless otherwise specified by the cryptographic suite, a Tag shall support SenRep=1_2 and may support SenRep=0_2.
- IncRepLen specifies whether a Tag omits or includes Length in its reply. If IncRepLen=0_2 then a Tag omits length from its reply; if IncRepLen=1_2 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_2. 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-34*). 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 6-29*).

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 SenRep, 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-34). If a Tag in the **secured** state receives an *Authenticate* that begins a new multi-step authentication sequence, then the Tag shall transition to the **open** state, discontinue using and reset the current cryptographic engine, and begin the new multi-step authentication sequence.

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 6-69: *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 Length from reply 1: include Length in reply	CSI	length of Message	Message (depends on CSI)	handle	CRC-16

6.3.2.12.3.12 *AuthComm* (optional)

Interrogators and Tags may implement the *AuthComm* command; if they do, they shall implement it as shown in Table 6-70. *AuthComm* allows authenticated communications from R=>T by encapsulating another command and typically also a MAC in the *AuthComm's* Message field. Table 6-29 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_2 then the Tag omits length from its reply; if IncRepLen=1_2 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 00_2. 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-34). 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 6-29) shall not execute the *AuthComm* and instead treat the command's parameters as unsupported (see Table C-34).

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-34).

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_2 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_2 and header=0_2, indicating that the *AuthComm* executed successfully, and length=0003_h and result=0_2, 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-34).

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 6-70: *AuthComm* command

	Command	RFU	IncRepLen	Message	RN	CRC
# of bits	8	2	1	Variable	16	16
description	11010111	00_2	0: omit Length from reply 1: include Length in reply	Message	handle	CRC-16

6.3.2.12.3.13 *SecureComm* (optional)

Interrogators and Tags may implement the *SecureComm* command; if they do, they shall implement it as shown in Table 6-71. *SecureComm* allows encrypted communications from R=>T by encapsulating another, encrypted command in the *SecureComm's* Message field. Table 6-29 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. Unless otherwise specified by the cryptographic suite, a Tag shall support SenRep=1_2 and may support SenRep=0_2. If a Tag receives a *SecureComm* specifying an unsupported value of SenRep then the Tag shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see Table C-34).
- IncRepLen specifies whether the Tag omits or includes Length in its reply. If IncRepLen=0_2 then the Tag omits length from its reply; if IncRepLen=1_2 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 00_2. 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-34*). 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 6-29*) shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see *Table C-34*).

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-34*).

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_2 and SenRep=1_2 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_2 and header=0_2, indicating the *SecureComm* executed successfully, and Length=0003_h and result=0_2, indicating that the *Lock* completed successfully. Alternatively, if SenRep=0_2 then the reply will be as shown in *Table 6-14* with done=1_2 and header=0_2, indicating the *SecureComm* executed successfully, and Length=0003_h and result=null, indicating that the *Lock* completed successfully and result (0_2) 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 6-71: *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 Length from reply 1: include Length in reply	length of Message	Message	handle	CRC-16

6.3.2.12.3.14 *KeyUpdate* (optional)

Interrogators and Tags may implement the *KeyUpdate* command; if they do, they shall implement it as shown in *Table 6-72*. *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. Unless otherwise specified by the cryptographic suite, a Tag shall support SenRep=1_2 and may support SenRep=0_2. If a Tag receives a *KeyUpdate* specifying an unsupported value of SenRep then the Tag shall not execute the *KeyUpdate* and instead treat the command's parameters as unsupported (see *Table C-34*).
- IncRepLen specifies whether the Tag omits or includes Length in its reply. If IncRepLen=0_2 then the Tag omits length from its reply; if IncRepLen=1_2 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 00_2. 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-34*). 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 6-29*). 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-34*). 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.

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-34*).

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 6-72: *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 Length from reply 1: include Length in reply	KeyID	length of Message	Message	handle	CRC-16

6.3.2.12.3.15 *TagPrivilege* (optional)

Interrogators and Tags may implement the *TagPrivilege* command; if they do, they shall implement it as shown in *Table 6-74*. *TagPrivilege* allows an Interrogator to read or modify the Tag privileges in *Table 6-23* or *Table 6-24* 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_2 then the Tag omits length from its reply; if IncRepLen=1_2 then the Tag includes Length in its reply.
- Action specifies whether the Interrogator is reading privileges or modifying them. Action=0_2 indicates read; Action=1_2 indicates modify.
- Target specifies whether the Interrogator is targeting the access password or a key. If Target=0_2 then the Tag reads or modifies the access-password privileges; if Target=1_2 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 6-23* or *Table 6-24* when an Interrogator is modifying a privilege (i.e. when Action=1_2).

A *TagPrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. 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-34*). 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_2 (i.e. specifying the access password).

An authenticated Interrogator shall encapsulate a *TagPrivilege* in a *SecureComm* or *AuthComm* (see *Table 6-29*). 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-34*).

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-34*).

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_2 in a *TagPrivilege* then it may use any value for privilege. A Tag shall ignore privilege when Action=0_2.

If an Interrogator specifies Target=0_2 in a *TagPrivilege* then it may use any value for the KeyID. If Tag receives a *TagPrivilege* with Target=0_2 then it shall ignore the value that the Interrogator supplies for KeyID.

Upon receiving an executable *TagPrivilege* with Action=1_2 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 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-34*).

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 6-73* for Action=0_2 or Action=1_2. The response includes Target, the Interrogator-supplied KeyID, and the current privileges (newly written if Action=1_2 and the Tag wrote the new privileges successfully).

Table 6-73: *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 Length from reply 1: include Length in reply	0: read 1: modify	0: access pwd 1: key	KeyID	privilege	handle	CRC -16

Table 6-74: Tag reply to a successful *TagPrivilege* command

	Target	KeyID	Privileges
# of bits	1	8	16
description	0: access pwd 1: key	KeyID	privilege

6.3.2.12.3.16 *ReadBuffer* (optional)

Interrogators and Tags may implement the *ReadBuffer* command; if they do, they shall implement it as shown in *Table 6-75*. *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. If a Tag implements the *ReadBuffer* command, then the Tag shall support WordPtr=000_h. The Tag may support non-zero WordPtr values. If a cryptographic suite supported by the Tag specifies support of non-zero WordPtr values, then the Tag shall support these specified non-zero WordPtr values. If a Tag receives a *ReadBuffer* specifying an unsupported value of WordPtr then the Tag shall not execute the *ReadBuffer* and instead treat the command's parameters as unsupported (see *Table C-34*).
- 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. If a Tag implements the *ReadBuffer* command, then the Tag shall support BitCount=000_h. The Tag may support non-zero BitCount values. If a cryptographic suite specifies support of non-zero BitCount values, then the Tag shall support these specified non-zero BitCount values. If a Tag receives a *ReadBuffer* specifying an unsupported value of BitCount then the Tag shall not execute the *ReadBuffer* and instead treat the command's parameters as unsupported (see *Table C-34*).

A *ReadBuffer* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. 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-34*). 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 6-29*).

If a Tag implements a ResponseBuffer then the Tag may implement the *ReadBuffer* command. If the Tag supports the *Challenge* command with the value Immed=0_2, then the Tag shall implement the

ReadBuffer command. If the Tag supports the *Authenticate*, *SecureComm*, or *KeyUpdate* commands with SenRep=0_2, then the 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_2 and the memory bits specified in the *ReadBuffer* exist then the Tag's reply shall be as shown in *Table 6-76* 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-34*, unsupported parameters) within time T_1 in *Table 6-16* rather than the reply shown in *Table 6-76*.

Table 6-75: *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	handle	CRC-16

Table 6-76: Tag reply to a successful *ReadBuffer* command

	Header	Data Bits	RN	CRC
# of bits	1	Variable	16	16
description	0	data	handle	CRC-16

6.3.2.12.3.17 *Untraceable* (optional)

Interrogators and Tags may implement the *Untraceable* command; if they do, they shall implement it as shown in *Table 6-77*. *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 $21C_h$ 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=0_2 then a Tag exposes EPC memory. If show/hide=1_2 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 *ACK*s 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 Tag 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-34*), 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-34*). 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-34*).

An unauthenticated Interrogator may issue an *Untraceable* without encapsulation. An authenticated Interrogator shall encapsulate an *Untraceable* in a *SecureComm* or *AuthComm* (see *Table 6-29*). 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-34*).

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-34*). 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-34*).

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-34*). 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 **RUM** then the untraceability status of User memory does not change **RUM**.

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-34*).

Table 6-77: *Untraceable* command

	# of bits	description
Command	16	11100010 00000000
RFU	2	00
U	1	0: deassert **U** in XPC_W1 1: assert **U** in XPC_W1
EPC	6	MSB (show/hide): 0: show memory above EPC 1: hide memory above EPC 5 LSBs (Length): new EPC length field (new **L** bits)
TID	2	00: hide none 01: hide some 10: hide all 11: RFU
User	1	0: view 1: hide
Range	2	00: normal 01: toggle temporarily 10: reduced 11: RFU
RN	16	handle
CRC	16	CRC-16

6.3.2.12.3.18 *FileOpen* (optional)

Interrogators and Tags may implement the *FileOpen* command; if they do, they shall implement it as shown in *Table 6-78*. *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 (file number) specifies the file to be opened.

A *FileOpen* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives a *FileOpen* with nonzero RFU bits or that specifies FileNum=1111111111_2 (RFU FileNum) shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see *Table C-34*). 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 6-29*). 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-34*).

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 0000_2 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 6-25* and *Table 6-26*).

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 6-79*. The reply includes a header (a 0-bit), FileNum, FileType, FileSize, BlockSize, IntPriv (interrogator privilege), 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 6-25* and *Table 6-26*). 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_2, otherwise it shall set LastFile to 1_2. 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 6-79*. 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-34*), 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 6-78: *FileOpen* command

	Command	RFU	FileNum	RN	CRC
# of bits	8	2	10	16	16
description	11010011	00	Which file to open	handle	CRC-16

Table 6-79: 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	FileType	File size in blocks	Block size in words	Interrogator's file privilege	0: not max FileNum 1: max FileNum	handle	CRC-16

6.3.2.12.3.19 *FileList* (optional)

Interrogators and Tags may implement the *FileList* command; if they do, they shall implement it as shown in *Table 6-80*. *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_2 then the Tag omits length from its reply; if IncRepLen=1_2 then the Tag includes Length in its reply.
- FileNum identifies the starting file for which the Interrogator is requesting information, inclusive.
- AddlFiles (additional files) identifies the number of additional files for which the Interrogator is requesting information. For example, if FileNum=4 and AddlFiles=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 00_2. A Tag in the **open** or **secured** states that receives a *FileList* with nonzero RFU bits or that specifies FileNum=1111111111_2 (RFU FileNum) shall not execute the *FileList* and instead treat the command's parameters as unsupported (see *Table C-34*). 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 6-29*). 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-34*).

An unauthenticated Interrogator may issue a *FileList* without encapsulation to a Tag in the **open** or **secured** state.

An Interrogator shall not specify AddlFiles=FF_h. If a Tag receives a *FileList* with AddlFiles=FF_h then the Tag shall behave as though it had received a *FileList* with AddlFiles=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 6-81 and includes a Message for each file for which the Interrogator requested information. The response includes the number of Message, 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 6-25* and *Table 6-26*). 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 1111111111_2. If a Tag receives a *FileList* with an unsupported FileNum, or AddlFiles 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-34*).

Table 6-80: *FileList* command

	Command	RFU	SenRep	IncRepLen	FileNum	AddlFiles	RN16	CRC
# of bits	16	2	1	1	10	8	16	16
description	11100010 00000001	00	0: store 1: send	0: omit Length from reply 1: include Length in reply	First file number	Total number of additional files	handle	CRC-16

Table 6-81: 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	Allocatable memory in blocks

6.3.2.12.3.20 *FilePrivilege* (optional)

Interrogators and Tags may implement the *FilePrivilege* command; if they do, they shall implement it as shown in *Table 6-83*. *FilePrivilege* allows an Interrogator to read or modify file privileges (*Table 6-25* or *Table 6-26*) 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_2 then the Tag omits length from its reply; if IncRepLen=1_2 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 6-25* and *Table 6-26*.

A *FilePrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. 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-34*). 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 6-29*). 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-34*).

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 6-82* 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 6-82: Action field behavior for a *FilePrivilege*

Action	Privilege a Tag Assigns to the Currently Open File	Tag Ignores	Required Privilege	Tag Reply to the *FilePrivilege*	Reference
000_2	–	KeyID and privilege	Any	FileNum and the **open**-state file privilege	*Table 6-84*, Action=000_2 or 001_2
001_2	privilege to the **open** state for FileNum	KeyID	File superuser	–	
010_2	–	KeyID and privilege	Any	FileNum and the access-password file privilege	*Table 6-84*, Action=010_2 or 011_2
011_2	privilege to the access password for FileNum	KeyID	File superuser to assign privilege for the access password DecFilePriv to decrement privilege for supplied access password	–	
100_2	–	privilege	Any	FileNum, KeyID, and KeyID's file privilege	*Table 6-84*, Action=100_2 or 101_2
101_2	privilege to KeyID for FileNum	–	File superuser to assign privilege for any Key_N DecFilePriv to decrement privilege for supplied key	–	
110_2	–	KeyID and privilege	Any	FileNum, NumKeys, and a KeyID/privilege pair for each key	*Table 6-84*, Action=110_2 or 111_2
111_2	privilege to all KeyID for FileNum	KeyID	File superuser	–	

As shown in *Table 6-82*, 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 6-82* 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 6-82* 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=001_2, 011_2, 101_2, or 111_2 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. Note that, if Action=111_2 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 6-84*.

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-34*).

Table 6-83: *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 Length from reply 1: include Length in reply	000: Read **open** state 001: Modify **open** state 010: Read access pwd 011: Modify access pwd 100: Read KeyID 101: Modify KeyID 110: Read all keys 111: Modify all keys	KeyID	privilege	handle	CRC-16

Table 6-84: Tag reply to a successful *FilePrivilege* with indicated Action fields

Action=000_2 or 001_2

	FileNum	Privilege
# of bits	10	4
description	Currently open file	**open** state privilege

Action=010_2 or 011_2

	FileNum	Privilege
# of bits	10	4
description	Currently open file	access password privilege

Action=100_2 or 101_2

	FileNum	KeyID	Privilege
# of bits	10	8	4
description	Currently open file	KeyID	KeyID privilege

Action=110_2 or 111_2

	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

6.3.2.12.3.21 *FileSetup* (optional)

Interrogators and Tags may implement the *FileSetup* command; if they do, they shall implement it as shown in *Table 6-85*. *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_2 then the Tag omits length from its reply; if IncRepLen=1_2 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 00_2. 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-34*). 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 6-29*). 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-34*).

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 6-27* 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 6-86.

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 6-86. 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-34*).

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-34*).

Table 6-85: *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 Length from reply 1: include Length in reply	FileType	Requested file size, in blocks	handle	CRC-16

Table 6-86: Tag reply to a successful *FileSetup* command[1]

	FileNum	FileType	FileSize
# of bits	10	8	10
description	Currently open file	FileType	Current file size, in blocks

Note 1: see also *Table 6-27*

7. Battery Assisted Passive (BAP) Interrogator Talks First systems (optional)

Please refer to Chapter 7 clause in ISO/IEC 18000-63 for the complete description of BAP Tags if needed.

8. Sensor support (optional)

For this document, Tags are passive, meaning that they receive all of their operating energy from the Interrogator's RF signal (see Introduction). Consequently, BAP (battery assisted passive) Tags with one or more sensors are not included in this document as they are not passive Tags. Please refer to Chapter 7 clause in ISO/IEC 18000-63 for the complete description of BAP Tags if needed.

8.1 Applicability

In case an Interrogator or Tag supports any command, response or feature of Clause 8 then this Interrogator or Tag shall support all mandatory commands, responses or features and it may support all optional commands, responses or features of Clause 8.

In case an Interrogator or Tag does not support any command, response or feature of Clause 8 then Clause 8 does not apply for this device.

8.2 Overview

This subclause describes an optional extension that adds sensor support for Tags. A Tag with sensor support shall implement XPC_W1 (see *6.3.2.1.2.5*) and shall utilize the Sensor Alarm indicator (**SA**) bit 214_h in XPC_W1 (see *6.3.2.1.2.5*) for the sensor interface and/or be as specified by one or more of the three classes of sensor supported by this protocol and summarized in this subclause:

- **Snapshot Sensor**: A Snapshot Sensor is the most basic implementation of a sensor and it is not required to be user programmed. A Snapshot Sensor generates a sensor measurement essentially in real-time as an Interrogator inventories or reads the Tag. The sensor measurement may occur during power-up or on demand from an Interrogator and the Tag delivers the sensor measurement via XPC_W2. A Tag having a Snapshot Sensor may be implemented in either passive or BAP Tags and do not require a Real Time Clock (RTC) to support sensor operations. Snapshot Sensors are defined in *8.7*.
- **Simple Sensor**: A Simple Sensor is programmed at source and is not required to be user programmed. A Tag having a Simple Sensor shall be implemented only in BAP Tags and shall have a Real Time Clock (RTC) to support sensor operations. Simple Sensors are defined in *8.5*.
- **Full Function Sensor**: Full Function Sensors provide greater flexibility than Simple Sensors, by:
    - supporting a greater variety of sensor types and measurement spans
    - enabling thresholds to be set within a wider range
    - capturing and processing different types of data

  A Tag having a Full Function Sensor shall only be implemented in BAP Tags and shall have a Real Time Clock (RTC) to support sensor operations.

  Full Function Sensors are defined in *8.6*.

Tags may be equipped with one or more sensors. If a Tag has sensor support for any of the sensor classes then the **XI** bit of the EPC memory shall be asserted and XPC_W1 shall be supported, see also *6.3.2.1.2*.

If a violation of at least one alarm condition in at least one of the attached sensors occurs, then the Sensor Alarm indicator (**SA**) bit 214_h shall be set to 1_2 in XPC_W1 (see *6.3.2.1.2.5*).

If it has the capability, a Tag may support any combination of Snapshot Sensors, Full Function Sensors, and one Simple Sensor. Each sensor shall be fully compliant with the class of sensor.

8.3 Real Time Clock (RTC)

An RTC is not included in this document as it is used only for Simple Sensors and Full Function Sensors which cannot be implemented on passive Tags. Refer to the corresponding subclause in ISO/IEC 18000-63 for the complete description of an RTC if needed.

8.4 HandleSensor command (optional)

HandleSensor command is used only for Full Function Sensors which cannot be implemented on passive Tags. Refer to corresponding subclause in ISO/IEC 18000-63 for complete description of *HandleSensor* if needed.

8.5 Simple Sensors

Although called "Simple Sensors", the devices are required to support features common to any type of sensor device. The Simple Sensor has to monitor the environmental characteristic for which it is designed, take samples at defined intervals, compare and process against criteria, and report its status.

Simple Sensors may be implemented on a Tag. If a Tag implements a Simple Sensor then the Simple Sensor indicator (**SS**) bit 215_h shall be set to 1_2 in XPC_W1 (see *6.3.2.1.2.5*).

Simple Sensors cannot be implemented on passive Tags. Refer to the corresponding subclause in ISO/IEC 18000-63 for the complete description of a Simple Sensor if needed.

8.6 Full Function Sensors and Sensor Directory System

Full Function Sensors may be implemented on a Tag. If a Tag implements a Full Function Sensor then the Full Function Sensor indicator (**FS**) bit 216_h shall be set to 1_2 in XPC_W1 (see *6.3.2.1.2.5*). Full Function Sensors cannot be implemented on passive Tags. Refer to the corresponding subclause in ISO/IEC 18000-63 for the complete description of a Full Function Sensor if needed.

8.7 Snapshot Sensors

8.7.1 General

A Snapshot Sensor is the most basic implementation of a sensor. A Snapshot Sensor generates a sensor measurement essentially in real-time as an Interrogator inventories or reads the Tag. The sensor measurement may occur during power-up or on demand from an Interrogator.

Snapshot Sensors may be implemented on a Tag. If a Tag implements a Snapshot Sensor then the Snapshot Sensor indicator (**SN**) bit 217_h shall be set to 1_2 in XPC_W1 (see *6.3.2.1.2.5*). A Tag may implement as many Snapshot Sensors as desired. Any Snapshot Sensor implemented shall be in accordance to *Table 8-1* and *Table 8-2*.

Table 8-1: Snapshot Sensor data formats

Sensor type	Sensor	Units	Sensor data
0000_2	Vendor Defined (VenDef)	VenDef	2-bit data type || 10-bit data value as follows: 00_2 || 11111111112 (error) 01_2 || 10 discrete bits (e.g. switches) 10_2 || 10-bit unsigned integer (e.g. A2D converter) 11_2 || 10-bit signed integer (e.g. A2D converter)
0001_2	Temperature	°C	12-bit signed integer
0010_2	Relative Humidity	%	12-bit unsigned integer
0011_2	Barometric Pressure	hPa (mbar)	12-bit unsigned integer
0100_2	Light	lux	1-bit scale factor || 11-bit data value as follows: 0_2 || 11-bit unsigned integer (scale factor is 0.5) 1_2 || 11-bit unsigned integer (scale factor is 50)
0101_2	Voltage	V	2-bit scale factor || 10-bit data value as follows: 00_2 || 10-bit unsigned integer (scale factor is 0.001) 01_2 || 10-bit unsigned integer (scale factor is 0.01) 10_2 || 10-bit unsigned integer (scale factor is 0.1) 11_2 || 10-bit unsigned integer (scale factor is 1)
0110_2	Magnetic Field	mT (10G)	12-bit signed integer
0111_2	Angular Position	°	12-bit unsigned integer
1000_2	Rotational Speed	rpm	1-bit scale factor || 11-bit data value as follows: 0_2 || 11-bit signed integer (scale factor is 1) 1_2 || 11-bit signed integer (scale factor is 10)
1001_2	Weight	kg	2-bit scale factor || 10-bit data value as follows: 00_2 || 10-bit unsigned integer (scale factor is 0.1) 01_2 || 10-bit unsigned integer (scale factor is 1) 10_2 || 10-bit unsigned integer (scale factor is 10) 11_2 || 10-bit unsigned integer (scale factor is 100)
1010_2	Liquid Flow	ml/min	2-bit scale factor || 10-bit data value as follows: 00_2 || 10-bit signed integer (scale factor is 0.00002) 01_2 || 10-bit signed integer (scale factor is 0.002) 10_2 || 10-bit signed integer (scale factor is 0.2) 11_2 || 10-bit signed integer (scale factor is 20)
1011_2	Gas Flow	l_n/min	1-bit scale factor || 11-bit data value as follows: 0_2 || 11-bit unsigned integer (scale factor is 0.000625) 1_2 || 11-bit unsigned integer (scale factor is 0.0625)
1100_2	Accelerometer	G (9.81m/s^2)	2-bit axis || 10-bit data value as follows: 00_2 || 10-bit signed integer (X-axis measurement) 01_2 || 10-bit signed integer (Y-axis measurement) 10_2 || 10-bit signed integer (Z-axis measurement) 11_2 || 10-bit signed integer (not used)
1101_2	Gyroscope	°/s	2-bit axis || 10-bit data value as follows: 00_2 || 10-bit signed integer (X-axis measurement) 01_2 || 10-bit signed integer (Y-axis measurement) 10_2 || 10-bit signed integer (Z-axis measurement) 11_2 || 10-bit signed integer (not used)

Sensor type	Sensor	Units	Sensor data
1110_2	Magnetometer	µT	2-bit axis || 10-bit data value as follows: 00_2 || 10-bit signed integer (X-axis measurement) 01_2 || 10-bit signed integer (Y-axis measurement) 10_2 || 10-bit signed integer (Z-axis measurement) 11_2 || 10-bit signed integer (not used)
1111_2	VenDef Supplemental Data	N/A	12-bit unsigned integer for number of words that follow which is supplemental sensor information for the preceding sensor

Table 8-2: Snapshot Sensor data values

Sensor type	Sensor	Scale factor	Minimum value		Maximum value		Error value
			binary	scaled	binary	scaled	
0000_2	VenDef Data Type = 00_2	N/A	N/A	N/A	N/A	N/A	1111111111_2
	VenDef Data Type = 01_2	N/A	0000000000_2	N/A	1111111111_2	N/A	Use Data Type = 00_2
	VenDef Data Type = 10_2	1	0000000000_2	0	1111111111_2	1023	Use Data Type = 00_2
	VenDef Data Type = 11_2	1	1000000001_2	-511	0111111111_2	511	Use Data Type = 00_2
0001_2	Temperature	0,0625	100000000001_2	−127,9375	011111111111_2	127,9375	100000000000_2
0010_2	Relative Humidity	0,0625	000000000000_2	0	011001111111_2	100	111111111111_2
0011_2	Barometric Pressure	0,125	000000000000_2	600	111111111110_2	1111,75	111111111111_2
0100_2	Light	0,5	00000000000_2	0	11111111110_2	1023	11111111111_2
		50	00000000000_2	0	11111111110_2	102,300	11111111111_2
0101_2	Voltage	0,001	0000000000_2	0	1111111110_2	1,022	1111111111_2
		0,01	0000000000_2	0	1111111110_2	10,22	1111111111_2
		0,1	0000000000_2	0	1111111110_2	102,2	1111111111_2
		1	0000000000_2	0	1111111110_2	1022	1111111111_2
0110_2	Magnetic Field	0,0625	100000000001_2	−127,9375	011111111111_2	127,9375	100000000000_2
0111_2	Angular Position	360 / 4095	000000000000_2	0	111111111110_2	359,912	111111111111_2
1000_2	Rotational Speed	1	10000000001_2	-1023	01111111111_2	1023	10000000000_2
		10	10000000001_2	-10,230	01111111111_2	10,230	10000000000_2
1001_2	Weight	0,1	0000000000_2	0	1111111110_2	102,2	1111111111_2
		1	0000000000_2	0	1111111110_2	1022	1111111111_2
		10	0000000000_2	0	1111111110_2	10,220	1111111111_2
		100	0000000000_2	0	1111111110_2	102,200	1111111111_2
1010_2	Liquid Flow	0,00002	1000000001_2	−0,01022	0111111111_2	0,01022	1000000000_2
		0,002	1000000001_2	-1,022	0111111111_2	1,022	1000000000_2
		0,2	1000000001_2	-102,2	0111111111_2	102,2	1000000000_2
		20	1000000001_2	-10,220	0111111111_2	10,220	1000000000_2
1011_2	Gas Flow	0,000625	00000000000_2	0	11111111110_2	1,26625	11111111111_2

Sensor type	Sensor	Scale factor	Minimum value		Maximum value		Error value
			binary	scaled	binary	scaled	
		0,0625	000000000000_2	0	11111111110_2	126,625	11111111111_2
1100_2	Accelerometer	0,03125	1000000001_2	−15,96875	0111111111_2	15,96875	1000000000_2
1101_2	Magnetometer	2	1000000001_2	−1022	0111111111_2	1022	1000000000_2
1111_2	VenDef Supplemental Data	N/A	N/A	N/A	N/A	N/A	N/A

8.7.2 Initiating Snapshot Sensor measurements

A tag shall support at least one of the defined methods to initiate a Snapshot Sensor measurement and it may support more than one of the defined methods. There are three defined methods to initiate a Snapshot Sensor measurement:

- Tag initiated during its power-up sequence and the sensor measurement typically occurs during T_s (see *Table 6-6*) and the Snapshot Sensor information shall be available starting with the first inventory round which includes the Tag.
- Interrogator initiated on demand by using a *Select* command on XPC_W2, i.e. with MemBank=01_2, Pointer=8420_h (EBV format for 220_h), Length=10_h, and Mask as defined in *Table 8-3* and *Table 8-4*. The sensor measurement occurs during T_4 (see *Table 6-16*) and the Snapshot Sensor information shall be available starting with the next inventory round which includes the Tag if it remains energized. If T_4 is shorter than the sensor measurement time, then the Tag might not remain energized. It is recommended to use a *Select* command to first create a population of Snapshot Sensor Tags and then use a *Select* command on XPC_W2 followed by a T_4 time of 20 ms.
- Interrogator initiated on demand by using *Write* or *BlockWrite* command to XPC_W2, i.e. with MemBank=01_2, WordPtr=22_h, and Data as defined in *Table 8-3* and *Table 8-4* for *Write*, and MemBank=01_2, WordPtr=22_h, WordCount=1_h, and Data as defined in *Table 8-3* and *Table 8-4* for *BlockWrite*. An Interrogator may initiate a measurement regardless of the lock or permalock status of EPC memory. The sensor measurement occurs during T_5 (see *Table 6-16*) and the Snapshot Sensor information shall be available for a subsequent *Read* of XPC_W2 and/or starting with the next inventory round which includes the Tag.

Table 8-3: XPC_W2 for Interrogator initiated Snapshot Sensor measurements

XPC_W2 bit assignments															
MSB															LSB
220_h	221_h	222_h	223_h	224_h	225_h	226_h	227_h	228_h	229_h	$22A_h$	$22B_h$	$22C_h$	$22D_h$	$22E_h$	$22F_h$
Cmd	Selected Sensors [14:0]														
Sensor Command	sensor type = 1110_2	sensor type = 1101_2	sensor type = 1100_2	sensor type = 1011_2	sensor type = 1010_2	sensor type = 1001_2	sensor type = 1000_2	sensor type = 0111_2	sensor type = 0110_2	sensor type = 0101_2	sensor type = 0100_2	sensor type = 0011_2	sensor type = 0010_2	sensor type = 0001_2	sensor type = 0000_2

Table 8-4: XPC_W2 for Interrogator initiated Snapshot Sensor measurements

Hex	Descriptor	Settings
220	Cmd Sensor Command	**When used by *Select* command:** 0: A Tag shall not initiate measurements for any Selected Sensors. 1: A Tag shall initiate a measurement for all Selected Sensors that are implemented by the Tag and if this method is supported by the selected Snapshot Sensor. **When used by *Write* command or *BlockWrite* command:** 0: Vendor defined command. 1: A Tag shall initiate a measurement for all Selected Sensors that are implemented by the Tag and only if this method is supported by the selected Snapshot Sensor.
221:22F	Selected Sensors	**When used by *Select* command:** A Tag shall be considered matching when any of the Selected Sensor types are implemented on the Tag. A Tag shall be considered non-matching if either there are no selected Sensor Types or none of the Selected Sensor types are implemented on the Tag. **When used by *Write* command or *BlockWrite* command:** If Sensor Command = 0 then it is a vendor defined command and the meaning of these bits are vendor defined. If Sensor Command = 1 then a Tag shall initiate a measurement for all Selected Sensors that are implemented by the Tag and only if this method is supported by the selected Snapshot Sensor.

8.7.3 Reporting Snapshot Sensor Information

A Tag having a Snapshot Sensor is not required to support the exact range or resolution as in *Table 8-2*. A Tag shall convert its sensor measurements accordingly to report its Snapshot Sensor information using the defined format based on the sensor type in *Table 8-2*. A Tag shall also report when a measurement error occurs using the defined value in *Table 8-2* for the sensor type.

A Tag shall report Snapshot Sensor information to an Interrogator via XPC_W2. Consequently, the Tag shall set **XEB** = 0_2 when Snapshot Sensor information is not available meaning that XPC_W2 = 0000_h, and the Tag shall set **XEB** = 1_2 when Snapshot Sensor information is available meaning that XPC_W2 ≠ 0000_h. Note that when Snapshot Sensor information is available then it is included in the Tag reply to an *ACK* (see *Table 6-18*) and it may also be obtained using a *Read* command to directly read XPC_W2.

Snapshot Sensor information in XPC_W2 shall use the following formats:

- XPC_W2 = (sensor type || sensor data) as defined in *Table 8-1*. This format is used for reporting a sensor measurement from only one Snapshot Sensor and the sensor type must be in the range 0000_2 to 1011_2. An example for this type of reporting is the following:
  - XPC_W1 = 8100_h indicating a Snapshot Sensor Tag with sensor information available
  - XPC_W2 = 1190_h indicating a temperature sensor with a measurement of 25°C
- XPC_W2 = (11_2 || Memory Bank || Word Address). This format is used for reporting sensor measurements from a multi-dimension sensor, or a multi-sensor Tag, or when one or more sensors provides additional vendor defined supplemental data. An Interrogator may obtain the Snapshot Sensor information using a *Read* command with MemBank = Memory Bank, WordPtr = Word Address converted to EBV format, and WordCount = 00_h. The Snapshot Sensor information consists of a sequence of words, each having the format (sensor type || sensor data) as defined in *Table 8-1*. The sequence of words is terminated with a null value = 0000_h which does not correspond to any valid value for (sensor type || sensor data). Sensor type = 1111_2 is reserved for a Tag to include optional vendor defined supplemental data for a Snapshot Sensor. The supplemental data consists of a sequence of words and the number of data words is specified following the sensor type, i.e. (sensor type = 1111_2 || number of data words). An example for this type of reporting is the following:
  - XPC_W1 = 8100_h indicating a Snapshot Sensor Tag with sensor information available.
  - XPC_W2 = $E100_h$ indicating sensor data is in TID memory starting at word address 100_h.
  - TID word 100_h = 1190_h indicates a temperature sensor with measurement of 25°C.

- TID word 101_h = $F002_h$ indicates 2 words follow of supplemental data for the temperature sensor.
- TID word 102_h = 1234_h is supplemental data word #1 for the temperature sensor.
- TID word 103_h = 5678_h is supplemental data word #2 for the temperature sensor.
- TID word 104_h = 2500_h indicates a relative humidity sensor with measurement of 80 %.
- TID word 105_h = 0000_h is the null value terminator indicating the end of sensor data.

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

Total number of entries (exponential format)	Total number of entries (decimal format)	Extension (bit)	Block data (bits)	Extension (bit)	Block data (bits)	Extension (bit)	Block data (bits)
--	0	0	0000000	--	--	--	--
2^0	1	0	0000001	--	--	--	--
$2^7 – 1$	127	0	1111111	--	--	--	--
2^7	128	1	0000001	0	0000000	--	--
$2^{14} – 1$	16383	1	1111111	0	1111111	--	--
2^{14}	16384	1	0000001	1	0000000	0	0000000

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. See 5.3 for notation conventions.

B.1 Present state: Ready

Table B-1: Ready state-transition table

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried** flag; AND matching **SL** flag	backscatter new RN16	**reply**
	slot≠0; matching **inventoried** flag; AND matching **SL** flag	–	**arbitrate**
	otherwise	–	**ready**
QueryX[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[3]	stop T_8 timeout and (backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2)	**reply**
	Init=1_2; slot≠0; AND matching **inventoried** flag[3]	stop T_8 timeout	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[3]	start T_8 timeout and set slot=$7FFF_h$	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[3]	start T_8 timeout	**ready**
	otherwise[4]	stop T_8 timeout	**ready**
QueryY[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[5]	stop T_8 timeout and (backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2)	**ready**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[5]	stop T_8 timeout	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[5]	start T_8 timeout and set slot=$7FFF_h$	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[5]	start T_8 timeout	**ready**
	otherwise[6]	stop T_8 timeout	**ready**
QueryRep	all	–	**ready**
QueryAdjust	all	–	**ready**
ACK	all	–	**ready**
NAK	all	–	**ready**
Req_RN	all	–	**ready**
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**ready**
Read	all	–	**ready**

Command	Condition	Action	Next State
ReadVar	all	–	**ready**
Write	all	–	**ready**
Kill	all	–	**ready**
Lock	all	–	**ready**
Access	all	–	**ready**
BlockWrite	all	–	**ready**
BlockErase	all	–	**ready**
BlockPermalock	all	–	**ready**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	all	–	**ready**
AuthComm	all	–	**ready**
SecureComm	all	–	**ready**
ReadBuffer	all	–	**ready**
KeyUpdate	all	–	**ready**
Untraceable	all	–	**ready**
FileSetup	all	–	**ready**
FileOpen	all	–	**ready**
FilePrivilege	all	–	**ready**
TagPrivilege	all	–	**ready**
FileList	all	–	**ready**
Faulty	invalid[2]	–	**ready**

Note 1: *Query* or *QueryX* start a new round and may change the session. *Query*, *QueryX* with Init=1_2, and *QueryY* with Init=1_2 also instruct a Tag to load a new random value into its slot counter.

Note 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.

Note 3: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 4: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2 ; unsupported DBLF; OR MemBank=00_2.

Note 5: Additional condition: delimiter received within $T_{8(max)}$; (MemBank≠00_2 OR Flx=0_2); AND Session matches inventory round.

Note 6: Otherwise condition: delimiter received when T_8 timeout inactive; Session does not match inventory round; (Init=1_2 AND not-matching **inventoried** flag); OR MemBank=00_2.

B.2 Present state: Arbitrate

Table B-2: Arbitrate state-transition table

Command	Condition	Action	Next State
Query[1,2]	slot=0; matching **inventoried** flag; AND matching **SL** flag	backscatter new RN16	**reply**
	slot≠0; matching **inventoried** flag; AND matching **SL** flag	–	**arbitrate**
	otherwise	–	**ready**
QueryX[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout and (backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2)	**reply**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout and set slot=$7FFF_h$	**arbitrate**
	Init=0_2; not-matching **inventoried** flag; AND additional condition[4]	start T_8 timeout	**ready**
	otherwise[5]	stop T_8 timeout	**ready**
QueryY[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[6]	stop T_8 timeout and (backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2)	**ready**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[6]	stop T_8 timeout	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[6]	start T_8 timeout and set slot=$7FFF_h$	**arbitrate**
	Init=0_2; not-matching **inventoried** flag; AND additional condition[6]	start T_8 timeout	**ready**
	otherwise condition[7]	stop T_8 timeout	**ready**
QueryRep	Session matches inventory round AND (slot=0 after decrementing slot counter)	decrement slot counter and (backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round)	**reply**
	Session matches inventory round AND (slot≠0 after decrementing slot counter)	decrement slot counter	**arbitrate**
	Session does not match inventory round	–	**arbitrate**
QueryAdjust [2]	Session matches inventory round AND slot=0	backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round	**reply**
	Session matches inventory round AND slot≠0	–	**arbitrate**
	Session does not match inventory round	–	**arbitrate**
ACK	all	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	all	–	**arbitrate**

Command	Condition	Action	Next State
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**arbitrate**
Read	all	–	**arbitrate**
ReadVar	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**
BlockErase	all	–	**arbitrate**
BlockPermalock	all	–	**arbitrate**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	all	–	**arbitrate**
AuthComm	all	–	**arbitrate**
SecureComm	all	–	**arbitrate**
ReadBuffer	all	–	**arbitrate**
KeyUpdate	all	–	**arbitrate**
Untraceable	all	–	**arbitrate**
FileSetup	all	–	**arbitrate**
FileOpen	all	–	**arbitrate**
FilePrivilege	all	–	**arbitrate**
TagPrivilege	all	–	**arbitrate**
FileList	all	–	**arbitrate**
Faulty	invalid[3]	–	**arbitrate**

Note 1: *Query* or *QueryX* starts a new round and may change the session.

Note 2: *Query*, *QueryX* with Init=1_2, *QueryY* with Init=1_2, and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

Note 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.

Note 4: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 5: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2.

Note 6: Additional condition: delimiter received within $T_{8(max)}$; (MemBank≠00_2 OR Flx=0_2); AND Session matches inventory round.

Note 7: Otherwise condition: delimiter received when T_8 timeout inactive; Session does not match inventory round; (Init=1_2 AND not-matching **inventoried** flag); OR MemBank=00_2.

B.3 Present state: Reply

Table B-3: Reply state-transition table

Command	Condition	Action	Next State
Query[1,2]	slot=0; matching **inventoried** flag; AND matching **SL** flag	backscatter new RN16	**reply**
	slot≠0; matching **inventoried** flag; AND matching **SL** flag	–	**arbitrate**
	Otherwise	–	**ready**
QueryX[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout and (backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2)	**reply**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[4]	start T_8 timeout and set slot=$7FFF_h$	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[4]	start T_8 timeout	**ready**
	otherwise[5]	stop T_8 timeout	**ready**
QueryY[2]	Session matches inventory round	stop T_8 timeout	**ready**
	Session does not match inventory round	stop T_8 timeout	**reply**
QueryRep	Session matches inventory round	–	**arbitrate**
	Session does not match inventory round	–	**reply**
QueryAdjust[2]	Session matches inventory round AND slot=0	backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round	**reply**
	Session matches inventory round AND slot≠0	–	**arbitrate**
	Session does not match inventory round	–	**reply**
ACK	correct RN16	See *Table 6-18*	**acknowledged**
	incorrect RN16	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	all	–	**arbitrate**
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**reply**
Read	all	–	**arbitrate**
ReadVar	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**

Command	Condition	Action	Next State
BlockErase	all	–	**arbitrate**
BlockPermalock	all	–	**arbitrate**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	all	–	**arbitrate**
AuthComm	all	–	**arbitrate**
SecureComm	all	–	**arbitrate**
ReadBuffer	all	–	**arbitrate**
KeyUpdate	all	–	**arbitrate**
Untraceable	all	–	**arbitrate**
FileSetup	all	–	**arbitrate**
FileOpen	all	–	**arbitrate**
FilePrivilege	all	–	**arbitrate**
TagPrivilege	all	–	**arbitrate**
FileList	all	–	**arbitrate**
T_2 timeout	See *Figure 6-18* and *Table 6-16*	–	**arbitrate**
Faulty	invalid[3]	–	**reply**

Note 1: *Query* or *QueryX* starts a new round and may change the session.

Note 2: *Query*, *QueryX* with Init=1_2, *QueryY* with Init=1_2, and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

Note 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.

Note 4: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 5: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2.

B.4 Present state: Acknowledged

Table B-4: Acknowledged state-transition table

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2] flag; AND matching **SL** flag	backscatter new RN16 and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	slot≠0; matching **inventoried**[2] flag; AND matching **SL** flag	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**arbitrate**
	otherwise	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**ready**
QueryX[1,2]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout; (backscatter new RN16 if ReplyCRC=0_2, or backscatter new RN16||CRC-5 if ReplyCRC=1_2); and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[4]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[4]	start T_8 timeout; set slot=$7FFF_h$; and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[4]	start T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
	otherwise[5]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
QueryY[1]	Session matches inventory round	stop T_8 timeout	**ready**
	Session does not match inventory round	stop T_8 timeout	**acknowledged**
QueryRep	Session matches inventory round	transition **inventoried**[2] from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**acknowledged**
QueryAdjust[2]	Session matches inventory round	transition **inventoried**[2] from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**acknowledged**
ACK	correct RN16	See Table 6-18	**acknowledged**
	incorrect RN16	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	correct RN16 AND access password≠0	backscatter handle	**open**

Command	Condition	Action	Next State
	correct RN16 AND access password=0	backscatter handle	**secured**
	incorrect RN16	–	**acknowledged**
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**acknowledged**
Read	all	–	**arbitrate**
ReadVar	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**
BlockErase	all	–	**arbitrate**
BlockPermalock	all	–	**arbitrate**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	all	–	**arbitrate**
AuthComm	all	–	**arbitrate**
SecureComm	all	–	**arbitrate**
ReadBuffer	all	–	**arbitrate**
KeyUpdate	all	–	**arbitrate**
Untraceable	all	–	**arbitrate**
FileSetup	all	–	**arbitrate**
FileOpen	all	–	**arbitrate**
FilePrivilege	all	–	**arbitrate**
TagPrivilege	all	–	**arbitrate**
FileList	all	–	**arbitrate**
T_2 timeout	See *Figure 6-18* and *Table 6-16*	–	**arbitrate**
Faulty	invalid[3]	–	**acknowledged**

Note 1: *Query* or *QueryX* starts a new round and may change the session. *Query*, *QueryX* with Init=1_2, and *QueryY* with Init=1_2 also instructs a Tag to load a new random value into its slot counter.

Note 2: As described in *6.3.2.10*, a Tag transitions its **inventoried** flag prior to evaluating the condition.

Note 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.

Note 4: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 5: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2.

B.5 Present state: Open

Table B-5: Open state-transition table

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2] flag; AND matching **SL** flag	backscatter new RN16 and transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**reply**
	slot≠0; matching **inventoried**[2] flag; AND matching **SL** flag	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**arbitrate**
	otherwise	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**ready**
QueryX[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[8]	stop T_8 timeout; (backscatter new RN16 if ReplyCRC=0_2, or backscatter new RN16||CRC-5 if ReplyCRC=1_2); and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[8]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[8]	start T_8 timeout; set slot=$7FFF_h$; and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[8]	start T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
	otherwise[9]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
QueryY[1]	Session matches inventory round	stop T_8 timeout	**ready**
	Session does not match inventory round	stop T_8 timeout	**open**
QueryRep	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**open**
QueryAdjust	Session matches inventory round	transition **inventoried**[2] from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**open**
ACK	correct handle	See *Table 6-18*	**open**
	incorrect handle	–	**open**
NAK	all	–	**arbitrate**
Req_RN	all	backscatter new RN16	**open**
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**open**
Read	all	backscatter data	**open**
ReadVar	all	backscatter data	**open**

Command	Condition	Action	Next State
Write	all	backscatter header when done	**open**
Kill (see also *Figure 6-26*)	supported security timeout AND within timeout	backscatter error code	**open**
	1st *Kill* received AND password-based kill	backscatter handle	**open**
	2nd *Kill* received; password-based kill; AND correct nonzero kill password	backscatter header when done	**killed**
	2nd *Kill* received; password-based kill; AND incorrect nonzero kill password	may start security timeout	**arbitrate**
	2nd *Kill* received; password-based kill; AND kill password=0	backscatter error code	**open**
	authenticated kill	backscatter error code and may start security timeout	**arbitrate**
Lock	all	–	**open**
Access (see also *Figure 6-28*)	supported security timeout AND within timeout	backscatter error code	**open**
	1st *Access* received	backscatter handle	**open**
	2nd *Access* received AND correct access password	backscatter handle	**secured**
	2nd *Access* received AND incorrect access password	may start security timeout	**arbitrate**
BlockWrite	all	backscatter header when done	**open**
BlockErase	all	backscatter header when done	**open**
BlockPermalock	all	–	**open**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	supported security timeout AND within timeout	backscatter error code	**open**
	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**open** or **secured** [3]
	executable AND SenRep=1_2	backscatter response when done	**open** or **secured** [3]
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
	new authentication sequence	reset cryptographic engine	**open**
AuthComm	supported security timeout AND within timeout	backscatter error code	**open**
	executable AND prior Tag authentication	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	**open**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**

Command	Condition	Action	Next State
SecureComm	supported security timeout AND within timeout	backscatter error code	**open**
	executable; prior Tag authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	see encapsulated command
	executable; prior Tag authentication; AND SenRep=1_2	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	**open**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
ReadBuffer	**C**=1_2	backscatter data	**open**
	C=0_2	backscatter error code	**open**
KeyUpdate	all	–	**open**
Untraceable	all	–	**open**
FileSetup	all	–	**open**
FileOpen	all	close current file; open requested file; and backscatter file info	**open**
FilePrivilege	all	–	**open**
TagPrivilege	all	–	**open**
FileList	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**open**
	executable AND SenRep=1_2	backscatter response when done	**open**
Faulty	unsupported parameters [4]	backscatter error code	**open**
	incorrect handle [5]	none unless specified by cryptographic suite	**open**
	improper [6]	–	**arbitrate**
	invalid [7]	none unless specified by cryptographic suite	**open**

Note 1: *Query* or *QueryX* starts a new round and may change the session. *Query*, *QueryX* with Init=1_2, and *QueryY* with Init=1_2 also instructs a Tag to load a new random value into its slot counter.

Note 2: As described in *6.3.2.10*, a Tag transitions its **inventoried** flag prior to evaluating the condition.

Note 3: See cryptographic suite.

Note 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 WordPtr, BitCount, SenRep or 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 6-29*); (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.

Note 5: "Incorrect handle" shall mean (1) an *ACK* command with incorrect handle or (2) 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.

Note 6: "Improper" shall mean a command (except *Req_RN*, *Query*, or *QueryX*) 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 6-26* and *Figure 6-28*).

Note 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.

Note 8: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 9: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2.

B.6 Present state: Secured

Table B-6: Secured state-transition table

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2] flag; AND matching **SL** flag	backscatter new RN16 and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	slot≠0; matching **inventoried**[2] flag; AND matching **SL** flag	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**arbitrate**
	Otherwise	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**ready**
QueryX[1]	Init=1_2; slot=0; matching **inventoried** flag; AND additional condition[7]	stop T_8 timeout; (backscatter new RN16 if ReplyCRC=0_2, or backscatter new RN16||CRC-5 if ReplyCRC=1_2); and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	Init=1_2; slot≠0; matching **inventoried** flag; AND additional condition[7]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; matching **inventoried** flag; AND additional condition[7]	start T_8 timeout; set slot=$7FFF_h$; and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**arbitrate**
	Init=0_2; not matching **inventoried** flag; AND additional condition[7]	start T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
	otherwise[8]	stop T_8 timeout and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**ready**
QueryY[1]	Session matches inventory round	stop T_8 timeout	**ready**
	Session does not match inventory round	stop T_8 timeout	**secured**
QueryRep	Session matches inventory round	transition **inventoried**[2] from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**secured**
QueryAdjust	Session matches inventory round	transition **inventoried**[2] from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**secured**
ACK	correct handle	See *Table 6-18*	**secured**
	incorrect handle	–	**secured**
NAK	all	–	**arbitrate**
Req_RN	all	backscatter new RN16	**secured**

Command	Condition	Action	Next State
Select	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**secured**
Read	all	backscatter data	**secured**
ReadVar	all	backscatter data	**secured**
Write	all	backscatter header when done	**secured**
Kill (see also Figure 6-26)	supported security timeout AND within timeout	backscatter error code	**secured**
	1st *Kill* received AND password-based kill	backscatter handle	**secured**
	2nd *Kill* received; password-based kill; AND correct nonzero kill password	backscatter header when done	**killed**
	2nd *Kill* received; password-based kill; AND incorrect nonzero kill password	may start security timeout	**arbitrate**
	2nd *Kill* received; password-based kill; AND kill password=0	backscatter error code	**secured**
	authenticated kill; prior Interrogator authentication; AND AuthKill privilege	backscatter response when done	**killed**
	authenticated kill AND (no prior Interrogator authentication OR no AuthKill privilege)	backscatter error code and may start security timeout	**arbitrate**
Lock	all	backscatter header when done	**secured**
Access (see also Figure 6-28)	supported security timeout AND within timeout	backscatter error code	**secured**
	1st *Access* received	backscatter handle	**secured**
	2nd *Access* received AND correct access password	backscatter handle	**secured**
	2nd *Access* received AND incorrect access password	may start security timeout	**arbitrate**
BlockWrite	all	backscatter header when done	**secured**
BlockErase	all	backscatter header when done	**secured**
BlockPermalock	Read/Lock=0_2	backscatter permalock bits	**secured**
	Read/Lock=1_2	backscatter header when done	**secured**
Challenge	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
Authenticate	supported security timeout AND within timeout	backscatter error code	**secured**
	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**

Command	Condition	Action	Next State
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
	executable AND new authentication sequence	reset cryptographic engine	**open**
AuthComm	supported security timeout AND within timeout	backscatter error code	**secured**
	executable AND prior Interrogator authentication	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
SecureComm	supported security timeout AND within timeout	backscatter error code	**secured**
	executable; prior Interrogator authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	see encapsulated command
	executable; prior Interrogator authentication; AND SenRep=1_2	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
ReadBuffer	**C**=1_2	backscatter data	**secured**
	C=0_2	backscatter error code	**secured**
KeyUpdate	supported security timeout AND within timeout	backscatter error code	**secured**
	executable; prior Interrogator authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable; prior Interrogator authentication; AND SenRep=1_2	backscatter response when done	**secured**
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
Untraceable	executable	backscatter header when done	**secured**
FileSetup	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
FileOpen	executable	close current file; open requested file; and backscatter file info	**secured**
FilePrivilege	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
TagPrivilege	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
FileList	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**

Command	Condition	Action	Next State
Faulty	unsupported parameters[3]	backscatter error code	**secured**
	incorrect handle[4]	none unless specified by cryptographic suite	**secured** or **open** [4]
	improper[5]	–	**arbitrate**
	invalid[6]	none unless specified by cryptographic suite	**secured** or **open** [6]

Note 1: *Query* or *QueryX* starts a new round and may change the session. *Query*, *QueryX* with Init=1_2, and *QueryY* with Init=1_2 also instructs a Tag to load a new random value into its slot counter.

Note 2: As described in *6.3.2.10*, a Tag transitions its **inventoried** flag prior to evaluating the condition.

Note 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 WordPtr, BitCount, SenRep or 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 6-29*); (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.

Note 4: "Incorrect handle" shall mean an *ACK* or 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.

Note 5: "Improper" shall mean a command (except *Req_RN*, *Query*, or *QueryX*) 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 6-26* and *Figure 6-28*).

Note 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.

Note 7: Additional condition: (MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel.

Note 8: Otherwise condition: not-matching Sel; (Init=1_2 AND not-matching **inventoried** flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2.

B.7 Present state: Killed

Table B-7: Killed state-transition table

Command	Condition	Action	Next State
Query	all	–	**killed**
QueryX	all	–	**killed**
QueryY	all	–	**killed**
QueryRep	all	–	**killed**
QueryAdjust	all	–	**killed**
ACK	all	–	**killed**
NAK	all	–	**killed**
Req_RN	all	–	**killed**
Select	all	–	**killed**
Read	all	–	**killed**
ReadVar	all	–	**killed**
Write	all	–	**killed**
Kill	all	–	**killed**
Lock	all	–	**killed**
Access	all	–	**killed**
BlockWrite	all	–	**killed**
BlockErase	all	–	**killed**
BlockPermalock	all	–	**killed**
Challenge	all	–	**killed**
Authenticate	all	–	**killed**
AuthComm	all	–	**killed**
SecureComm	all	–	**killed**
ReadBuffer	all	–	**killed**
KeyUpdate	all	–	**killed**
Untraceable	all	–	**killed**
FileSetup	all	–	**killed**
FileOpen	all	–	**killed**
FilePrivilege	all	–	**killed**
TagPrivilege	all	–	**killed**
FileList	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. See 5.3 for notation conventions.

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: *Query* [1] command-response table

Starting State	Condition	Response	Next State
ready, **arbitrate**, **reply**	slot=0; matching **inventoried** flag; AND matching **SL** flag	backscatter new RN16	**reply**
	slot≠0; matching **inventoried** flag; AND matching **SL** flag	–	**arbitrate**
	Otherwise	–	**ready**
acknowledged, **open**, **secured**	slot=0; matching **inventoried**[2] flag; AND matching **SL** flag	backscatter new RN16 and (transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session)	**reply**
	slot≠0; matching **inventoried**[2] flag; AND matching **SL** flag	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**arbitrate**
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new Session matches prior Session	**ready**
killed	all	–	**killed**

Note 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.

Note 2: As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

C.3 Command response: *QueryX*

Table C-3: *QueryX* [1] command-response table

<table>
<tr><th>Starting State</th><th colspan="2">Condition[2]</th><th>Response</th><th>Next State</th></tr>
<tr><td rowspan="5">ready, arbitrate, reply</td><td rowspan="4">(MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel</td><td>Init=1_2; matching inventoried flag; AND slot=0</td><td>backscatter new RN16 if ReplyCRC=0_2, or backscatter new RN16||CRC-5 if ReplyCRC=1_2</td><td>reply[3]</td></tr>
<tr><td>Init=1_2; matching inventoried flag; AND slot≠0</td><td>–</td><td>arbitrate[3]</td></tr>
<tr><td>Init=0_2 AND matching inventoried flag</td><td>–</td><td>arbitrate[5]</td></tr>
<tr><td>Init=0_2 AND not-matching inventoried flag</td><td>–</td><td>ready[4]</td></tr>
<tr><td colspan="2">not-matching Sel; (Init=1_2 AND not-matching inventoried flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2</td><td>–</td><td>ready[3]</td></tr>
<tr><td rowspan="5">acknowledged, open, secured</td><td rowspan="4">(MemBank≠00_2 OR Flx=0_2); (AckData=01_2 OR AckData=10_2); supported DBLF; AND matching Sel</td><td>Init=1_2; matching inventoried flag; AND slot=0</td><td>(backscatter new RN16 if ReplyCRC=0_2 or backscatter new RN16||CRC-5 if ReplyCRC=1_2) and (transition inventoried[2] from A→B or B→A if and only if new Session matches prior Session)</td><td>reply[3]</td></tr>
<tr><td>Init=1_2; matching inventoried flag; AND slot≠0</td><td>transition inventoried[2] from A→B or B→A if and only if new Session matches prior Session</td><td>arbitrate[3]</td></tr>
<tr><td>Init=0_2 AND matching inventoried flag</td><td>transition inventoried from A→B or B→A if and only if new Session matches prior Session</td><td>arbitrate[5]</td></tr>
<tr><td>Init=0_2 AND not-matching inventoried flag</td><td>transition inventoried from A→B or B→A if and only if new Session matches prior Session</td><td>ready[4]</td></tr>
<tr><td colspan="2">not-matching Sel; (Init=1_2 AND not-matching inventoried flag); AckData=00_2; AckData=11_2; unsupported DBLF; OR MemBank=00_2</td><td>transition inventoried from A→B or B→A if and only if new Session matches prior Session</td><td>ready[3]</td></tr>
<tr><td>killed</td><td colspan="2">All</td><td>–</td><td>killed</td></tr>
</table>

Note 1: *QueryX* (in any state other than **killed**) initializes a new inventory round and may change the session; *QueryX* also instructs a Tag to load a new random value into its slot counter. See *Figure 6-24* and *Figure 6-25*.

Note 2: As described in *6.3.2.10*, a Tag transitions its **inventoried** flag prior to evaluating the condition.

Note 3: Tag stops T_8 timeout.

Note 4: Tag starts T_8 timeout.

Note 5: Tag starts T_8 timeout and sets slot=$7FFF_h$.

C.4 Command response: *QueryY*

Table C-4: *QueryY*[1] command-response table

Starting State	Condition[2]		Response	Next State
ready, **arbitrate**	delimiter received before T_8 timeout expires; (MemBank≠00_2 OR Flx=0_2); AND Session matches inventory round	Init=1_2; matching **inventoried** flag; AND slot=0	backscatter new RN16 if ReplyCRC=0_2 in *QueryX*, or backscatter new RN16||CRC-5 if ReplyCRC=1_2 in *QueryX*	**reply**[3]
		Init=1_2; matching **inventoried** flag; AND slot≠0	–	**arbitrate**[3]
		Init=0_2 AND matching **inventoried** flag	–	**arbitrate**[5]
		Init=0_2 AND not-matching **inventoried** flag	–	**ready**[4]
		Init=1_2 AND not-matching **inventoried** flag	–	**ready**[3]
	delimiter received when T_8 timeout inactive; Session does not match inventory round; OR MemBank=00_2		–	**ready**[3]
reply	Session matches inventory round		–	**ready**[3]
	Session does not match inventory round		–	**reply**[3]
acknowledged	Session matches inventory round		–	**ready**[3]
	Session does not match inventory round		–	**acknowledged**[3]
open	Session matches inventory round		–	**ready**[3]
	Session does not match inventory round		–	**open**[3]
secured	Session matches inventory round		–	**ready**[3]
	Session does not match inventory round		–	**secured**[3]
killed	all		–	**killed**

Note 1: *QueryY* (in any state other than **killed**) refines a filtering criteria after initiating a new inventory round with *QueryX*; *QueryY* also instructs a Tag to load a new random value into its slot counter. See *Figure 6-24* and *Figure 6-25*.

Note 2: Intentionally left blank.

Note 3: Tag stops T_8 timeout.

Note 4: Tag starts T_8 timeout.

Note 5: Tag starts T_8 timeout and sets slot=$7FFF_h$.

C.5 Command response: *QueryRep*

Table C-5: *QueryRep* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate	Session matches inventory round AND slot=0 after decrementing slot counter	decrement slot counter and (backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round)	**reply**
	Session matches inventory round AND slot≠0 after decrementing slot counter	decrement slot counter	**arbitrate**
	Session does not match inventory round	–	**arbitrate**
reply	Session matches inventory round	–	**arbitrate**
	Session does not match inventory round	–	**reply**
acknowledged	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**acknowledged**
open	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**open**
secured	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**secured**
killed	all	–	**killed**

C.6 Command response: *QueryAdjust*

Table C-6: *QueryAdjust* [1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate	Session matches inventory round AND slot=0	backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round	**reply**
	Session matches inventory round AND slot≠0	–	**arbitrate**
	Session does not match inventory round	–	**arbitrate**
reply	Session matches inventory round AND slot=0	backscatter new RN16 if *QueryX* with ReplyCRC=0_2 began inventory round or backscatter new RN16||CRC-5 if *QueryX* with ReplyCRC=1_2 began inventory round	**reply**
	Session matches inventory round AND slot≠0	–	**arbitrate**
	Session does not match inventory round	–	**reply**
acknowledged	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**acknowledged**
open	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**open**
secured	Session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
	Session does not match inventory round	–	**secured**
killed	all	–	**killed**

Note 1: *QueryAdjust*, in the **arbitrate** or **reply** states, instructs a Tag to load a new random value into its slot counter.

C.7 Command response: *ACK*

Table C-7: *ACK* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate	all	–	**arbitrate**
reply, acknowledged	correct RN16	see *Table 6-18*	**acknowledged**
	incorrect RN16	–	**arbitrate**
open[1]	all	see *Table 6-18*	**open**
secured[1]	all	see *Table 6-18*	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle.

C.8 Command response: *NAK*

Table C-8: *NAK* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**, **open**, **secured**	all	–	**arbitrate**
killed	all	–	**killed**

C.9 Command response: *Req_RN*

Table C-9: *Req_RN* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**	all	–	**arbitrate**
acknowledged	correct RN16 AND access password≠0	backscatter handle	**open**
	correct RN16 AND access password=0	backscatter handle	**secured**
	incorrect RN16	–	**acknowledged**
open[1]	all	backscatter new RN16	**open**
secured[1]	all	backscatter new RN16	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle.

C.10 Command response: *Select*

Table C-10: *Select* command-response table

Starting State	Condition	Response	Next State
ready	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**ready**
arbitrate	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**arbitrate**
reply	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**reply**
acknowledged	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**acknowledged**
open	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**open**
secured	correct parameters	(assert or deassert **SL**) or (set **inventoried** to *A* or *B*)	**ready**
	incorrect parameters	–	**secured**
killed	all	–	**killed**

C.11 Command response: *Read*

Table C-11: *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**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.12 Command response: *ReadVar*

Table C-12: *ReadVar* 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**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.13 Command response: *Write*

Table C-13: *Write* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	all	backscatter header when done	**open**
secured[1]	all	backscatter header when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.14 Command response: *Kill*

Table C-14: *Kill* [1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[2]	supported security timeout AND within timeout	backscatter error code	**open**
	1st *Kill* received AND password-based kill	backscatter handle	**open**
	2nd *Kill* received; password-based kill; AND correct nonzero kill password	backscatter header when done	**killed**
	2nd *Kill* received; password-based kill; AND incorrect nonzero kill password	may set security timeout	**arbitrate**
	2nd *Kill* received; password-based kill; AND kill password=0	backscatter error code	**open**
	authenticated kill	backscatter error code; may set security timeout	**arbitrate**
secured[2]	supported security timeout AND within timeout	backscatter error code	**secured**
	1st *Kill* received AND password-based kill	backscatter handle	**secured**
	2nd *Kill* received; password-based kill; AND correct nonzero kill password	backscatter header when done	**killed**
	2nd *Kill* received; password-based kill; AND incorrect nonzero kill password	may set security timeout	**arbitrate**
	2nd *Kill* received; password-based kill; AND kill password=0	backscatter error code	**secured**
	authenticated kill; prior Interrogator authentication; AND AuthKill privilege	backscatter response when done	**killed**
	authenticated kill AND (no prior Interrogator authentication OR no AuthKill privilege)	backscatter error code; may set security timeout	**arbitrate**
killed	all	–	**killed**

Note 1: See also *Figure 6-26*.

Note 2: See *Table C-34* for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.15 Command response: *Lock*

Table C-15: *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 header when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.16 Command response: *Access*

Table C-16: *Access* [1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[2]	supported security timeout AND within timeout	backscatter error code	**open**
	1st *Access* received	backscatter handle	**open**
	2nd *Access* received AND correct access password	backscatter handle	**secured**
	2nd *Access* received AND incorrect access password	may set security timeout	**arbitrate**
secured[2]	supported security timeout AND within timeout	backscatter error code	**secured**
	1st *Access* received	backscatter handle	**secured**
	2nd *Access* received AND correct access password	backscatter handle	**secured**
	2nd *Access* received AND incorrect access password	may set security timeout	**arbitrate**
killed	all	–	**killed**

Note 1: See also *Figure 6-28*.

Note 2: See *Table C-34* for the Tag response to an incorrect handle or an improper command.

C.17 Command response: *BlockWrite*

Table C-17: *BlockWrite* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	all	backscatter header when done	**open**
secured[1]	all	backscatter header when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.18 Command response: *BlockErase*

Table C-18: *BlockErase* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	all	backscatter header when done	**open**
secured[1]	all	backscatter header when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.19 Command response: *BlockPermalock*

Table C-19: *BlockPermalock* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open	all	–	**open**
secured[1]	Read/Lock=0_2	backscatter permalock bits	**secured**
	Read/Lock=1_2	backscatter header when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.20 Command response: *Challenge*

Table C-20: *Challenge* command-response table

Starting State	Condition	Response	Next State
ready, **arbitrate**, **reply**, **acknowledged**, **open**, **secured**	(supported security timeout AND within timeout); unsupported value of Immed; unsupported value of CSI; not-executable Message; cryptographic error; OR nonzero RFU bits	set **C**=0_2	**ready**
	supported value of Immed; supported value of CSI; AND executable Message	store result and set **C**=1_2	**ready**
killed	all	–	**killed**

C.21 Command response: *Authenticate*

Table C-21: *Authenticate* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	supported security timeout AND within timeout	backscatter error code	**open**
	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**open** or **secured**[2]
	executable AND SenRep=1_2	backscatter response when done	**open** or **secured**[2]
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
	executable AND new authentication sequence	reset cryptographic engine	**open**
secured[1]	supported security timeout AND within timeout	backscatter error code	**secured**
	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
	executable AND new authentication sequence	reset cryptographic engine	**open**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

Note 2: See cryptographic suite.

C.22 Command response: *AuthComm*

Table C-22: *AuthComm* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	supported security timeout AND within timeout	backscatter error code	**open**
	executable AND prior Tag authentication	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	**open**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
secured[1]	supported security timeout AND within timeout	backscatter error code	**secured**
	executable AND prior Interrogator authentication	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.23 Command response: *SecureComm*

Table C-23: *SecureComm* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	supported security timeout AND within timeout	backscatter error code	**open**
	executable; prior Tag authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	see encapsulated command
	executable; prior Tag authentication; AND SenRep=1_2	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	**open**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
secured[1]	supported security timeout AND within timeout	backscatter error code	**secured**
	executable; prior Interrogator authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	see encapsulated command
	executable; prior Interrogator authentication; AND SenRep=1_2	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.24 Command response: *ReadBuffer*

Table C-24: *ReadBuffer* command-response table

Starting State	Condition	Response	Next State
Ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	**C**=1_2	backscatter data	**open**
	C=0_2	backscatter error code	**open**
secured[1]	**C**=1_2	backscatter data	**secured**
	C=0_2	backscatter error code	**secured**
Killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.25 Command response: *KeyUpdate*

Table C-25: *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 AND within timeout	backscatter error code	**secured**
	executable; prior Interrogator authentication; AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable; prior Interrogator authentication; AND SenRep=1_2	backscatter response when done	**secured**
	no prior Interrogator authentication	backscatter error code	**secured**
	not-executable OR cryptographic error	see cryptographic suite	**arbitrate**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.26 Command response: *Untraceable*

Table C-26: *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 header when done	**secured**
Killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.27 Command response: *FileSetup*

Table C-27: *FileSetup* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open	all	–	**open**
secured[1]	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.28 Command response: *FileOpen*

Table C-28: *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; and backscatter file info	**open**
secured[1]	executable	close current file; open requested file; and backscatter file info	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.29 Command response: *FilePrivilege*

Table C-29: *FilePrivilege* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open	all	–	**open**
secured[1]	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.30 Command response: *TagPrivilege*

Table C-30: *TagPrivilege* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open	all	–	**open**
secured[1]	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.31 Command response: *FileList*

Table C-31: *FileList* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, **reply**, **acknowledged**	all	–	**arbitrate**
open[1]	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**open**
	executable AND SenRep=1_2	backscatter response when done	**open**
secured[1]	executable AND SenRep=0_2	store result; set **C**=1_2; and backscatter response when done	**secured**
	executable AND SenRep=1_2	backscatter response when done	**secured**
killed	all	–	**killed**

Note 1: See *Table C-34* for the Tag response to an incorrect handle or unsupported parameters.

C.32 Command response: T_2 timeout

Table C-32: 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.33 Command response: T_8 timeout

Table C-33: T_8 timeout command-response table[1]

Starting State	Condition	Response	Next State
ready, **arbitrate**	T_8 timeout expires	stop T_8 timeout	**ready**
	delimiter of not-exempt command[2] received before T_8 timeout expires	stop T_8 timeout and ignore command	**ready**
	delimiter of exempt command received before T_8 timeout expires[2]	stop T_8 timeout and execute command	see command
reply	all	–	**reply**
acknowledged	all	–	**acknowledged**
open	all	–	**open**
secured	all	–	**secured**
killed	all	–	**killed**

Note 1: see *Figure 6-18*, *Table 6-16*, *Figure 6-24* and *Figure 6-25*.

Note 2: Exempt commands are *Select*, *Challenge*, *Query*, *QueryX*, and *QueryY*. All other commands, including faulty commands, are not-exempt.

C.34 Command response: Faulty command

Table C-34: 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 handle[4]	none unless specified by cryptographic suite	**open**
	improper[6]	–	**arbitrate**
	invalid[7]	none unless specified by cryptographic suite	**open**
secured	unsupported parameters[3]	backscatter error code	**secured**
	incorrect handle[5]	none unless specified by cryptographic suite	**secured** or **open**[9]
	improper[6]	–	**arbitrate**
	invalid[8]	none unless specified by cryptographic suite	**secured** or **open**[9]
killed	all	–	**killed**

Note 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.

Note 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 value in RFU parameter or RFU value used in a parameter; (2) an unsupported WordPtr, BitCount, SenRep or 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 6-29*); (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 or parameter values not supported by the Tag.

Note 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 value in RFU parameter or RFU value used in a parameter; (2) an unsupported WordPtr, BitCount, SenRep or 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 unallocatable memory; (8) an unencrypted Message that requires encryption; (9) a command that requires encapsulation but is nonetheless unencapsulated (see *Table 6-29*); (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 or parameter values not supported by the Tag.

Note 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.

Note 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 cryptographic engine and transition to the **open** state upon receiving a security command with an incorrect handle.

Note 6: "Improper" shall mean a command (except *Req_RN, Query*, or *QueryX*) 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 6-26* and *Figure 6-28*).

Note 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.

Note 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.

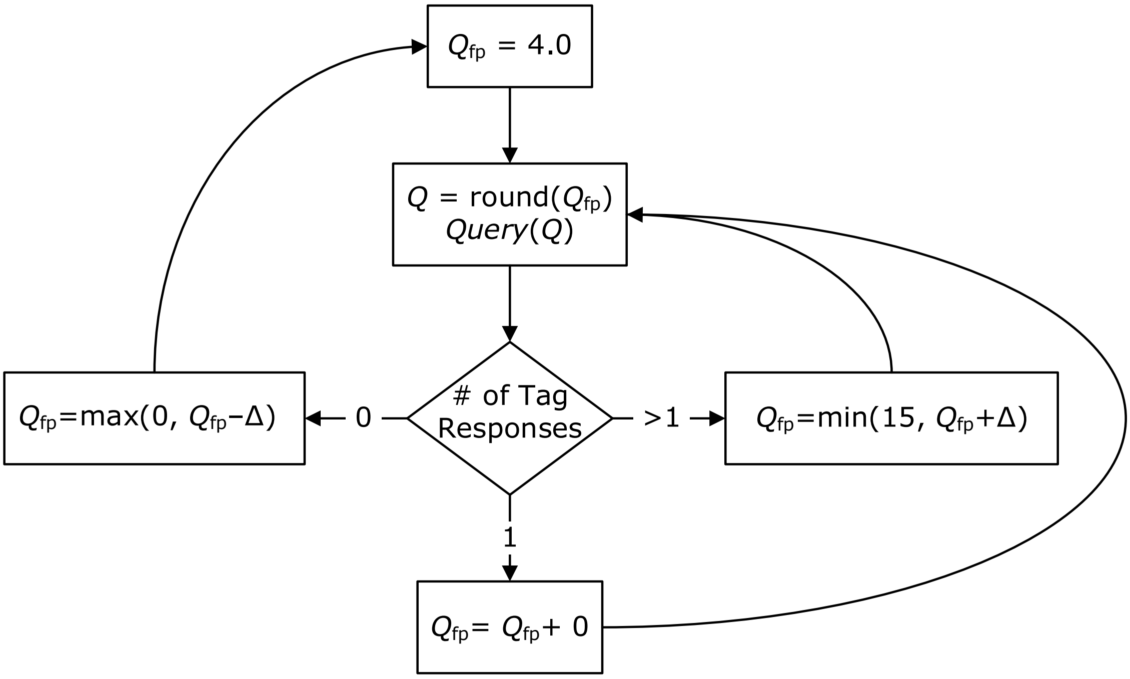
Note 9: See cryptographic suite.

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*

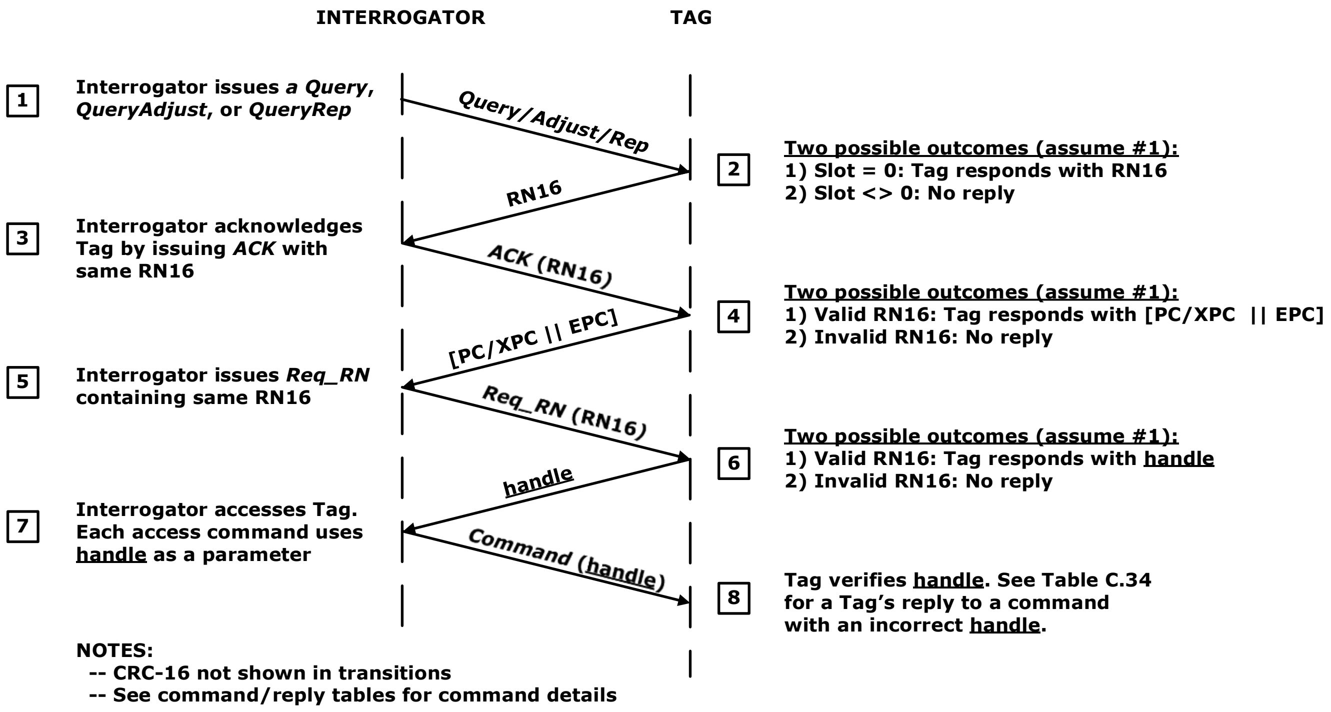


Annex E
(informative)
Example Tag inventory and access

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 using *Query*.

Figure E-1: Example of Tag inventory and access



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 $x^5 + x^3 + 1$ and preset defined in *Table 6-12*.

To calculate a CRC-5, first preload the entire CRC register (i.e. Q[4:0], Q4 being the MSB and Q0 the LSB) with the value 01001_2 (see Table F-1), then clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, Q[4:0] holds the CRC-5 value.

To check a CRC-5, first preload the entire CRC register (Q[4:0]) with the value 01001_2, 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] = 00000_2.

Figure F-1: Example CRC-5 circuit

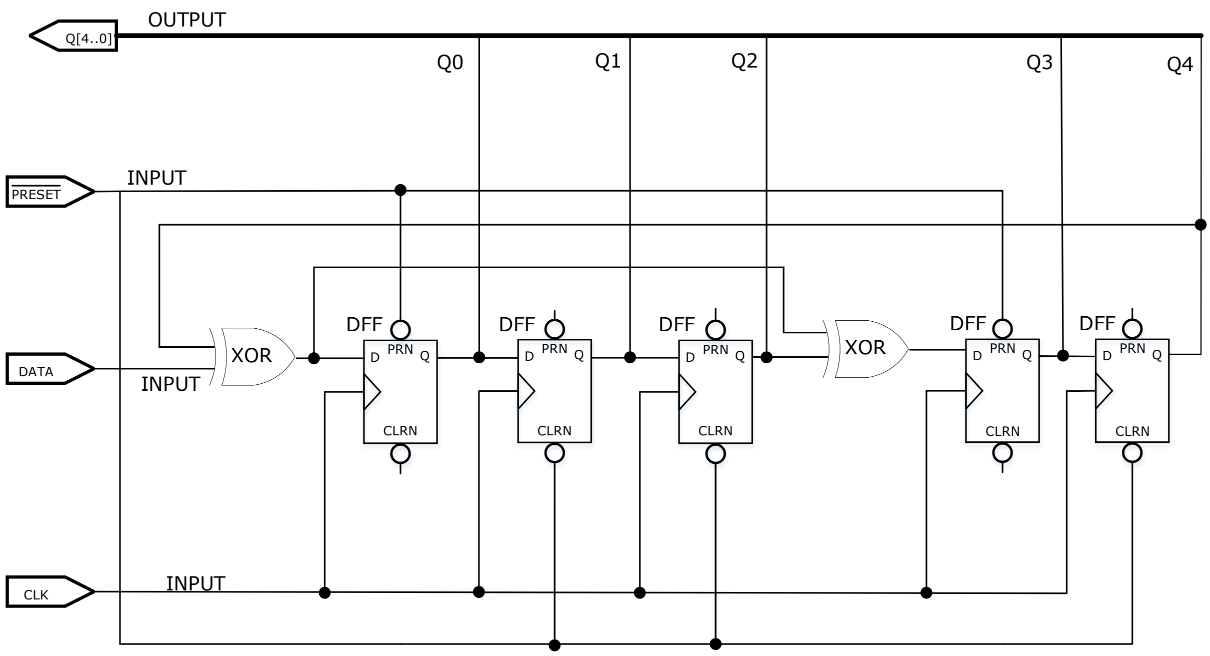


Table F-1: CRC-5 register preload values

Register	Preload value (bits)
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).

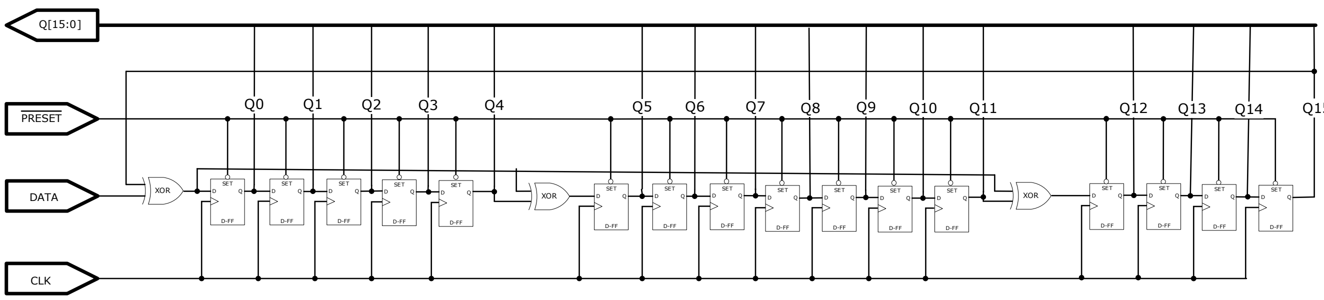
To calculate a CRC-16, first preload the entire CRC register (i.e. Q[15:0], Q15 being the MSB and Q0 the LSB) with the value FFFF$_h$. Second, clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, Q[15:0] holds the ones-complement of the CRC-16. 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



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$_h$	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$_h$
70$_h$	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 region the regulations may have 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 local regulations.

When an Interrogator in a multiple- or dense-Interrogator environment instructs Tags to use subcarrier backscatter, the Interrogator shall adopt the channel plan for the regulatory region in which it is operating.

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, then 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. Three frequency-division multiplexing (FDM) methods minimize Interrogator-on-Tag collisions by spectrally separating Interrogator transmissions and Tag responses:

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 dense-Interrogator methods using alternative-channel backscatter.

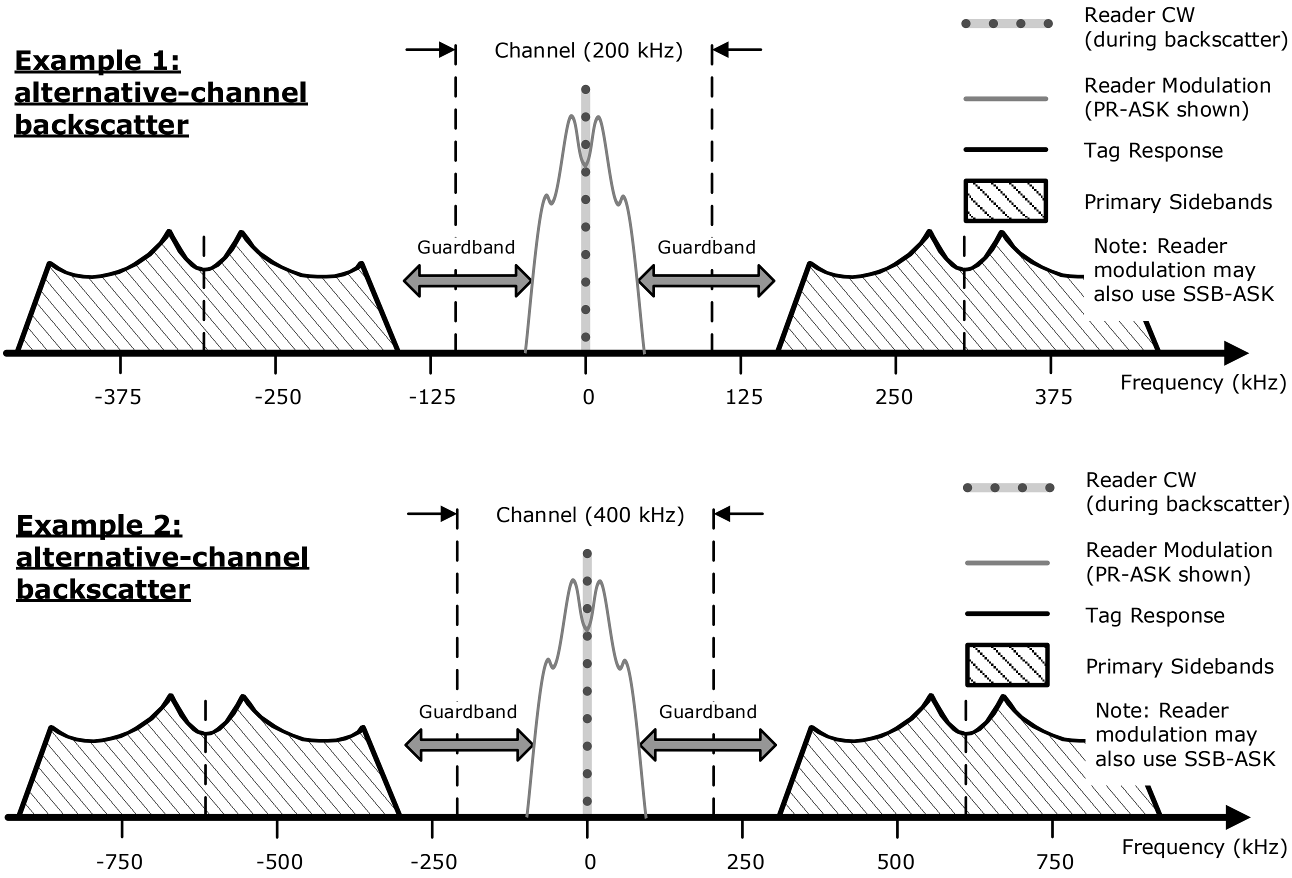
Example 1: Alternative-channel backscatter

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 1 of *Figure G-1* shows Interrogator transmissions using PR-ASK modulation with Tari = 25 µs, and 75 kbps Tag data backscatter on a 300 kHz subcarrier (BLF = 300 kHz, M = 4).

Example 2: Alternative-channel backscatter

At the time of this protocol ETSI EN 302 208 allows four high-power 400 kHz channels, each spaced 1200 kHz apart, in the 916–921 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 PR-ASK modulation with Tari = 12.5 µs, and 150 kbps Tag data backscatter on a 600 kHz subcarrier (BLF = 600 kHz, M = 4).

Figure G-1: Examples of dense-Interrogator-mode operation

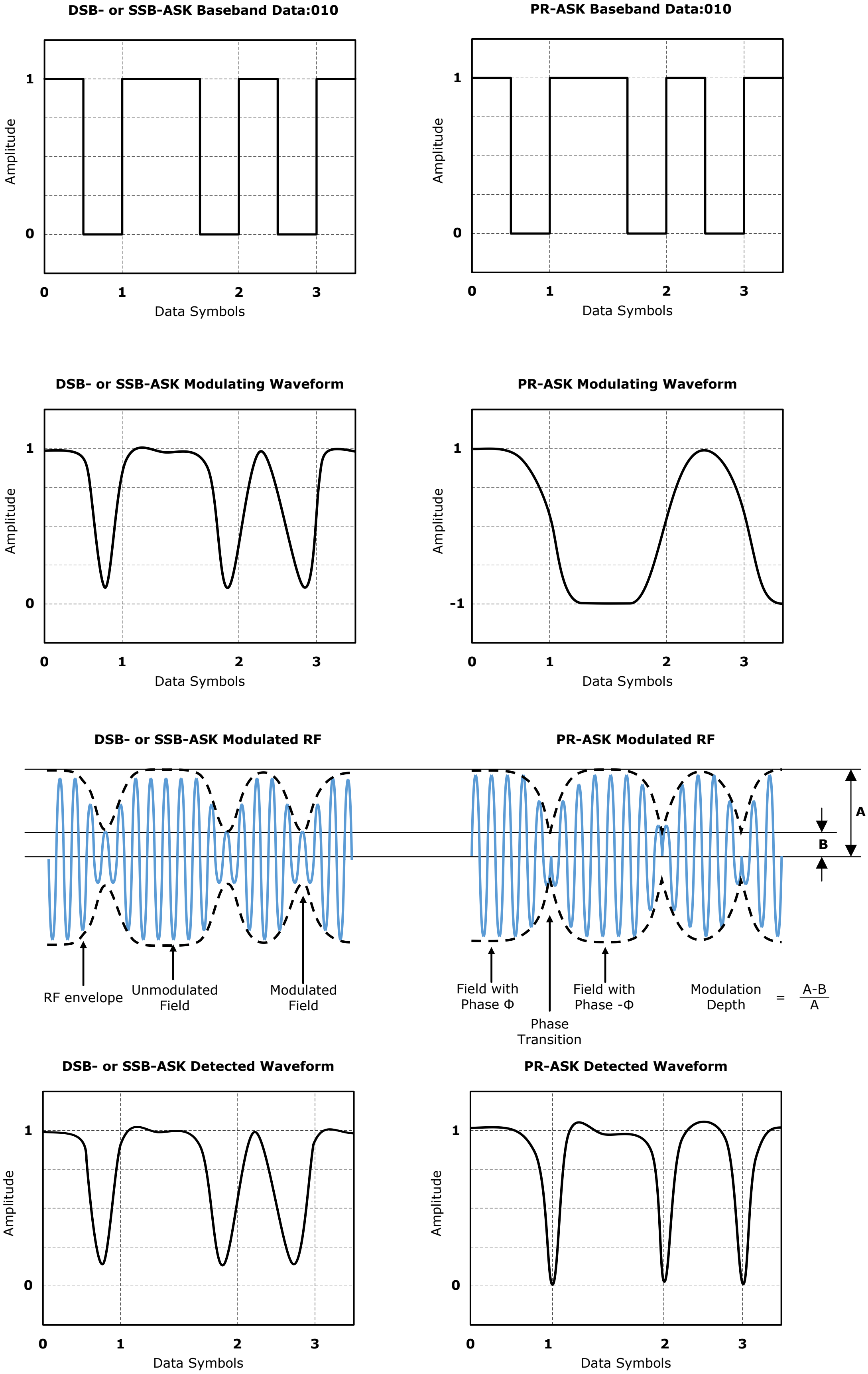


Annex H
(informative)
Interrogator-to-Tag link modulation

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.

Figure H-1: Interrogator-to-Tag 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 00001111_2 (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 Tag shall backscatter only one of these error codes.
- The header for an error reply is 1_2, unlike the header for a success response which is 0_2.

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 writable[1] or not readable.
	00000101_2	Cryptographic 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 *AuthComm* or *SecureComm* 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
	00010001_2	Reserved for ISO/IEC 18000-63	
	00010010_2	Reserved for ISO/IEC 18000-63	
	00010011_2	Reserved for ISO/IEC 18000-63	
Non-specific	00001111_2	Non-specific error	The Tag does not support error-specific codes

Note 1: Read-only memory is, by definition, not writable.

Annex J
(normative)
Slot counter

J.1 Slot-counter operation

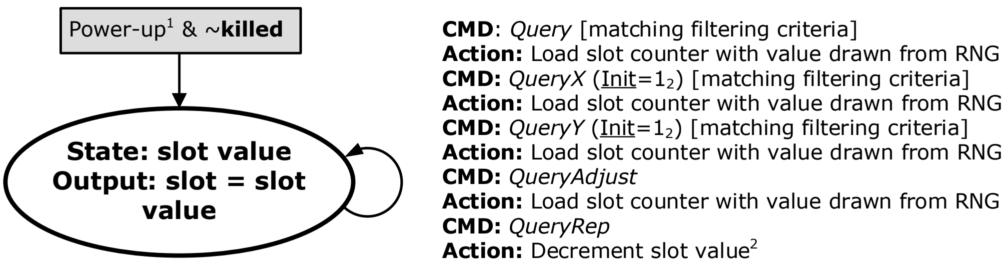
As described in 6.3.2.6.8, Tags implement a 15-bit slot counter. As described in 6.3.2.10, Interrogators use the slot counter to regulate the probability of a Tag responding to a *Query*, *QueryX*, *QueryY*, *QueryAdjust*, or *QueryRep* command. Upon receiving a *Query*, *QueryX* with Init=1_2, *QueryY* with Init=1_2, or *QueryAdjust* a Tag loads a value between 0 and 2^Q-1, 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* or *QueryX* specifies *Q*; a *QueryAdjust* may modify *Q* from the prior *Query* or *QueryX*.

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*, *QueryX*, or *QueryY* 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*, *QueryX* with Init=1_2, *QueryY* with Init=1_2, 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 and the StoredPC 15_h-$1F_h$ is set to zero.
- The Tag's access password is $ACCEC0DE_h$.
- The Tag's kill password is $DEADC0DE_h$.
- The 1st half of the access password (password bit addresses 31-16) XORed with RN16_2 = $ACCE_h \otimes 1602_h = BACC_h$.
- The 2nd half of the access password (password bit addresses 15-0) XORed with RN16_3 = $C0DE_h \otimes 1603_h = D6DD_h$.

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 (see *Table 6-61*), 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–$3F_h$	$C0DE_h$
	access password[31:16]	20_h–$2F_h$	$ACCE_h$
	kill password[15:0]	10_h–$1F_h$	$C0DE_h$
	kill password[31:16]	00_h–$0F_h$	$DEAD_h$

Table K-2: Lock-field values

Kill Password		Access Password		EPC Memory		TID Memory		File_0 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 6-28* with a *Read* command added at the end. The sequence of Interrogator commands and Tag replies is as follows (for reasons of clarity, command codes and Tag reply headers are not shown):

Step 1a: *Req_RN*[RN16_0||CRC-16]

Step 1b: Tag backscatters RN16_1||CRC-16 (RN16_1 becomes the handle for the entire *Access* procedure)

Step 2a: *Req_RN*[handle||CRC-16]

Step 2b: Tag backscatters RN16_2||CRC-16

Step 3a: *Access*[access password[31:16] XORed with RN16_2||handle||CRC-16]

Step 3b: Tag backscatters handle||CRC-16

Step 4a: *Req_RN*[handle||CRC-16]

Step 4b: Tag backscatters RN16_3||CRC-16

Step 5a: *Access*[access password[15:0] XORed with RN16_3||handle||CRC-16]

Step 5b: Tag backscatters handle||CRC-16

Step 6a: *Read*[MemBank=Reserved||WordPtr=00_h||WordCount=2||handle||CRC-16]

Step 6b: Tag backscatters kill password||handle||CRC-16

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: *Req_RN* command	R => T	11000001	0001 0110 0000 0000 (RN16_0=1600_h)	**acknowledged → open**
1b: Tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	
2a: *Req_RN* command	R => T	11000001	0001 0110 0000 0001 (handle=1601_h)	**open → open**
2b: Tag response	T => R		0001 0110 0000 0010 (RN16_2=1602_h)	
3a: *Access* command	R => T	11000110	1011 1010 1100 1100 ($BACC_h$) 0001 0110 0000 0001 (handle=1601_h)	**open → open**
3b: Tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	
4a: *Req_RN* command	R => T	11000001	0001 0110 0000 0001 (handle=1601_h)	**open → open**
4b: Tag response	T => R		0001 0110 0000 0011 (RN16_3=1603_h)	
5a: *Access* command	R => T	11000110	1101 0110 1101 1101 ($D6DD_h$) 0001 0110 0000 0001 (handle=1601_h)	**open → secured**
5b: Tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	
6a: *Read* command	R => T	11000010	00 (MemBank=Reserved) 00000000 (WordPtr=kill password) 00000010 (WordCount=2) 0001 0110 0000 0001 (handle=1601_h)	**secured → secured**
6b: Tag response	T => R		0 (header) 1101 1110 1010 1101 ($DEAD_h$) 1100 0000 1101 1110 ($C0DE_h$)	

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, FileNum, and UWC 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 *6.3.2.12.3.7*.
- *BlockWrite*: A Tag may support the BlockWrite command. See *6.3.2.12.3.8*.
- *BlockErase*: A Tag may support the BlockErase command. See *6.3.2.12.3.9*.
- *BlockPermalock*: A Tag may support the BlockPermalock command. See *6.3.2.12.3.10*.
- *Authenticate*: A Tag may support the Authenticate command. See *6.3.2.12.3.11*.
- *AuthComm*: A Tag may support the AuthComm command. *6.3.2.12.3.12*.
- *SecureComm*: A Tag may support the SecureComm command. See *6.3.2.12.3.13*.
- *KeyUpdate*: A Tag may support the KeyUpdate command. See *6.3.2.12.3.14*.
- *TagPrivilege*: A Tag may support the TagPrivilege command. See *6.3.2.12.3.15*.
- *ReadBuffer*: A Tag may support the ReadBuffer command. See *6.3.2.12.3.16*.
- *Untraceable*: A Tag may support the Untraceable command. See *6.3.2.12.3.17*.
- *FileOpen*: A Tag may support the FileOpen command. See *6.3.2.12.3.18*.
- *FileList*: A Tag may support the FileList command. See *6.3.2.12.3.19*.
- *FilePrivilege*: A Tag may support the FilePrivilege command. See *6.3.2.12.3.20*.

- *FileSetup*: A Tag may support the FileSetup command. See 6.3.2.12.3.21.

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.
- A Tag may support Immed=0_2 in the *Challenge* command.
- A Tag may support SenRep=0_2 in the *Authenticate*, *SecureComm*, and *KeyUpdate* commands.

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: a Tag may be used in a GS1 EPCglobal or a non-GS1 EPCglobal Application (**T**=0_2 or **T**=1_2 respectively). If **T**=0_2 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_2 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**, **SA**, **SS**, **FS**, **SN**, **B**, **C**, **SLI**, **TN**, **U**, **K**, **NR**, and **H** flags, corresponding to whether the Tag has a nonzero XPC_W2, a sensor alarm, a simple sensor, a full-function sensor, a snapshot sensor, 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–213_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 *6.3.2.12.3.5*.

A Tag may store its lock bits in a readable or an unreadable memory location. See *6.3.2.12.3.6*.

A Tag may support non-zero WordPtr values and non-zero BitCount values for *ReadBuffer* commands.

A Tag may support BAP functionality. See *7*.

A Tag may support sensor functionality. See *8*.

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 CSI.	
Message fields for commands supported by the cryptographic suite from the set *Challenge*, *Authenticate*, *AuthComm*, *SecureComm*, and *KeyUpdate*.	
Functional descriptions of and bit fields for all parameters, formatting, encoding, decoding, and integrity protection used in or by all Message and responses 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 KeyID in a Message field.	
Whether Message and/or responses include an explicit step count.	
Whether the cryptographic suite requires a *KeyUpdate* to be encapsulated in a *SecureComm* or an *AuthComm*.	
The formatting of every Message and every response 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:	n/a
▪ 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-electronic article surveillance, 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 Electronic Article Surveillance	Tag Alteration (Core)	Consumer Electronics
6.3.2.1.3 • 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* • 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* • 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
6.3.2.12.3.17 *Untraceable* command. A Tag may, but is not required to support all *Untraceable* field values	*6.3.2.12.3.17* *Untraceable* command. A Tag may, but is not required to support all *Untraceable* field values.	*6.3.2.12.3.10* *BlockPermalock* command
6.3.2.5 Security timeout for the *Access* command with a timeout range as specified in *Table 6-21* is mandatory	*6.3.2.5* Security timeout for the *Access* command with a timeout range as specified in *Table 6-21* is mandatory	*6.3.2.5* Security timeout for the *Access* command with a timeout range as specified in *Table 6-21* is mandatory
6.3.2.1.4.1 $\geq$32 bits User memory is mandatory	Intentionally left blank	*6.3.2.12.3.18* *FileOpen* command
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 *Challenge* command	*6.3.2.12.3.11* *Authenticate* command	*6.3.2.12.1.2* *Challenge* command
Intentionally left blank	Intentionally left blank	*6.3.2.12.3.8* *Blockwrite* command
Intentionally left blank	Intentionally left blank	*6.3.2.12.3.11* *Authenticate* command
Intentionally left blank	Intentionally left blank	*6.3.2.12.3.13* *SecureComm* command
Intentionally left blank	Intentionally left blank	*6.3.2.12.3.14* *KeyUpdate* command

Page intentionally blank